Nonlinear Dyn (2023) 111:17037–17077
https://doi.org/10.1007/s11071-023-08725-y

ORIGINAL PAPER

A deep reinforcement learning control approach for high-performance aircraft

Agostino De Marco · Paolo Maria D'Onza · Sabato Manfredi

Received: 26 September 2022 / Accepted: 25 May 2023 / Published online: 2 August 2023

Abstract This research introduces a flight controller for a high-performance aircraft, able to follow randomly generated sequences of waypoints, at varying altitudes, in various types of scenarios. The study assumes a publicly available six-degree-of-freedom (6-DoF) rigid aeroplane flight dynamics model of a military fighter jet. Consolidated results in artificial intelligence and deep reinforcement learning (DRL) research are used to demonstrate the capability to make certain manoeuvres AI-based fully automatic for a high-fidelity nonlinear model of a fixed-wing aircraft. This work investigates the use of a deep deterministic policy gradient (DDPG) controller agent, based on the successful applications of the same approach to other domains. In the particular application to flight control presented here, the effort has been focused on the design of a suitable reward function used to train the agent to achieve some given navigation tasks. The trained controller is successful on highly coupled manoeuvres, including rapid sequences of turns, at both low and high flight Mach numbers, in simulations reproducing a prey–chaser dogfight scenario. Robustness to sensor noise, atmospheric disturbances, different initial flight conditions and varying reference signal shapes is also demonstrated.

Keywords Deep reinforcement learning · Flight dynamics · UCAV · Aeroplane controllability · Nonlinear control

A. De Marco (✉) · P. M. D'Onza
Department of Industrial Engineering (DII), Università degli Studi di Napoli Federico II, Via Claudio, 21, 80125 Naples, Italy
e-mail: agostino.demarco@unina.it

P. M. D'Onza
e-mail: p.donza@studenti.unina.it

S. Manfredi (✉)
Department of Electrical Engineering and Information Technology (DIETI), Università degli Studi di Napoli Federico II, Via Claudio, 21, 80125 Naples, Italy
e-mail: sabato.manfredi@unina.it

1 Introduction

Within the next decade, we will witness an exponential increase in the use of artificial intelligence to support the operations of all sorts of flying machines. The interest will range from military applications to the growing civil aviation market, which will also include the new category of personal flying vehicles. This research makes use of consolidated results in artificial intelligence and deep reinforcement learning research to demonstrate the AI-based flight control of a high-performance fixed-wing aircraft, taken as an example of an aerial platform exhibiting nonlinear behaviour.

1.1 Nonlinear dynamics in fixed-wing aircraft and established engineering approach to flight control

Many current fixed-wing aerial vehicles display some nonlinear dynamics, such as inertial coupling, or aero-

dynamic and propulsive nonlinearities. Future aircraft are likely to have even more significant nonlinearities in their flight dynamics. For example, to enhance their agility some unmanned aerial vehicles (UAV) designs will feature bio-inspired morphing wings that significantly change their shapes. High-altitude long-endurance UAVs as well as future commercial airliner designs will have a high aspect ratio, and highly flexible lifting surfaces to improve their efficiency. Airbreathing hypersonic vehicles (AHV) are yet another example of plants exhibiting significant cross-coupling, nonlinearity, non-minimum phase behaviour, and model uncertainty, even more than traditional aircraft in terms of dynamics. All these designs make flight control a challenging problem.

An established engineering approach to design flight controllers is to rely on gain scheduling, to switch between different linear controllers that were tuned for a known set of specific operating points [1]. Because of their dependence on linearized plant dynamics, these gain scheduling techniques must be carefully designed to accomplish their tasks across the whole mission envelope. In practice, the classical linear controllers are generally used in a conservative fashion by setting some constraints to the flight envelopes and limiting the operability of the aerial vehicles. On the other hand, nonlinear control algorithms can be designed to attain the advantage of using the full flight envelope, especially for unconventional and disruptive aircraft configurations. Model-free adaptive control as well as intelligent control techniques provide the possibility to replace the classical controller structure with a more general approach, which turns out to be also fault-tolerant to changes in model dynamics [2].

1.2 Intelligent control approaches

Many authors propose intelligent control approaches that combine artificial intelligence and automatic control methodologies such as adaptive neural control [3] and fuzzy control [4]. Examples of application to flying vehicles include adaptive neural control of AHVs, integral sliding mode control, and stochastic adaptive attitude control [5], as well as prescribed performance dynamic neural network control of flexible hypersonic vehicles [6]. Yet these methods assume a simplified motion dynamic model that considers only the longitudinal dynamics and an affine form of the control input. An alternative adaptive neural control method has been proposed in [7] in order to better reflect the characteristics of the actual plant model ensuring the reliability of the designed control laws, still considering a reduced 3-DoF motion model.

Reinforcement learning (RL) is a branch of machine learning that offers an approach to intelligent control inspired by the way biological organisms learn their tasks. RL is quite a large subject and provides the means to design a wide range of control strategies. In essence, an RL-based control task is performed by an agent—that is, a piece of software—able to learn a control policy through trial and error. The agent learns by interacting with its environment, which includes the plant under study [8]. In its early implementations, RL was based on tabular methods with discrete action and state spaces and gained quite a lot of popularity for its applications to gaming environments. One notable early usage example of RL in the aviation domain has been the control of glider soaring [9]. This study demonstrated that a flight control policy to effectively gain lift from ascending thermal plumes could be successfully learnt.

1.3 Deep reinforcement learning as a promising field for nonlinear control

To address the curse of dimensionality encountered in RL approaches when making the action and state spaces larger, function approximators (typically neural networks, NNs) have been used in the past decade in various fields to enable continuous control, with agents implemented as structures known as actor-critic. The controller agent has a policy function and a value function to interact with its environment. By observing the state of the environment, the agent performs an action based on its current policy to change the environment state, and consequently, receives a reward. Both the policy function and the value function are represented by neural networks that have to be made optimal by the end of the learning process. The policy function is updated in a gradient ascent manner according to the estimated advantage of performing the action; the value function is updated in a supervised fashion to estimate the advantage based on the collected rewards. The nonlinear activation functions in neural networks can represent a highly nonlinear transfer from inputs to outputs, and deep reinforcement learning (DRL) is

in fact a promising field for nonlinear control. Through the actor-critic scheme, an agent can learn to control an aircraft model by maximizing the expected reward, and the reward calculation function is a crucial aspect of the agent training process. Particular forms of possible reward functions for the aircraft control scenarios studied in this work are discussed in the article.

Several novel implementations of the actor-critic paradigm have been recently introduced, and most of them fall under the category of DRL approaches. Their ability to perform end-to-end offline learning has made it possible to design agents that surpass human performance, as demonstrated with different Atari games [10]. In this study, the agent observes an image of the game, a high-dimensional space, and learns a policy using a deep Q-network (DQN) for its discrete action space. DRL was extended to continuous-time control applications with the off-policy deep deterministic policy gradient (DDPG) algorithm [11]. For its actor-network policy update, the DDPG algorithm uses a critic network that provides a value function approximator. This method also estimates the temporal difference of the action-state value function as well as the state value function during learning and is capable of reducing the variance of the gradient estimation.

DDPG flight control applications have been limited to small-scale flying wings [12] and quadcopters [13]. An on-policy RL algorithm known as proximal policy optimization (PPO) [14] has been introduced in order to reduce DDPG's learning instability. The PPO showed improved policy convergence when applied to the flight control of an unmanned flying-wing aircraft [15]. A refined algorithm was proposed in [16] to control underwater glider attitude based on a combination of an active disturbance rejection control (ADRC) algorithm and a natural actor-critic (NAC) algorithm. A recently developed actor-critic agent structure for closed-loop attitude control of a UAV is discussed in [17], which addresses the optimal tracking problem of continuous-time nonlinear systems with input constraints. The work introduces a novel tuning law for the critic's neural network parameters that can improve the learning rate.

1.4 Applications to unmanned combat aerial vehicles (UCAVs)

DRL has a distinct advantage over other approaches to flight control in its ability to perceive information and learn effectively. This makes it an ideal method for solving complex, high-dimensional sequential decision-making problems, such as those encountered in air combat [18]. In modern aerial combat with unmanned combat aerial vehicles (UCAVs), short-range aerial combat (dogfight) is still considered an important topic [19,20]. The USA has planned a roadmap by 2030 for the formation flight of UCAVs [21]. In the roadmap, such a UAV mission is expected to be complex and multifaceted, requiring precise waypoint-line tracking and advanced formation flight algorithms to enable UCAVs to execute it effectively. In complex formation flights such as aerial combat situations, the ability of a UCAV to quickly adjust its flight attitude and path is crucial. This ability is often referred to as flight agility, which involves the rapid and seamless transition between different flight states.

To simulate aerial combat scenarios involving un manned vehicles, Wang et al. [22] extended their basic manoeuvre library to account for the enhanced manoeuvring capabilities of UCAVs. They also proposed a robust decision-making method for manoeuvring under incomplete target information. Additionally, reinforcement learning [19,23] was explored as a machine learning approach to generate strategies, and McGrew [24] utilized the approximate dynamic programming (ADP) approach to solve a two-dimensional aerial combat problem involving a single UCAV acting as an agent and learning near-optimal actions through interaction with the combat environment.

Researchers have been exploring as well the potential of combining DRL with UCAVs' air combat decision-making [18,20,25]. For instance, Li [20] proposed a model for UCAV autonomous manoeuvre decision-making in short-range air combat, based on the multi-step double deep Q-network (MS-DDQN) algorithm. To address the limitations of traditional methods such as poor flexibility and weak decision-making ability, some researchers have proposed the use of deep learning for manoeuvring [18]. Liu et al. [25] proposed a multi-UCAV cooperative decision-making method based on a multi-agent proximal policy optimization (MAPPO) algorithm. DRL has been used by Hu et al. [26] to plan beyond-visual-range air combat tactics and by Wang et al. [27] to quantify the relationship between the flight agility of a UCAV and its short-range aerial combat effectiveness. A similar approach has been also presented by Yang et al. [28].

While the aforementioned approaches have demonstrated some success in simulating aerial combat scenarios, they have primarily focused on generating action strategies, and only a few have applied a six-degree-of-freedom (6-DoF) nonlinear aircraft model to the simulations. The mass point model that was used in some studies fails to accurately represent the flight performance of a high-order UCAV. Additionally, directly applying these complex algorithms to a 6-DoF nonlinear model has been often discarded because of its demanding computational requirements. To address this limitation, Shin et al. [29] proposed a basic flight manoeuvre (BFM)-based approach to guide the attacker and simulate a 6-DoF nonlinear UCAV model in an aerial combat environment. These approaches combine a 3D point mass model with a control law design based on nonlinear dynamic inversion (NDI) to control the three-dimensional flight path of a 6-DoF nonlinear UCAV model [22].

1.5 Research contribution

This research introduces an effective nonlinear control application based on the DDPG actor-critic approach to the full-blown nonlinear, high-fidelity flight dynamics model (FDM) of a fighter jet. The use of this type of controller has been documented only for small-scale UAVs, and the present article addresses the application to the three-dimensional navigation of a high-performance UCAV. For this kind of application, the research does not assume a reduced order model to train the agent—such as the methods presented in [22] and [29]. In addition, unlike the BFM-based approaches, the proposed AI-based strategy generates the piloting commands to control directly the nonlinear 6-DoF UCAV model. Finally, extensive validation is carried out by a well-known realistic simulator. A high-fidelity FDM of the General Dynamics F-16 Fighting Falcon has been used, paving the way for future applications to other similar aircraft. The chosen FDM is publicly available and implemented for the JSBSim flight dynamics software library [30]. The simulation results presented in this work were obtained using JSBSim within the MATLAB/Simulink computational environment.

Moreover, the present effort deals with some modelling aspects in the aircraft state equations that are not commonly found in the literature (when DRL is applied to flight control). The underlying mathematical formulation to the 6-DoF aircraft-controlled dynamics presented here is based on the attitude quaternion; hence, it is general and suitable for all kinds of flight task simulations.

This work validates an AI-based flight control approach in a complex and high-performance requirement scenario, such as that of the target following. The control is accomplished by an agent trained with a DRL algorithm, whose applications have been documented in literature so far only for simpler models and less dynamic operative scenarios. In fact, the FDM implementation of the chosen aeroplane is highly nonlinear and realistic. The application examples shown at the end of the article demonstrate simulation scenarios where significant variations occur in the state space (for instance, altitude and flight Mach number changes) and multiple nonlinear effects come into play.

Finally, an additional contribution to the works available in the literature is the use of an established engineering simulation environment to validate the proposed approach. This, combined with the fact that the FDM includes realistic aerodynamics, propulsion, and FCS models, makes the agent validation closer to those learning applications occurring in experimental environments.

1.6 Article organization

The next section summarizes the main concepts regarding deep reinforcement learning and discusses how this approach has been employed to control an aircraft in atmospheric flight.

The mathematical background of the article is summarized in Sect. 2. The foundations of DRL applied to flight control and the chosen control approach are explained in Sect. 3, where the details about the reward shaping are also presented. Simulation test scenarios that validate the proposed control strategy are presented in Sect. 4. A discussion of the presented results is given in Sect. 5. Finally, the conclusions are presented in Sect. 6. The article includes also three appendices A, B, and C with further details on the aerodynamic, propulsive, and flight control system models of the F-16 aircraft.

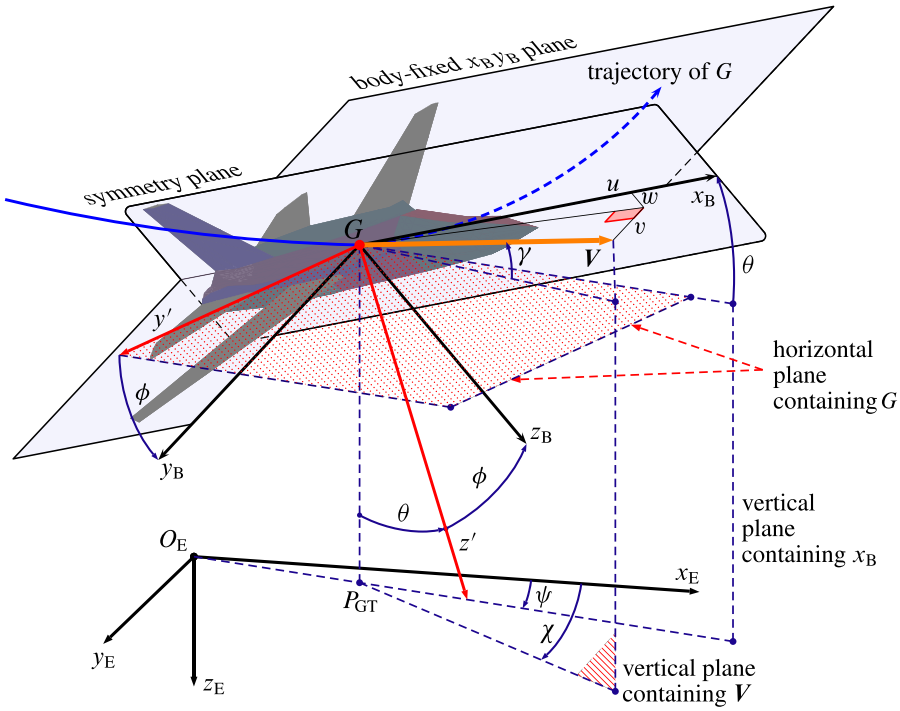


Fig. 1 Earth-based NED frame $\mathcal{F}_E$ and aircraft body-fixed frame $\mathcal{F}_B$. Velocity vector $\boldsymbol{V}$ of aircraft gravity centre G with respect to the inertial observer. Ground track point P_{GT} of the instantaneous gravity centre position $(x_{E,G}, y_{E,G}, x_{E,G})$. Standard definitions of aircraft Euler angles (ψ, θ, ϕ) and flight path angles (γ, χ)

2 Mathematical background

2.1 Rigid aeroplane nonlinear 6-DoF flight dynamics model

The 6-DoF atmospheric motion of a rigid aeroplane is governed by a set of nonlinear ordinary differential equations. These equations are standard; therefore, the complete derivation is herein omitted for sake of brevity—deferring the interested reader to the related references [1]. The derivation of such a system assumes a clear identification of a ground-based reference frame and an aircraft-based reference frame. The former is treated as an inertial frame, neglecting the effects of the rotational velocity of the Earth, which is a good approximation when describing the motion of aeroplanes in the lower regions of the Earth's atmosphere. This approximation is acceptable for subsonic as well as supersonic flight and fits the fighter jet control example considered in this research. These two main reference frames, besides other auxiliary frames, are shown in Fig. 1. The Earth-based inertial frame is named here $\mathcal{F}_E = \{O_E; x_E, y_E, z_E\}$, having its origin fixed to a convenient point O_E on the ground and its plane $x_E y_E$ tangent to the assumed Earth geometric model; the axis x_E points towards the geographic North, the axis y_E points towards the East; the axis z_E points downwards, towards the centre of the Earth. For this reason, the frame $\mathcal{F}_E$ is also called tangent NED frame (North-East-Down). The aircraft body-fixed frame $\mathcal{F}_B = \{G; x_B, y_B, z_B\}$, has its origin located at the centre of gravity G of the aircraft; the roll axis x_B runs along the fuselage and points out of the nose; the pitch axis y_B points towards the right wing tip; the yaw axis z_B points towards the belly of the fuselage.

The motion equations are derived from Newton's second law applied to the flight of an air vehicle, leading to six core scalar equations (the conservation laws of the linear and angular momentum projected onto the moving frame $\mathcal{F}_B$), followed by the flight path equations (used for navigation purposes, for tracking the aircraft flight in terms of centre-of-gravity position with respect to the Earth-based frame $\mathcal{F}_E$), and by the rigid-body motion kinematic equations (providing a relationship for the aircraft attitude quaternion, used for expressing the orientation of the body axes with respect to the inertial ground frame).

The full set of equations in closed form is introduced in this section. The JSBSim flight dynamics library

implements them in a more general form, which is beyond the scope of this article. However, the following presentation serves to highlight the intricacies and the nonlinearities inherent to the control problem explored by the present research.

2.1.1 Conservation of the linear momentum equations (CLMEs)

The conservation of the linear momentum equation (CLMEs) for a rigid aeroplane of constant mass provides the following three core scalar equations [1]:

$$\dot{u} = r\,v - q\,w + \frac{1}{m}\left(W_x + F_x^{(\mathrm{A})} + F_x^{(\mathrm{T})}\right) \tag{1a}$$

$$\dot{v} = -r\,u + p\,w + \frac{1}{m}\left(W_y + F_y^{(\mathrm{A})} + F_y^{(\mathrm{T})}\right) \tag{1b}$$

$$\dot{w} = q\,u - p\,v + \frac{1}{m}\left(W_z + F_z^{(\mathrm{A})} + F_z^{(\mathrm{T})}\right) \tag{1c}$$

where the $\boldsymbol{W}$ is the aircraft weight force, $\boldsymbol{F}^{(\mathrm{A})}$ is the resultant aerodynamic force, and $\boldsymbol{F}^{(\mathrm{T})}$ is the resultant thrust force. Their components in the body frame $\mathcal{F}_\mathrm{B}$ are conveniently expressed to obtain a closed form of Eqs. (1a)–(1b)–(1c).

The aircraft weight is a vertical force vector, i.e. always aligned to the inertial axis z_E, of constant magnitude mg, that is expressed in terms of its components in body axes as follows:

$$\begin{Bmatrix} W_x \\ W_y \\ W_z \end{Bmatrix} = [T_\mathrm{BE}] \begin{Bmatrix} 0 \\ 0 \\ mg \end{Bmatrix} = \begin{Bmatrix} 2\left(q_z\,q_x - q_0\,q_y\right) \\ 2\left(q_y\,q_z + q_0\,q_x\right) \\ q_0^2 - q_x^2 - q_y^2 + q_z^2 \end{Bmatrix} mg \tag{2}$$

where $[T_\mathrm{BE}]$ is the direction cosine matrix representing the instantaneous attitude of frame $\mathcal{F}_\mathrm{B}$ with respect to frame $\mathcal{F}_\mathrm{E}$. The entries of $[T_\mathrm{BE}]$ are functions of the aircraft attitude quaternion components (q_0, q_x, q_y, q_z) [1]:

$$[T_\mathrm{BE}] = \begin{bmatrix} q_0^2+q_x^2-q_y^2-q_z^2 & 2\left(q_x\,q_y+q_0\,q_z\right) & 2\left(q_z\,q_x-q_0\,q_y\right) \\ 2\left(q_x\,q_y-q_0\,q_z\right) & q_0^2-q_x^2+q_y^2-q_z^2 & 2\left(q_y\,q_z+q_0\,q_x\right) \\ 2\left(q_z\,q_x+q_0\,q_y\right) & 2\left(q_y\,q_z-q_0\,q_x\right) & q_0^2-q_x^2-q_y^2+q_z^2 \end{bmatrix} \tag{3}$$

The instantaneous resultant aerodynamic force $\boldsymbol{F}^{(\mathrm{A})}$ acting on the air vehicle, when projected onto $\mathcal{F}_\mathrm{B}$, is commonly expressed as follows [1]:

$$\begin{Bmatrix} F_x^{(\mathrm{A})} \\ F_y^{(\mathrm{A})} \\ F_z^{(\mathrm{A})} \end{Bmatrix} = [T_\mathrm{BW}] \begin{Bmatrix} -D \\ -C \\ -L \end{Bmatrix} = \begin{Bmatrix} -D\cos\alpha\cos\beta + L\sin\alpha + C\cos\alpha\sin\beta \\ -C\cos\beta - D\sin\beta \\ -D\sin\alpha\cos\beta - L\cos\alpha + C\sin\alpha\sin\beta \end{Bmatrix} \tag{4}$$

where the aerodynamic drag D, the aerodynamic cross force C and the aerodynamic lift L are involved to conveniently model the effect of the external airflow, and where

$$[T_\mathrm{BW}] = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} \cos\beta & \sin(-\beta) & 0 \\ -\sin(-\beta) & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

is the coordinate transformation matrix from the standard wind frame $\mathcal{F}_\mathrm{W} = \{G; x_\mathrm{W}, y_\mathrm{W}, z_\mathrm{W}\}$, commonly used by aerodynamicists (see Fig. 2), to $\mathcal{F}_\mathrm{B}$.

Equations (1a)–(1b)–(1c) are actually set in closed form because the aerodynamic angles (α, β) and the aerodynamic force components (D, C, L) appearing in expressions (4) are modelled as functions of aircraft state variables and of external inputs. According to Fig. 2, the state variables (u, v, w), being defined as components in $\mathcal{F}_\mathrm{B}$ of the aircraft gravity centre velocity vector $\boldsymbol{V}$, are expressed in terms of (α, β) as follows:

$$u = V\cos\beta\cos\alpha, \quad v = V\sin\beta, \quad w = V\cos\beta\sin\alpha \tag{6}$$

with

$$V = \sqrt{u^2 + v^2 + w^2} \tag{7}$$

Consequently, the instantaneous angles of attack and of sideslip are given by

$$\alpha = \tan^{-1}\frac{w}{u}, \quad \beta = \sin^{-1}\frac{v}{\sqrt{u^2+v^2+w^2}} \tag{8}$$

The aerodynamic force components are expressed in terms of their aerodynamic coefficients according to the conventional formulas:

$$D = \frac{1}{2}\rho V^2\,S\,C_D, \quad C = \frac{1}{2}\rho V^2\,S\,C_C, \quad L = \frac{1}{2}\rho V^2\,S\,C_L \tag{9}$$

where the external air density ρ is a known function of the flight altitude $h = -z_{\mathrm{E},G}$ (along with other gas properties, such as the sound speed a) [31], S is a constant reference area, and coefficients (C_D, C_C, C_L) are modelled as functions of aircraft state variables and external inputs. Appendix A presents the details of a nonlinear aerodynamic model for the F-16 fighter jet.

Finally, according to Fig. 3, the resultant thrust force $\boldsymbol{F}^{(\mathrm{T})}$, which is a vector of magnitude T, can be expressed in terms of its body-frame components, in

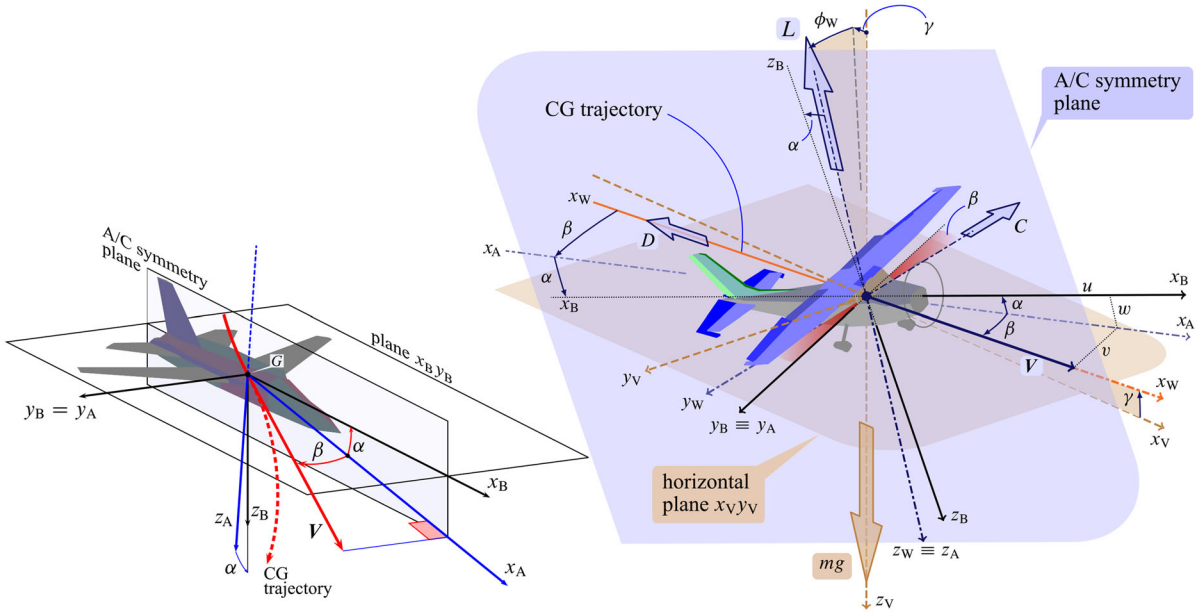


Fig. 2 (Left) Aerodynamic angles, aerodynamic (or stability) frame $\mathcal{F}_A$. (Right) Wind frame $\mathcal{F}_W$ and aerodynamic forces (D, C, L)

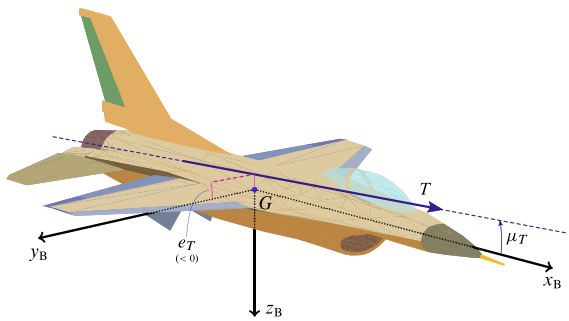


Fig. 3 Thrust vector, thrust magnitude T, thrust line angle μ_T, thrust line eccentricity e_T

the case of symmetric propulsion (zero y-component), as follows:

$$\begin{Bmatrix} F_x^{(T)} \\ F_y^{(T)} \\ F_z^{(T)} \end{Bmatrix} = \delta_T \, T_{\max}(h, M) \begin{Bmatrix} \cos \mu_T \\ 0 \\ \sin \mu_T \end{Bmatrix} \tag{10}$$

where μ_T is a known constant angle formed by the thrust line in the aircraft symmetry plane with the reference axis x_B, $T = \delta_T \, T_{\max}(h, M)$, δ_T is the throttle setting (an external input to the system), and $T_{\max}(h, M)$ is the maximum available thrust, i.e. a known function of altitude and flight Mach number $M = V/a$. Appendix B presents the details of a nonlinear thrust model for the F-16 fighter jet.

2.1.2 *Conservation of the angular momentum equations (CAMEs)*

The conservation of the angular momentum equations (CAMEs) for a rigid aeroplane of constant mass are the following [1]:

$$\dot{p} = \left(C_1 \, r + C_2 \, p\right) q + C_3 \, \mathcal{L} + C_4 \, \mathcal{N} \tag{11a}$$

$$\dot{q} = C_5 \, p \, r - C_6 \left(p^2 - r^2\right) + C_7 \, \mathcal{M} \tag{11b}$$

$$\dot{r} = \left(C_8 \, p - C_2 \, r\right) q + C_4 \, \mathcal{L} + C_9 \, \mathcal{N} \tag{11c}$$

where

$$C_1 = \frac{1}{\Gamma}\left[(I_{yy} - I_{zz}) I_{zz} - I_{xz}^2\right],$$

$$C_2 = \frac{1}{\Gamma}\left[(I_{xx} - I_{yy} + I_{zz}) I_{xz}\right] \tag{12a}$$

$$C_3 = \frac{I_{zz}}{\Gamma}, \quad C_4 = \frac{I_{xz}}{\Gamma}, \quad C_5 = \frac{I_{zz} - I_{xx}}{I_{yy}} \tag{12b}$$

$$C_6 = \frac{I_{xz}}{I_{yy}}, \quad C_7 = \frac{1}{I_{yy}},$$

$$C_8 = \frac{1}{\Gamma}\left[(I_{xx} - I_{yy}) I_{xx} + I_{xz}^2\right], \quad C_9 = \frac{I_{xx}}{\Gamma} \tag{12c}$$

and $\Gamma = I_{xx} I_{zz} - I_{xz}^2$ are constants of the model known from the rigid aeroplane inertia matrix calculated with respect to the axes of $\mathcal{F}_B$.

The instantaneous resultant external moment $\boldsymbol{M}$ about the pole G acting on the air vehicle is the sum

of the resultant aerodynamic moment $\boldsymbol{M}^{(\mathrm{A})}$ and of the resultant moment $\boldsymbol{M}^{(\mathrm{T})}$ due to thrust line eccentricity with respect to G. When $\boldsymbol{M}$ is projected onto $\mathcal{F}_{\mathrm{B}}$, it is commonly expressed as:

$$\begin{Bmatrix} \mathcal{L} \\ \mathcal{M} \\ \mathcal{N} \end{Bmatrix} = \begin{Bmatrix} \mathcal{L}^{(\mathrm{A})} \\ \mathcal{M}^{(\mathrm{A})} \\ \mathcal{N}^{(\mathrm{A})} \end{Bmatrix} + \begin{Bmatrix} 0 \\ \mathcal{M}^{(\mathrm{T})} \\ 0 \end{Bmatrix} \tag{13}$$

in the case of symmetric propulsion (thrust line in the aircraft symmetry plane).

Equations (11) are actually set in closed form because the aerodynamic moments about the roll, pitch and yaw axes $(\mathcal{L}^{(\mathrm{A})}, \mathcal{M}^{(\mathrm{A})}, \mathcal{N}^{(\mathrm{A})})$ are modelled as functions of aircraft state variables and of external inputs. The same applies to the pitching moment $\mathcal{M}^{(\mathrm{T})}$.

The body-axis components of $\boldsymbol{M}^{(\mathrm{A})}$ are commonly expressed in terms of their coefficients according to the conventional formulas:

$$\begin{Bmatrix} \mathcal{L}^{(\mathrm{A})} \\ \mathcal{M}^{(\mathrm{A})} \\ \mathcal{N}^{(\mathrm{A})} \end{Bmatrix} = \frac{1}{2}\rho V^2 S \begin{Bmatrix} b\, C_{\mathcal{L}} \\ \bar{c}\, C_{\mathcal{M}} \\ b\, C_{\mathcal{N}} \end{Bmatrix} \tag{14}$$

where b and $\bar{c}$ are reference lengths known from the aeroplane's geometry. Appendix A presents a high-fidelity nonlinear model of the aerodynamic roll-, pitch-, and yaw-moment coefficients $(C_{\mathcal{L}}, C_{\mathcal{M}}, C_{\mathcal{N}})$ for the F-16 fighter jet.

The pitching moment due to thrust $\mathcal{M}^{(\mathrm{T})}$ is given by the direct action of the thrust vector:

$$\mathcal{M}^{(\mathrm{T})} = T\, e_T \tag{15}$$

where the eccentricity e_T is a known parameter, positive when the thrust line is located beneath the centre of gravity ($e_T < 0$ as shown in Fig. 3).

2.1.3 Flight path equations (FPEs)

Systems (1a)–(1b)–(1b)–(11a)–(11b)–(11c) of CLMEs and CAMEs projected onto the moving frame $\mathcal{F}_{\mathrm{B}}$ must be necessarily augmented with two additional systems of equations to solve the aircraft dynamics and propagate its state in time. One such system is needed for expressing the trajectory of the aircraft with respect to the Earth-based inertial frame. The flight path equations (FPEs) are specifically used for this purpose. The outputs of this system of differential equations form the instantaneous position $\{x_{\mathrm{E},G}(t), y_{\mathrm{E},G}(t), z_{\mathrm{E},G}(t)\}$ of the aircraft centre of gravity G in the space $\mathcal{F}_{\mathrm{E}}$. The reduced 2D version $\{x_{\mathrm{E},G}(t), y_{\mathrm{E},G}(t)\}$ of the FPEs provides the so-called ground track relative to the aircraft flight.

The FPEs are simply derived from the component transformation of vector $\boldsymbol{V}$ from frame $\mathcal{F}_{\mathrm{B}}$ to frame $\mathcal{F}_{\mathrm{E}}$

$$\begin{Bmatrix} \dot{x}_{\mathrm{E},G} \\ \dot{y}_{\mathrm{E},G} \\ \dot{z}_{\mathrm{E},G} \end{Bmatrix} = [T_{\mathrm{EB}}] \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{16}$$

knowing that $[T_{\mathrm{EB}}] = [T_{\mathrm{BE}}]^{\mathrm{T}}$ and accounting for definition (3). The FPEs are then written in matrix format as follows:

$$\begin{Bmatrix} \dot{x}_{\mathrm{E},G} \\ \dot{y}_{\mathrm{E},G} \\ \dot{z}_{\mathrm{E},G} \end{Bmatrix} = \begin{bmatrix} q_0^2+q_x^2-q_y^2-q_z^2 & 2(q_x q_y - q_0 q_z) & 2(q_z q_x + q_0 q_y) \\ 2(q_x q_y + q_0 q_z) & q_0^2-q_x^2+q_y^2-q_z^2 & 2(q_y q_z - q_0 q_x) \\ 2(q_z q_x - q_0 q_y) & 2(q_y q_z + q_0 q_x) & q_0^2-q_x^2-q_y^2+q_z^2 \end{bmatrix} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix} \tag{17}$$

The inputs for the FPEs are the aircraft attitude quaternion components along with the components (u, v, w), which are provided by the solution of the combined (CLMEs)-(CAMEs) system.

The quaternion components instead are derived from the body-frame components (p, q, r) of the aircraft angular velocity vector $\boldsymbol{\Omega}$ through the solution of another set of equations to be introduced next.

2.1.4 Kinematic equations (KEs)

The rigid-body kinematic equations (KEs) based on the aircraft attitude quaternion components [1] are written in matrix format as follows:

$$\begin{Bmatrix} \dot{q}_0 \\ \dot{q}_x \\ \dot{q}_y \\ \dot{q}_z \end{Bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} \begin{Bmatrix} q_0 \\ q_x \\ q_y \\ q_z \end{Bmatrix} \tag{18}$$

The inputs to the KEs are the angular velocity components (p, q, r) in $\mathcal{F}_{\mathrm{B}}$. Their solution provides the numerical values of the kinematic state variables (q_0, q_x, q_y, q_z).

The above system is the set of differential equations of choice, nowadays, in large-scale simulations because it prevents the singularity known as 'gimbal lock' associated with the alternative formulation based on aircraft Euler angles. However, at each instant t when a quadruplet of quaternion components is known, the Euler angles (ψ, θ, ϕ), shown in Fig. 4, are calculated according to a standard algorithm [32].

2.1.5 Summary of the equations and of system inputs

The system of (CLMEs)-(CAMEs)-(FPEs)-(KEs), that is, (1)–(11)–(17)–(18), is the full set of 13 coupled

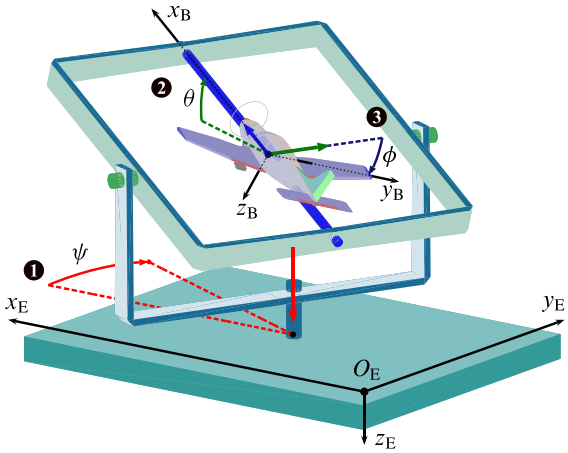


Fig. 4 Aircraft Euler angles and attitude with respect to the Earth frame

nonlinear differential equations governing the 6-DoF, rigid-body dynamics of atmospheric flight. They are in a closed form once the aerodynamic as well as the propulsive external forces and moments are completely modelled as functions of 13 state variables

$$\boldsymbol{x} = \left[u, v, w,\ p, q, r,\ x_{\mathrm{E},G}, y_{\mathrm{E},G}, z_{\mathrm{E},G},\ q_0, q_x, q_y, q_z\right]^{\mathrm{T}} \tag{19}$$

i.e. of a state vector $\boldsymbol{x}$, and of a number of external inputs grouped into an input vector, commonly known as vector $\boldsymbol{u}$.

The F-16 public domain model used for this research features a quite articulated and high-fidelity flight control system (FCS), whose simplified scheme is depicted in Fig. 5. The FCS, which receives state feedback from the aircraft dynamics block, includes the following channels: (i) Roll command $\tilde{\delta}_{\mathrm{a}}$ (acting on right aileron deflection angle δ_{a} and on the antisymmetric left aileron deflection), (ii) Pitch command $\tilde{\delta}_{\mathrm{e}}$ (acting on elevon deflection angle δ_{e}), (iii) Yaw command $\tilde{\delta}_{\mathrm{r}}$ (acting on rudder deflection angle δ_{r}), (iv) Throttle lever command $\tilde{\delta}_T$ (acting on throttle setting δ_T, with the possibility to trigger the jet engine afterburner), (v) Speed brake command $\tilde{\delta}_{\mathrm{sb}}$ (acting on speed break deflection angle δ_{sb}), (vi) Wing trailing-edge flaps command $\tilde{\delta}_{\mathrm{f,TE}}$ (acting on trailing-edge flap deflection angle $\delta_{\mathrm{f,TE}}$) and (vii) Wing leading-edge flap command $\tilde{\delta}_{\mathrm{f,LE}}$ (acting on leading-edge flap deflection angle $\delta_{\mathrm{f,LE}}$).

Some of these channels are associated with actual pilot's input command signals, such as the primary flight controls $\tilde{\delta}_{\mathrm{a}}$, $\tilde{\delta}_{\mathrm{e}}$, $\tilde{\delta}_{\mathrm{r}}$, $\tilde{\delta}_T$. Some other signals are mainly actuated and controlled by the FCS, for instance, $\tilde{\delta}_{\mathrm{f,TE}}$, $\tilde{\delta}_{\mathrm{f,LE}}$ and $\tilde{\delta}_{\mathrm{b}}$. Additional details on the FCS including a summary of the control logic are reported in Appendix C.

A selected number of the above inputs form the vector $\tilde{\boldsymbol{u}}_{\mathrm{agt}}$ of normalized commands used by the agent to interact with the system in all training sessions:

$$\tilde{\boldsymbol{u}}_{\mathrm{agt}} = \left[\tilde{\delta}_{\mathrm{a}}, \tilde{\delta}_{\mathrm{e}}, \tilde{\delta}_{\mathrm{r}}, \tilde{\delta}_T\right]^{\mathrm{T}} \tag{20}$$

The full input vector to the nonlinear aircraft flight dynamics model during all simulations performed in this work—resulting from the action of the agent in combination with the FCS logics—is then the following:

$$\boldsymbol{u} = \left[\delta_{\mathrm{a}}, \delta_{\mathrm{e}}, \delta_{\mathrm{r}}, \delta_T, \delta_{\mathrm{f,TE}}, \delta_{\mathrm{f,LE}}, \delta_{\mathrm{sb}}\right]^{\mathrm{T}} \tag{21}$$

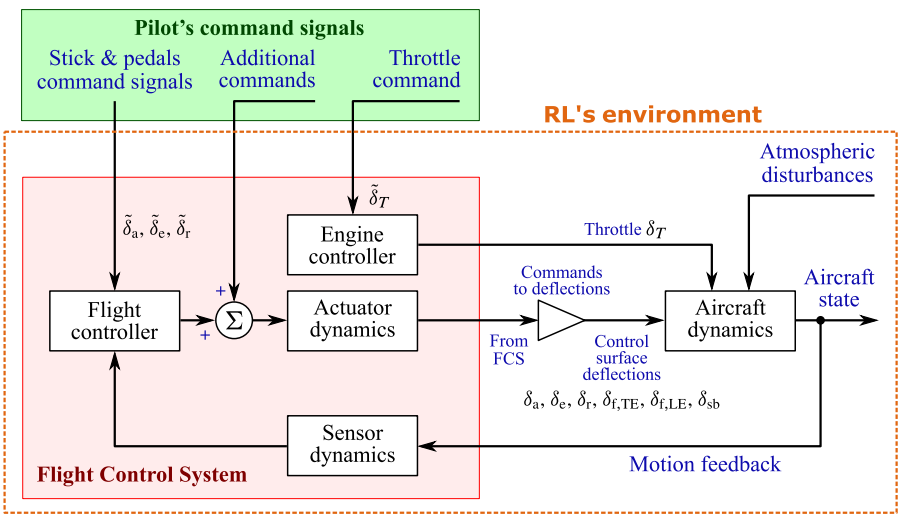


Fig. 5 Simplified scheme of a General Dynamics F-16 flight control system. See Appendixes B and C for more details on the flight controller and engine controller

Fig. 6 General Dynamics F-16 Fighting Falcon. Aerodynamic control surface deflections

The deflection angles of the F-16 aerodynamic control surfaces are depicted in Fig. 6.

3 Deep reinforcement learning approach applied to flight control

There is a growing interest, nowadays, in AI-based pilot models, which are going to augment the manned legacy fighters' capabilities and might bring them to compete with the next-generation air dominance systems. This is the main motivation for the present research. A related motivation lies in the possibility of implementing more effective AI-assisted pilot training procedures for high-precision tasks, such as dogfights and formation flights [20,25–27].

DRL is a promising approach to be combined with established flight control design techniques, and it provides the following advantages: (i) it can make a flight control system learn optimal policies, by interacting with the environment, when the aircraft nonlinear dynamics are not completely known or difficult to model; (ii) it can deal with changing situations in uncertain and strongly dynamic environments. In this section, we recall the basic concepts to explain how DRL, in particular the DDPG idea, is applicable to the chosen flight control scenarios presented later in the article.

3.1 The reinforcement learning framework

RL is a significant branch of machine learning that is concerned with how to learn control laws and policies from experience, which make a dynamical system interact with a complex environment according to a given task. Both control theory and machine learning fundamentally rely on optimization, and likewise, RL involves a set of optimization techniques within an observational framework for learning how to interact with the environment. In this sense, RL stands at the intersection of control theory and machine learning. This is explained rigorously in a well-known introductory book by Sutton and Barto [8]. On the other hand, in their recent textbook on data-driven science and engineering Brunton and Kutz [33] use a modern and unified notation to present an overview of state-of-the-art RL techniques applied to various fields, including a mathematical formulation for DRL. The reader may refer to these bibliographical references for a comprehensive explanation of the theory behind all variants of RL approaches.

RL methods apply to problems where an agent interacts with its environment in discrete time steps [8,33], as depicted in Fig. 7. At time step k, the agent senses the state $\boldsymbol{s}_k$ of its environment and decides to perform an action $\boldsymbol{a}_k$ that brings to a new state $\boldsymbol{s}_{k+1}$ at the next time step, obtaining a reward r_k. The scheme is reiterated with advancing time steps, $k \leftarrow k+1$, and the RL agent *learns* to take appropriate actions to achieve optimal immediate or delayed rewards.

The agent's behaviour over time is given by the finite sequence of states and actions $\boldsymbol{e} = \big[(\boldsymbol{s}_0, \boldsymbol{a}_0), (\boldsymbol{s}_1, \boldsymbol{a}_1), \ldots, (\boldsymbol{s}_{\mathcal{T}}, \boldsymbol{a}_{\mathcal{T}})\big]$, also called an *episode* (or *trajectory*), where $\mathcal{T}$ is the episode's final time step. An episode may end when a target terminal state is reached, or else when $\mathcal{T}$ becomes the maximum allowed number of time steps.

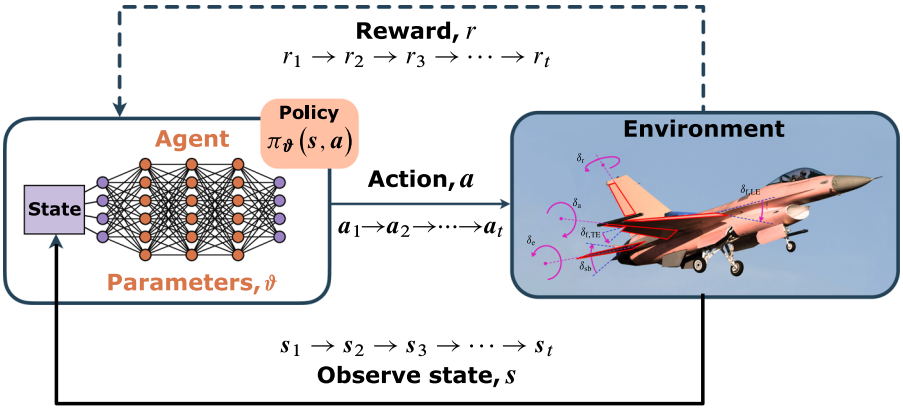


Fig. 7 Schematic of reinforcement learning, in the particular case of an agent's policy π represented by a deep neural network (adapted from [33]). In the case of flight control, the action $\boldsymbol{a}$ is the input vector $\tilde{\boldsymbol{u}}_{\text{agt}}$ defined by Eq. (20); the observed state is the vector $\boldsymbol{s} = [\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u}, \dot{\boldsymbol{u}}]^{\text{T}}$, where $\boldsymbol{x}$ and $\boldsymbol{u}$ are defined by Eqs. (19) and (21), respectively

In RL, the environment and the agent are dynamic. The goal of learning is to find an optimal policy, i.e. a control law (or behaviour) that generates an action sequence able to obtain the best possible outcome. A behaviour is optimal when it collects the highest cumulated reward. This is accomplished by allowing a piece of software—that is, the agent—to explore the environment, interact with it, and learn by observing how it evolves over time. Whenever the agent takes an action, it affects the environment, which transitions to a new state, or even when the agent takes no action the environment still might change. Within an entire episode, the evolutionary environment produces a sequence of rewards, and using this information, the agent can adjust its future behaviour and learn from the process. In the particular scheme of Fig. 7, for the sake of example, the agent is a deep neural network that receives a large number of input signals and then adjusts its parameters $\boldsymbol{\vartheta}$ during training so that it can control the environment as requested. In the specific environment of an aeroplane in controlled flight, the action $\boldsymbol{a}$ is the control input vector $\tilde{\boldsymbol{u}}_{\text{agt}}$ defined by Eq. (20), while the observed state $\boldsymbol{s} = [\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u}, \dot{\boldsymbol{u}}]^{\text{T}}$ incorporates vectors $\boldsymbol{x}$ and $\boldsymbol{u}$ (Eqs. (19) and (21), respectively), and their time derivatives. More details are given in Sect. 3.7 and in Fig. 9.

An important point to observe is that in RL the environment is everything that exists outside of the agent. Practically speaking, it is where the agent sends actions and what generates rewards and observations. In this context, the environment is different from what is accepted by control engineers, who tend to think of the environment as everything outside of the controller *and* the plant. In classical flight control approaches things like wind gusts and other disturbances that impact the system represent the environment. In RL, the environment is everything outside the controller as shown in Fig. 5, and this includes the plant dynamics as well. The agent is just a bit of software that is generating the actions and updating the control laws through learning. In the present case study, the agent acts like the 'brain' of the pilot that learns to control the aircraft.

The state $\boldsymbol{s}$ is a collection of variables observed by the agent while interacting with the environment, which is taken from the list $(\boldsymbol{x}, \boldsymbol{u})$ defined by (19) and (21). For the agent that controls the F-16 flight dynamics, the action $\boldsymbol{a}$ is given by the inputs $\tilde{\boldsymbol{u}}_{\text{agt}}$ defined by (20).

An important characteristic of reinforcement learning is that the agent is not required to know in advance the environment's dynamics. It will learn on its own only by observation, provided that it has access to the states and that rewards are appropriately calculated. This makes RL a powerful approach that can lead to an effective model-free control design. In flight control, for instance, the learning agent does not need to initially know anything about a flying vehicle. It will still figure out how to collect rewards without knowing the aircraft weight, how the aerosurfaces move, or how effective they are. The training technique used in this research applies a method designed originally for model-free applications to a problem of flight control where a high-fidelity model of the environment is known.

An RL framework can be applied to a simulated environment, that requires to be modelled, or even to a physical environment. In the present study, the agent is trained within a high-fidelity model of the environment; hence, the learning experience is carried out with flight simulations. Learning is a process that often requires millions or tens of millions of samples, which means

a huge number of trial simulations, error calculations and corrections. The obvious advantage of a simulated environment is that the learning process can be run at faster than real time and that simulations may be executed in parallel on multiprocessor machines or GPU clusters.

An important step in RL-based control design workflows is the deployment of the control algorithm on the target hardware or within a chosen simulation environment. When learning ends successfully, the agent's policy can be considered optimal and frozen, and a static instance of the policy can be deployed onto the target as any other form of control law. Examples of use of an optimized agent deployed in different simulation environments are presented in Sect. 4.

3.2 Rewards

Given the environment, the RL has to be based on how the agent should behave and how it will be rewarded for doing what an assigned task prescribes. The reward can be earned at every time step (immediate), or be sparse (delayed), or can only come at the very end of an episode after long periods of time. In the F-16 flight simulations, an immediate reward is available at each environment transition from time t_k to t_{k+1} as a function $r(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1})$ of the two consecutive states and of the action that causes the transition.

In RL, there is no restriction on creating a reward function, unlike LQR in the theory of optimal control where the cost function is quadratic. The reward can be calculated from a nonlinear function or using thousands of parameters. It completely depends on what it takes to effectively train the agent. For instance, if a flight attitude is requested, with a prescribed heading ψ_c, then one should intuitively tend to give more rewards to the agent as the heading angle ψ gets closer to the commanded direction. If one wants to take controller effort into account, then rewards should be subtracted as actuator use increases.

A reward function can really be any function one can think of. But making an effective reward function, on the other hand, requires ingenuity. Unfortunately, there is no straightforward way of crafting a reward to guarantee the agent will converge on the desired control. The definition of a reward calculation function, appropriate to a prescribed control task, is called *reward shaping* and is one of the most difficult tasks in RL. Section 3.11 introduces a convenient reward calculation function, which can be adopted in a flight control task where a target altitude and heading are commanded.

3.3 Policy function

Besides the environment that provides the rewards, an RL-based control must have a well-designed agent. The agent is comprised of its *policy* and of the learning algorithm, two things that are closely interlaced. Many learning algorithms require a specific policy structure and choosing an algorithm depends on the nature of the environment.

A policy $\pi(\boldsymbol{s}, \boldsymbol{a})$ defines the behaviour of the agent, in order to determine which action should be taken in each state. Formally speaking, it is a function $\pi : (\boldsymbol{s}, \boldsymbol{a}) \mapsto [0, 1]$ that maps a state–action pair to a scalar value between 0 and 1, regarded as a conditional probability density function of taking the action $\boldsymbol{a}$ while observing the state $\boldsymbol{s}$ (random policy). But a policy function can also be *deterministic*, in that case, given a state $\boldsymbol{s}$ the action $\boldsymbol{a}$ is nonrandom, i.e. $\pi : \boldsymbol{s} \mapsto \boldsymbol{a}$.

The term 'policy' means that the agent has to make a decision based on the current state. Training an agent to accomplish a given task in the simulated environment means generating several simulations and updating the policy parameters while episodes are generated by maximizing the policy performance. The policy function tells the agent what to do, and learning a policy function is the most important part of an RL algorithm.

As shown in Fig. 7, the policy is a function that takes in state observations and outputs actions; therefore, any function with that input and output relationship can work. Environments with a continuous state–action space have a continuous policy function, and it makes sense to represent π with a *general-purpose function approximator*, i.e. something that can handle a continuous state $\boldsymbol{s}(t)$ and a continuous action $\boldsymbol{a}(t)$, without having to set the nonlinear function structure ahead of time. This is accomplished with deep neural networks, a function approximation approach which forms the basis of DRL. The training algorithm selected for this research, known as DDPG and available in MATLAB/Simulink Reinforcement Learning Toolbox, adopts a *deterministic policy*, i.e. a function $\pi_{\boldsymbol{\vartheta}}(\boldsymbol{s})$ that is parameterized by a finite vector $\boldsymbol{\vartheta}$. With this struc-

ture, hundreds of time samples of the environment state are used, gathering several multi-dimensional observations of the F-16 in simulated flight as the input into π_{ϑ}, which outputs the actuator commands that drive the aerosurface movements and the thrust lever. Even though the function might be extremely complex, there will be a neural network of some kind that can achieve it.

3.4 Value function

The policy is used to explore the environment to generate episodes and calculate rewards. A performance index R_k of the policy at time step k is also called *discounted return*, i.e. the weighted sum of all rewards that the agent is expected to receive from time t_k onwards:

$$R_k = \sum_{i=0}^{\mathcal{T}} \gamma^i \, r_{k+i} \tag{22}$$

where $\gamma \in]0, 1]$ is called as the *discount rate*. The discount rate is a hyperparameter tuned by the user, and it represents how much future events lose their value in terms of policy performance according to how far away in time they are. Future rewards are discounted, reflecting the economic principle that current rewards are more valuable than future rewards. In general, R_k is a random variable resulting from future states and actions that are unknown at time t_k during learning.

Given a policy π, a *value function* is defined as the function that quantifies the desirability of being in a given state:

$$V_\pi(\boldsymbol{s}) = \mathbb{E}_\pi\big(R_k \mid \boldsymbol{s}_k = \boldsymbol{s} \big) \tag{23}$$

where $\mathbb{E}_\pi$ indicates the expected reward over the time steps from k to $\mathcal{T}$, when the state at time step k is $\boldsymbol{s}$. When the subscript π is omitted from the notation V_π, one refers to the value function for the best possible policy:

$$V(\boldsymbol{s}) = \max_\pi V_\pi(\boldsymbol{s}) \tag{24}$$

One of the most important properties of the value function is that the value $V(\boldsymbol{s}_k)$ at a time step k—also called V-value—is given by an elegant recursive formula known as *Bellman equation for* V:

$$V(\boldsymbol{s}) = \max_\pi \mathbb{E}_\pi\big[r_0 + \gamma V(\boldsymbol{s}') \big] \tag{25}$$

where $\boldsymbol{s}' = \boldsymbol{s}_{k+1}$ is the next state after $\boldsymbol{s} = \boldsymbol{s}_k$ given the action $\boldsymbol{a}_k$ rewarded with $r_0 = r_k$, and the expectation is over actions selected by the optimal policy $\pi_\star$. The value function V is the unique solution to its Bellman equation, which forms the basis of a number of ways to compute, approximate, and learn V. This formula, besides its implications to modern RL methods, is derived and extensively explained in [8].

Learning the value function V and jointly the optimal policy:

$$\pi_\star = \arg\max_\pi \mathbb{E}_\pi\big[r_0 + \gamma V(\boldsymbol{s}') \big] \tag{26}$$

is the central challenge in RL. All methods in RL that find an optimal value function V and the corresponding optimal policy $\pi_\star$ in a two-step procedure based on (25) and (26), are called *policy iteration* learning algorithms. In some cases, a large number of trials must often be evaluated in order to determine an optimal policy by iteration and, in practice, reinforcement learning may become very expensive to train. Yet, RL is well suited for the flight control problem stated in this research, where there is a known model for the environment, and where evaluating a policy is relatively affordable as there are sufficient resources to perform a near brute-force optimization.

In the present research, the agent learns through a hybrid technique that mixes policy-iteration learning with a strategy known as *Q-learning* (see Sect. 3.6). Moreover, all complexities in prediction functions are delegated to function approximators based on deep neural networks, to learn the policy $\pi_\star$, the value function V, and the quality function Q. The latter is introduced in the next subsection.

3.5 Quality function

The quality function, or Q-value, of a state–action pair $(\boldsymbol{s}, \boldsymbol{a})$ is defined as the expected value

$$Q(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}\big[r(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma V(\boldsymbol{s}') \big] \tag{27}$$

when at time step k the state is $\boldsymbol{s}$, the agent is assumed to follow the optimal policy, and the generic action $\boldsymbol{a}$ is taken. This brings to the next state $\boldsymbol{s}'$, resulting in the immediate reward $r(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')$ and a discounted future cumulated reward $\gamma V(\boldsymbol{s}')$.

While the value function V tells what is the value of being in a current state $\boldsymbol{s}$, the function Q—also called *action-value function*—is a joint quality or value of taking an action $\boldsymbol{a}$ given a current state $\boldsymbol{s}$. The quality function is somehow richer and it contains more information about what is the quality of being in a given state

for any action it might take. For this reason, sometimes Q is also called 'critic' because it can look at the possible actions and be used to criticize the agent's choices. This function can be approximated by a deep neural network, as well.

The optimal policy $\pi_\star(\boldsymbol{s}, \boldsymbol{a})$ and the optimal value function $V(\boldsymbol{s})$ contain redundant information, as one can be determined from the other via the quality function $Q(\boldsymbol{s}, \boldsymbol{a})$:

$$\pi_\star(\boldsymbol{s}, \boldsymbol{a}) = \arg\max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a}), \qquad V(\boldsymbol{s}) = \max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a}) \tag{28}$$

This formulation is used to define the Q-learning strategy, which is recalled in the next subsection.

3.6 Temporal difference and Q-learning

In an approach to learning from trial-and-error experience, the value function V or quality function Q is learned through a repeated evaluation of many policies [8,33]. In this work, the chosen learning process is not episodic (does not wait for the end of a control trajectory to update the policy), but instead, it is implemented in such a way as to learn continuously by *bootstrapping*. This technique is based on current estimates of V or Q, which are then repeatedly updated by scanning the successive states in the same control trajectory. In the simplest case, at each iteration of this learning technique, the value function is updated by means of a one-step look ahead, namely a value prediction for the next state $\boldsymbol{s}'$ given the current state $\boldsymbol{s}$ and action $\boldsymbol{a}$. This approach relies on Bellman's principle of optimality, which states that a large multi-step control policy must also be locally optimal in every subsequence of steps [33].

The temporal difference learning method known as TD(0) method simply approximates the expected discounted return with an estimation based on the reward immediately received summed to the value of the next state. Given a control trajectory generated through an optimal policy $\pi_\star$, for the Bellman's optimality condition the V-value of state $\boldsymbol{s}_k$ is given by:

$$V(\boldsymbol{s}_k) = \mathbb{E}_{\pi_\star}\big[\, r_k + \gamma V(\boldsymbol{s}_{k+1}) \,\big] \tag{29}$$

where $r_k + \gamma V(\boldsymbol{s}_{k+1})$ acts as an unbiased estimator for $V(\boldsymbol{s}_k)$, in the language of Bayesian statistics. For non-optimal policies π, this same idea is used to update the value function based on the value function one step ahead in the future, thus approximating the expected return as:

$$R_k \approx r(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) + \gamma\, V_\pi(\boldsymbol{s}_{k+1}) \tag{30}$$

or, using the Q-value, as:

$$R_k \approx r(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) + \gamma\, Q_\pi(\boldsymbol{s}_{k+1}, \boldsymbol{a}_{k+1}) \tag{31}$$

These approximations bring to the following definitions of the learning rules or update equations:

$$\begin{aligned} V_\pi(\boldsymbol{s}_k) &\leftarrow V_\pi(\boldsymbol{s}_k) + \eta\, \delta_{\mathrm{RPE}}(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) \\ Q_\pi(\boldsymbol{s}_k, \boldsymbol{a}_k) &\leftarrow Q_\pi(\boldsymbol{s}_k, \boldsymbol{a}_k) + \eta\, \delta_{\mathrm{TDE}}(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) \end{aligned} \tag{32}$$

where η is a learning rate between 0 and 1, and the quantities:

$$\begin{aligned} \delta_{\mathrm{RPE}} &= r(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) + \gamma\, V_\pi(\boldsymbol{s}_{k+1}) - V_\pi(\boldsymbol{s}_k) \\ \delta_{\mathrm{TDE}} &= r(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) + \gamma\, Q_\pi(\boldsymbol{s}_{k+1}, \boldsymbol{a}_{k+1}) \\ &\quad - Q_\pi(\boldsymbol{s}_k, \boldsymbol{a}_k) \end{aligned} \tag{33}$$

are called reward-prediction error (RPE) and temporal difference error (TDE), respectively. If the error is positive, the transition was positively surprising: one obtains more reward or lands in a better state than expected: the initial state or action was actually underrated, so its estimated value must be increased. Similarly, if the error is negative, the transition was negatively surprising: the initial state or action was overrated, and its value must be decreased. All methods based on update equations (32) are called *value-iteration* learning algorithms.

TD-based learning offers the advantage that after each state transition, the V-value or Q-value updates can be immediately applied, that is, there is no need to wait until an entire episode is completed. This process allows very fast learning and is called *online learning*. The policy updates may be applied at every single transition (TD(0) or 1-step look ahead) or the learning may proceed from batches of consecutive state transitions (TD(n) or n-step look ahead).

Q-learning is a technique that derives from TD learning and is particularly suitable for model-free RL. In Q-learning, the Q function is learned directly only by observing the evolutionary environment, in an approach that post-processes the generated control trajectories. It can be seen as a generalization of the many available model-based learning strategies, applicable to all those control scenarios that are difficult or impossible to model. As seen from (27), the Q function incorporates in its definition the very concept of a one-step look ahead, and does not need an environment's model.

Fig. 8 The Q-learning algorithm for estimating the optimal policy $\pi_\star$

Q-learning (Off-policy TD control) for estimating $\pi \approx \pi_\star$

Algorithm parameters: discount rate, learning rate $\gamma, \eta \in (0, 1]$
Assume an initial function $Q(\boldsymbol{s}, \boldsymbol{a})$ (i.e. initialize an ANN)
Loop for each episode:
    Initialize $\boldsymbol{s}$
    Loop for each step of episode:
        Choose $\boldsymbol{a}$ from $\boldsymbol{s}$ using policy derived from Q
        Take action $\boldsymbol{a}$, observe r and $\boldsymbol{s}'$ using policy derived from Q
        $Q(\boldsymbol{s}, \boldsymbol{a}) \leftarrow Q(\boldsymbol{s}, \boldsymbol{a}) + \eta\left[r + \gamma \max_{\boldsymbol{a}} Q(\boldsymbol{s}', \boldsymbol{a}) - Q(\boldsymbol{s}, \boldsymbol{a})\right]$
        $\boldsymbol{s} \leftarrow \boldsymbol{s}'$
    until $\boldsymbol{s}$ is terminal

Therefore, the learned Q function, the optimal policy, and the value function may be extracted as in (28).

In Q-learning, the Q function update equation is [8, 33]:

$$Q(\boldsymbol{s}_k, \boldsymbol{a}_k) \leftarrow Q(\boldsymbol{s}_k, \boldsymbol{a}_k) + \eta\, \hat{\delta}_{\text{TDE}}(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) \quad (34)$$

where

$$\begin{aligned}\hat{\delta}_{\text{TDE}}(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) = r(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) \\ +\gamma \max_{\boldsymbol{a}} Q(\boldsymbol{s}_{k+1}, \boldsymbol{a}) \\ -Q(\boldsymbol{s}_k, \boldsymbol{a}_k)\end{aligned} \quad (35)$$

is called the *off-policy* TDE. Due to the fact that the optimal $\boldsymbol{a}$ is used in (35) to determine the correction $\hat{\delta}_{\text{TDE}}$ based on the current estimate for Q, while taking a different action $\boldsymbol{a}_{k+1}$ based on a different behaviour policy for the next state transition, Q-learning is called an off-policy technique. Thus, Q-learning takes sub-optimal actions to explore future states but uses the optimal action $\boldsymbol{a}$ to improve the Q function at each state transition. The Q-learning algorithm for estimating the optimal policy $\pi_\star$ is summarized in Fig. 8. In practice, rather than finding the true Q-value of the state–action pair in one go at each time step t_k, through the Bellman equation the agent improves the approximation of the function Q over time.

Q-learning was formalized for solving problems with discrete action and state spaces governed by a finite Markov decision process. Yet, when the action and state spaces are continuous, as occurs in the flight control problems, time is discretized with an appropriate frequency and Q-learning becomes equally applicable. Its off-policy character has made Q-learning particularly well suited to many real-world applications, enabling the RL agent to improve when its policy is sub-optimal. In *deep Q-learning*, the Q function is conveniently represented as a neural network.

In all RL approaches, the concept of *learning rate* is very important: it determines to what extend, for each episode during agent training, newly acquired information overrides old information. A sufficient level of exploration has to be ensured in order to make sure that the estimates converge to the optimal values: this is known as the *exploration-exploitation* problem. At successive time steps within the generic episode, if the agent selects always the same action policy from the beginning (exploitation), it will never discover better (or worse) alternatives. On the other hand, if the policy is updated in such a way that it picks random actions (exploration), this random sub-optimal policy might have a chance to bring new information to the learning process. A trade-off between exploitation and exploration is ensured by the available Q-learning techniques: usually, a lot of exploration happens at the beginning of the training to accumulate knowledge about the environment and the control task, less towards the end to actually use the acquired knowledge and perform optimally. Generally, Q-learning will learn a more optimal solution faster than alternative techniques.

3.7 The actor-critic architecture

In RL, an actor-critic method consists in simultaneously learning a policy function and a value function by conveniently mixing value-iteration as well as policy-iteration learning. As shown in Fig. 9, in this agent architecture there is an actor-network, which is policy-based, and a critic network, which is value-based. The temporal difference signal from the critic is used to update the actor's policy parameters. In this particular case, the actor tries to take what it considers is the best action according to the current state (just as in a simpler

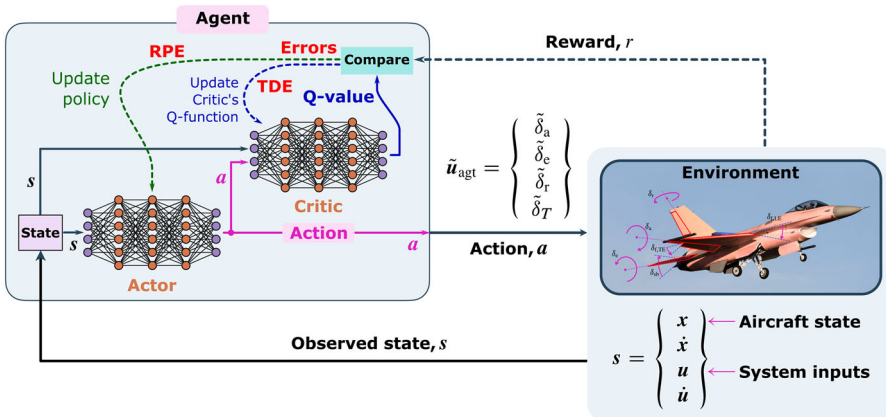


Fig. 9 Actor-critic architecture scheme

policy function method, Fig. 7); the critic estimates the Q-value associated to the state and to the action that the actor just took (as in Q-learning methods).

The actor-critic scheme works for continuous action spaces because the critic—which is supposedly the optimal Q function evaluator in the current condition—only needs to consider a single action, the one that the actor just took. In fact, when the actor selects an action, it is applied to the environment, the critic estimates the value of that state–action pair, and then it uses the reward from the environment as a metric to determine how good its Q-value prediction was. The error is the difference between the new estimated value of the previous state and the old value of the previous state from the critic network. The critic uses the error to update itself so that it has a less sub-optimal prediction the next time it is in that state. The actor-network also updates its parameters with the response from the critic and the error term so that it can improve its future behaviour. This research uses the DDPG training algorithm which is based on the actor-critic architecture [12]. The algorithm can learn from environments with continuous state and action spaces, and since it estimates a deterministic policy, it learns much faster than those techniques based on stochastic policies.

To compute the prediction errors, usually, many successive samples (single transitions) are gathered and concatenated in mini-batches, so that the critic's neural network could learn to minimize the prediction error from these chunks of data. Yet, the successive transitions concatenated inside a mini-batch are not independent of each other but correlated: $(\boldsymbol{s}_k, \boldsymbol{a}_k, r_{k+1}, \boldsymbol{s}_{k+1})$ will be followed by $(\boldsymbol{s}_{k+1}, \boldsymbol{a}_{k+1}, r_{k+2}, \boldsymbol{s}_{k+2})$, and so on, which is not a distribution of random samples. This is a major problem that promotes the tendency of the involved neural networks to overfit and fall into local minima.

Another major problem occurs with the actor and critic implemented as neural networks because their loss functions have non-steady targets. As opposed to classification or regression problems where the desired values are fixed throughout the network update iterations, in Q-learning, the target $r(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}) + \gamma \max_{\boldsymbol{a}} Q_{\boldsymbol{\vartheta}}(\boldsymbol{s}_{k+1}, \boldsymbol{a})$ will change because the function approximator $Q_{\boldsymbol{\vartheta}}$ depends on the weights $\boldsymbol{\vartheta}$. This circumstance can make the actor-critic pair particularly inefficient, especially if they are implemented as feed-forward networks and the control task features a moving reference.

3.8 Deep Q-networks

The problem in DRL caused by correlated samples within mini-batches has been solved by introducing a learning technique called deep Q-network (DQN) algorithm [34]. The approach relies on data structures called experience replay memory (ERM), or *replay buffers*, which are huge buffers where hundreds of thousands of successive transitions are stored. The agent is trained by randomically sampling mini-batches from the ERM, which is cyclically emptied and refilled with new samples.

On the other hand, to fix the problem related to the loss function target inherent unsteadiness, a DQN algorithm uses a cyclically frozen version of the agent called *target network*, which computes the transitions intended to feed the ERM. Only every few thousand

iterations the target network is updated so that the loss function targets remain stationary.

The learning rate of DQNs is known to be lower than those of other available approaches. This is due to the *sample complexity*, i.e. to the fact that the agent must inevitably enact a huge number, in the order of millions, of transitions to obtain a satisfying policy.

The function of the DQN within the agent's behaviour is depicted in the left part of Fig. 9 showing how the DQN trains the critic network to estimate the future rewards and to update the actor's policy (see also Fig. 8). In the case of flight control, the updated policy determines the action on aerosurfaces and engine throttle that maximizes the expected reward. The reward shaping approach for the control problem introduced by this work is presented in Sect. 3.11.

3.9 Policy gradient methods

In policy-based function approximation—which is the part of the actor-critic scheme that updates the actor—when the policy π is parameterized by $\boldsymbol{\vartheta}$, it is possible to use gradient optimization on the parameters to improve the policy much faster than other iterative methods.

The objective is to learn a policy that maximizes the expected return of each transition. The goal of the neural network is to maximize an objective function given by the return (22) calculated over a set of trajectories E selected by the policy:

$$\mathcal{J}(\boldsymbol{\vartheta}) = \mathbb{E}_E\big[R_k\big] = \mathbb{E}_E\left[\sum_{i=0}^{\mathcal{T}} \gamma^i r_{k+i}\right] \tag{36}$$

The algorithm known as *policy gradient method* applies a gradient ascent technique to the weights in order to maximize $\mathcal{J}$ in the space of $\boldsymbol{\vartheta}$'s. All it is actually necessary with this technique is the gradient of the objective function with respect to the weights $\nabla_{\boldsymbol{\vartheta}}\mathcal{J} = \partial\mathcal{J}/\partial\boldsymbol{\vartheta}$. As a suitable estimation of this policy gradient is obtained, the gradient ascent formula is straightforward:

$$\boldsymbol{\vartheta} \leftarrow \boldsymbol{\vartheta} + \eta\,\nabla_{\boldsymbol{\vartheta}}\mathcal{J}\big(\boldsymbol{\vartheta}\big) \tag{37}$$

The reader is referred to the article by Peters and Schaal [35] for an overview of policy gradient methods, with details on how to estimate the policy gradient and how to improve the sample complexity.

3.10 Deep deterministic policy gradient

In the application presented here, the DRL agent learns a deterministic policy. Deterministic policy gradient (DPG) algorithms form a family of methods developed to approximate the gradient of the objective function when the policy is deterministic [36]. The DPG approach has been improved in order to work with nonlinear function approximators [11] resulting in the method known as deep deterministic policy gradient. DDPG combines concepts from DQN and DPG obtaining an algorithm able to effectively solve continuous control problems with an off-policy method. As in DQN, an ERM (to store the past transitions and then learn off-policy) and target networks (to stabilize the learning) are used. However, in DQN the target networks are updated only every couple of thousands of steps: they change significantly between two updates, but not very often. Anyway, during the DDPG development, it turned out to be more efficient to make the target networks slowly track the trained networks. For this reason, both the target networks and the trained networks are updated together, by using the following update rule:

$$\boldsymbol{\vartheta}_{k+1} \leftarrow \tau\,\boldsymbol{\vartheta}_k + (1-\tau)\,\boldsymbol{\vartheta}_{k+1} \qquad \text{with } \tau \le 1 \tag{38}$$

where τ is a *smoothing factor* (usually much less than 1) that defines how much *delay* exists between target networks and trained networks. The above update strategy improves the stability in the Q function learning process.

The key idea taken from DPG is the policy gradient for the actor. The critic is learned by using regular Q-learning. Anyway, as the policy is deterministic, it can quickly produce always the same actions: this is an exploration problem. Some environments are naturally noisy, improving the exploration, but this cannot be assumed in the general case. To deal with that, DDPG perturbs the deterministic action with an additive noise $\boldsymbol{\xi}$ generated with a stochastic noise model, such that $\boldsymbol{a}_k \leftarrow \boldsymbol{a}_k + \boldsymbol{\xi}_k$, in order to force the exploration of the environment.

3.11 Reward function shaping

Finally, we define a basic reward calculation function r that determines the agent's reward r_k at time t_k. Generally speaking, the reward is based on current values of

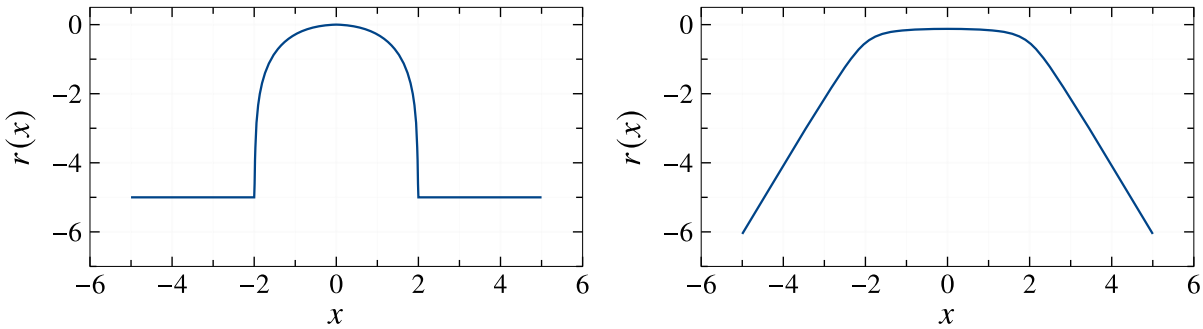


Fig. 10 Penalty functions tested in this research. (Left) Logarithmic barrier function. See Eq. (39), with $x_{\min} = -2$, $x_{\max} = 2$, $r_{\min} = -5$, $C = 1$. (Right) Hyperbolic function. See Eq. (40), with $x_{\min} = -2$, $x_{\max} = 2$, $\lambda = 1$, $\tau = 0.5$

the aircraft state and observable variables and is defined for specific control tasks. The formulation proposed here, which might appear limited to a given particular control scenario, in practice has proven to be flexible and widely applicable.

The reward signal is the way of communicating to the agent *what* has to be achieved, not *how* it has to be achieved. Therefore, the reward function is constructed in such a way as to properly guide the agent to the desired state, making sure not to impart *a priori* knowledge about how to achieve what the agent is supposed to accomplish. Moreover, a reward should not be set to be able to achieve subgoals and prefer them to the ultimate control aim. A properly rewarded agent does not find ways to reach a subgoal without achieving the real end target. The reward calculation is therefore accomplished by evaluating some properly designed simple functions of one or more scalar variables. These have the role of *penalty functions* because their output is usually much higher when their independent variables lie in given ranges.

A *logarithmic barrier penalty* reward function was first used in this work to test the default settings of the MATLAB Reinforcement Learning Toolbox. This function is defined as follows:

$$r(x)=\begin{cases} \max\left\{r_{\min},\ C\left[\ \log\left(\frac{1}{4}(x_{\max}-x_{\min})^2\right)\right.\right. & \\ \qquad \left.\left. -\log\left((x-x_{\min})(x_{\max}-x_{\min})\right)\right]\right\} & \text{if } x \in [x_{\min}, x_{\max}] \\ r_{\min} & \text{if } x \notin [x_{\min}, x_{\max}] \end{cases} \tag{39}$$

where $r_{\min} < 0$ is the minimum allowed reward, C is a nonnegative curvature parameter, and $[x_{\min}, x_{\max}]$ is the range where the reward cannot be as low as $r_{\min}$.

After some investigation, a second basic reward calculation method, named *hyperbolic penalty function*, turned out to give better results, in terms of flight control, as opposed to the logarithmic barrier penalty. The hyperbolic penalty evaluates to nearly constant values inside a given range of the independent variable and exhibits a nearly linear varying behaviour outside that range. It is defined as follows:

$$\begin{aligned} r(x) =& \lambda(x - x_{\min}) - \sqrt{\lambda^2(x - x_{\min})^2 + \tau^2} \\ &+ \lambda(x_{\max} - x) - \sqrt{\lambda^2(x_{\max} - x)^2 + \tau^2} \end{aligned} \tag{40}$$

where τ and λ are nonnegative shape parameters. In particular, $\pm\lambda$ are the slopes of the linear segments outside of the interval $[x_{\min}, x_{\max}]$. Two examples of logarithmic and hyperbolic penalty functions are shown in Fig. 10.

The above-defined functions are used to construct a reward r_k in terms of all or a subset of variables in the triplet $(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1})$. Care is taken when linking a higher reward to good behaviour because for a poorly defined r_k the agent may prefer to maximize its reward at the cost of not reaching the desired state. For instance, for a waypoint following task assigned to an aircraft controller, the agent might approach the target point in space and fly around the given waypoint in order to accumulate as much reward as possible instead of passing through the target and then finishing the episode (which will of course result in a lower total reward). For this reason, the maximum of the chosen function $r(x)$ will not be positive.

Therefore, assuming we have an array $\boldsymbol{\chi} = (\chi_1, \chi_2, \ldots, \chi_n)$ of n scalars, where each χ_i is a function of variables in $(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1})$, the total reward of a transition from time step k to $k+1$ will be:

$$r_k = \sum_{i=1}^{n} r(\chi_i) \tag{41}$$

where r can be the logarithmic (39) or the hyperbolic barrier penalty function (40). The number n will depend on the type of control task under study.

Section 4 introduces the flight dynamics model used in all simulations and then presents the main learning experiment, with its set of hyperparameters, that trains an agent to follow a given combination of heading and altitude. Successively, some additional test cases are also reported that validate this DRL-based control strategy.

4 Control strategy validation

The flight dynamics model used for this research is provided by the JSBSim software library. JSBSim is a multi-platform, general-purpose, object-oriented FDM written in C++. The FDM is essentially the physics/-math model that defines the 6-DoF movement of a flying vehicle, subject to the interaction with its natural environment, under the forces and moments applied to it using the various control mechanisms. The mathematical engine of JSBSim solves the nonlinear system (1)–(11)–(17)–(18) of differential equations starting from a given set of initial conditions, with prescribed input laws or determined by a pilot-in-the-loop operating mode. The FDM implements fully customizable, data-driven flight control systems, aerodynamic models, propulsive models, and landing gear arrangement through a set of configuration text files in XML format.

The software can be run as an engineering flight simulator in a standalone batch mode (no graphical displays), or it can be integrated with other simulation environments. JSBSim includes a MATLAB S-function that makes it usable as a simulation block in Simulink. This feature, besides the MATLAB/Simulink Reinforcement Learning Toolbox, has made possible all simulations and learning strategies performed in this research.

A validation of JSBSim as a flight dynamics software library has been reported by several authors [30,37].

4.1 Agent training

Several learning experiments were carried out with various control goals, in order to assess the appropriate tuning of hyperparameters related to the DDPG algorithm and to the reward calculation function. A representative example of agent training is presented here, whose Simulink overall scheme is reported in Fig. 11, while a selection of successful control examples in different scenarios is presented in the next section.

The agent training process was tuned for a flight control task where a reference F-16 FDM was: (i) set in flight at 30,000ft of altitude and at an assigned speed, with a randomly generated initial heading, and (ii) required to reach a target commanded heading $\psi_c = 0$ deg and a target commanded altitude $h_c = 27{,}000$ ft within a time $t_{\mathcal{T}} = 30$ s, (iii) with a final wings level and horizontal fuselage attitude—that is, following zero commanded roll and elevation angles $\phi_c = \theta_c = 0$ deg. The target flight condition is a translational flight, i.e. a motion with zero commanded angular speeds $p_c = q_c = r_c = 0$ deg/s.

Thousands of simulations were performed to train the agent within the MATLAB/Simulink environment and to reach a fine-tuned control for the assigned task. In the initial trials, the agent was trained with the two different types of reward functions presented in Sect. 3.11, and finally, it was determined that the hyperbolic penalty function (40) was the one that gave the best results.

By defining the error variables:

$$\begin{aligned} &\epsilon_h = h_c - h, \quad \epsilon_\phi = \phi_c - \phi, \quad \epsilon_\theta = \theta_c - \theta, \\ &\epsilon_\psi = \psi_c - \psi, \quad \epsilon_p = p_c - p, \quad \epsilon_q = q_c - q \end{aligned} \tag{42}$$

the observation vector $\boldsymbol{\chi}$ in this scenario is defined as follows:

$$\boldsymbol{\chi} = \left[\, \epsilon_h,\, \epsilon_\phi,\, \epsilon_\theta,\, \epsilon_\psi,\, \epsilon_p,\, \epsilon_q,\, \alpha,\, \beta,\, \tilde{\delta}_T,\, \tilde{\delta}_a,\, \tilde{\delta}_e,\, \tilde{\delta}_r,\, \tilde{\delta}_f \,\right] \tag{43}$$

All simulations reaching a terminal state at the final time $t_{\mathcal{T}}$ with an altitude error $|\epsilon_h| > 2000$ ft were marked with a final cumulative reward $R_{\mathcal{T}} = -1000$ (control target unattained).

Table 1 summarizes the initial conditions of all simulations required to train the agent to follow a given heading and altitude. The main hyperparameters of the fine-tuned training process with the DDPG algorithm are reported in Table 2, while the reward function hyperparameters are listed in Table 3. The major training setup options are reported in Table 4. The computational cost to run the simulations on a personal computer equipped with an Intel i7-9750h CPU, a DDR4 RAM of 32 GB

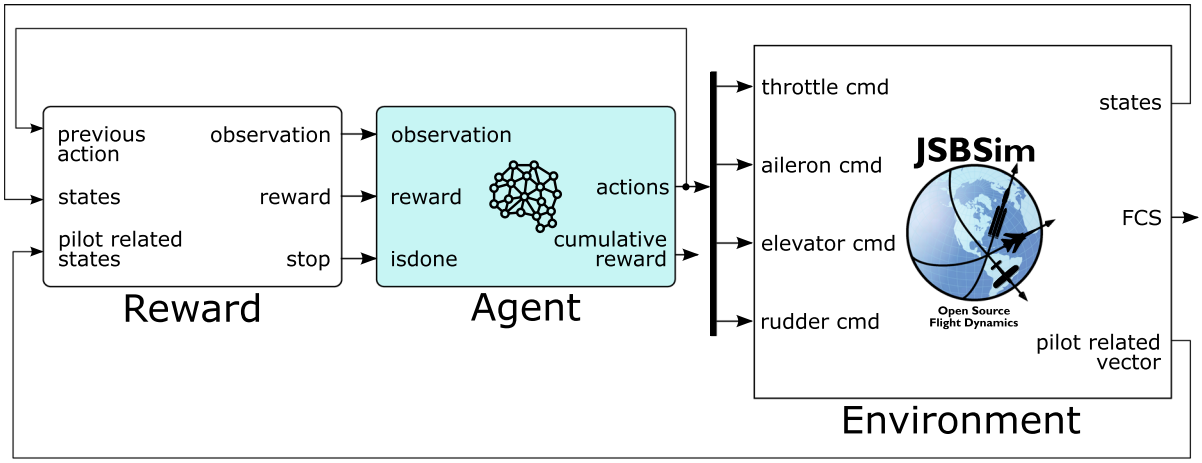


Fig. 11 Simulation scheme in Simulink

Table 1 Initial conditions for the heading and altitude control training

Parameter	Symbol	Value
Altitude	h	30, 000.0 ft (9144.0 m)
First aircraft velocity component	u	750.0 ft/s (228.6 m/s)
Second aircraft velocity component	v	0.0 ft/s (0.0 m/s)
Third aircraft velocity component	w	0.0 ft/s (0.0 m/s)
Latitude	μ	47.0 deg
Longitude	l	122.0 deg
Roll angle	ϕ	0.0 deg
Pitch angle	θ	0.0 deg
Heading angle	ψ	Random $\in$ (0, 360) deg

Table 2 Agent hyperparameters for the heading and altitude control training

Parameter	Value
Sample time	0.2
Batch size	256
Noise standard deviation decay	1×10^{-5}
Actor learning rate	1×10^{-3}
Actor gradient threshold	1
Critic learning rate	1×10^{-3}
Critic gradient threshold	1

Table 5 Statistics for the heading and altitude control agent training scenario (hardware: Intel i7-9750 h CPU, DDR4 32 GB of RAM, Nvidia RTX2060 GPU)

Performed episodes	2889
Maximum obtained reward	−24.60
Elapsed time	6 h 51 min

and an Nvidia GPU RTX2060 is summarized in Table 5. Finally, the cumulative reward history, R_e as a function of the number of episodes, is plotted in Fig. 12.

4.2 Simulation scenarios

This section presents the results of various test case simulations where different control tasks are successfully accomplished by the same agent presented in Sect. 4.1.

With reference to the scheme of Fig. 5, the pilot's commanded inputs are essentially replaced by the agent's action on the primary controls—i.e. stick and pedals ($\tilde{\delta}_a$, $\tilde{\delta}_e$, $\tilde{\delta}_r$) and the throttle lever, $\tilde{\delta}_T$—forming the four-dimensional input vector $\tilde{\boldsymbol{u}}_{agt}$ defined in (20). These signals are filtered by the FCS, whose output is then converted in aerosurface deflections and actual throttle setting, and passed to the aircraft dynamics simulation block (directly interfaced to JSBSim) besides other control effector signals produced by the FCS logics. These form the full input vector $\boldsymbol{u}$ defined in (21).

Table 3 Hyperbolic penalty parameters for the heading and altitude control scenario

Parameter	Lower bound ($x_{\min}$)	Upper bound ($x_{\max}$)	λ	τ
Altitude error (ϵ_h)	−5	5	1/398	0
Roll rate (ϵ_p)	−0.1	0.1	125/78	0
Pitch rate (ϵ_q)	−0.1	0.1	125/78	0
Roll angle error (ϵ_ϕ)	−0.02	0.02	125/78	0
Pitch angle error (ϵ_θ)	−0.02	0.02	125/78	0
Heading angle error (ϵ_ψ)	−0.02	0.02	125/78	0

Table 4 Training options for the heading and altitude control scenario

Parameter	Value
Maximum number of episodes	5000
Maximum number of steps per episode	150
Stop training criteria	Average reward (of five successive episodes) ≥ -50

4.2.1 Heading and altitude following

The heading and altitude following scenario is the one that was actually set up to train the agent and introduced in Sect. 4.1. The details of a representative simulation with control inputs provided by the trained agent are presented here.

The task of achieving a zero heading angle and an assigned new flight attitude is accomplished within the prescribed 30 s. The agent's inputs as well as the FCS outputs are plotted as normalized flight commands time histories in Fig. 13, where the throttle setting values above 1 mean that the jet engine afterburner is being used. The time histories of the primary aerosurface deflections, as actual inputs to the aircraft dynamics model, are shown in Fig. 14. Time histories of aircraft state variables, such as attitude angles and aerodynamic angles, velocity components, normal load factor, Mach number and angular velocity components, are reported in Figs. 16 and 17.

4.2.2 Waypoint following

This test case generalizes the previous scenario by introducing a number of sequentially generated random waypoints during the simulation, that the aircraft is required to reach under the agent's control. As shown in Fig. 18, the same agent that was trained to accomplish the simpler task presented in the previous example, is now deployed in a new simulation scenario where the reference values ψ_c and h_c change over time. The additional logic with respect to the previous case receives a multi-dimensional reference signal, calculates the error terms, and injects them into a reward calculation block. This reward thus calculated directs the agent to follow a given flight path marked by multiple waypoints.

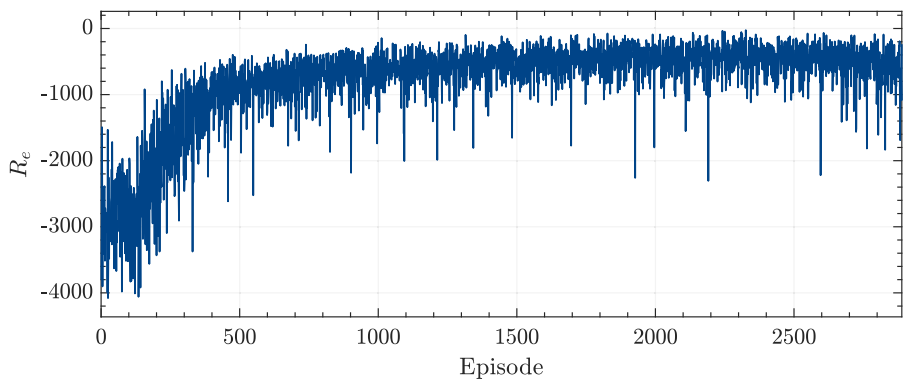


Fig. 12 Episode reward history during the training for the heading and altitude control scenario

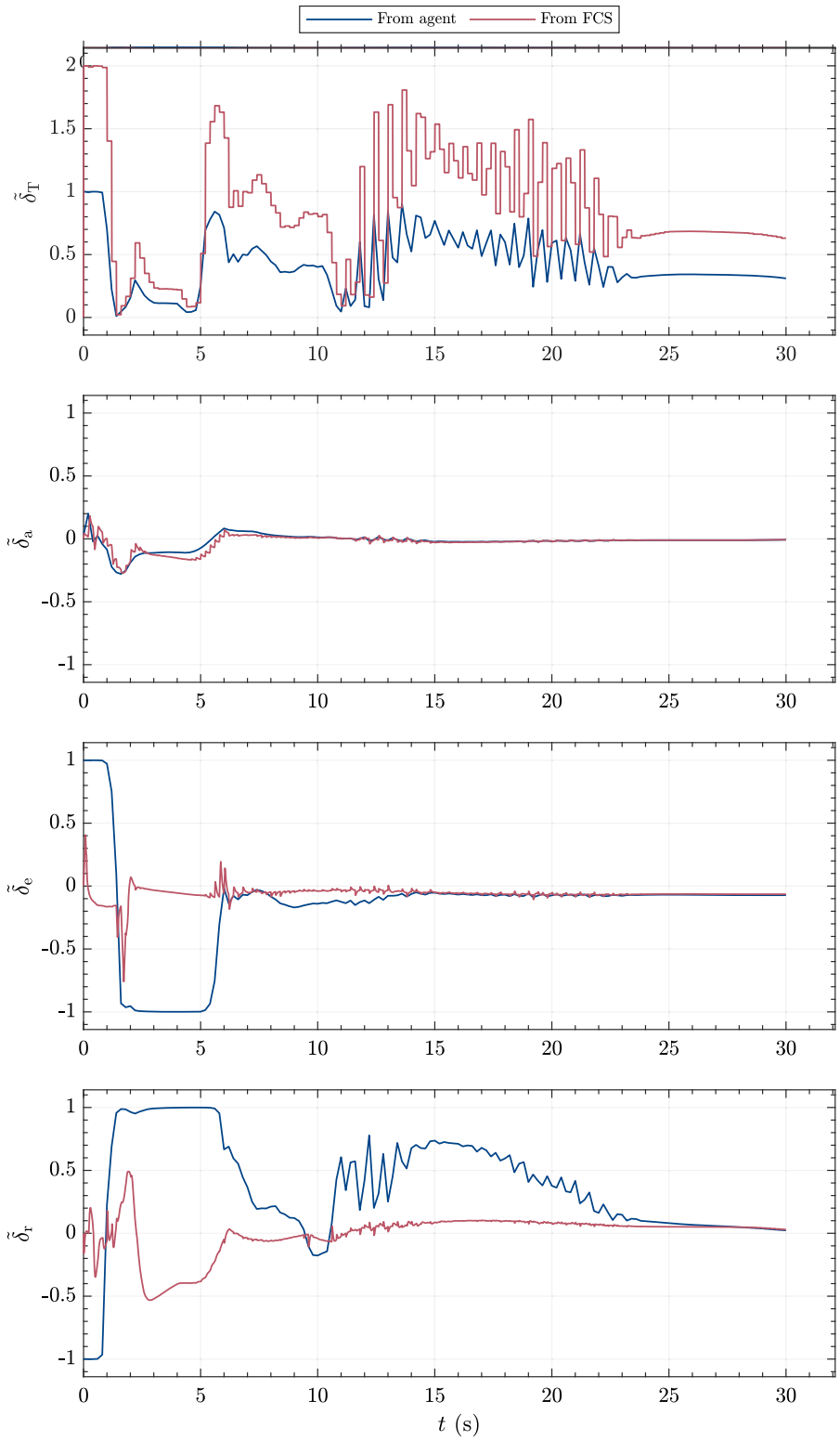


Fig. 13 F-16 agent-controlled heading and altitude following simulation scenario. Normalized flight commands histories, as provided by the agent and filtered by the FCS

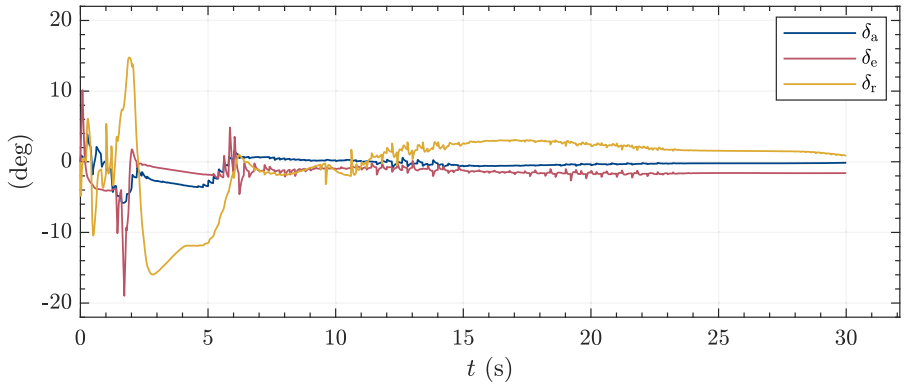


Fig. 14 F-16 agent-controlled heading and altitude following simulation scenario. Actual primary aerosurface deflections correspond to the command inputs coming from the FCS. See also Fig. 5

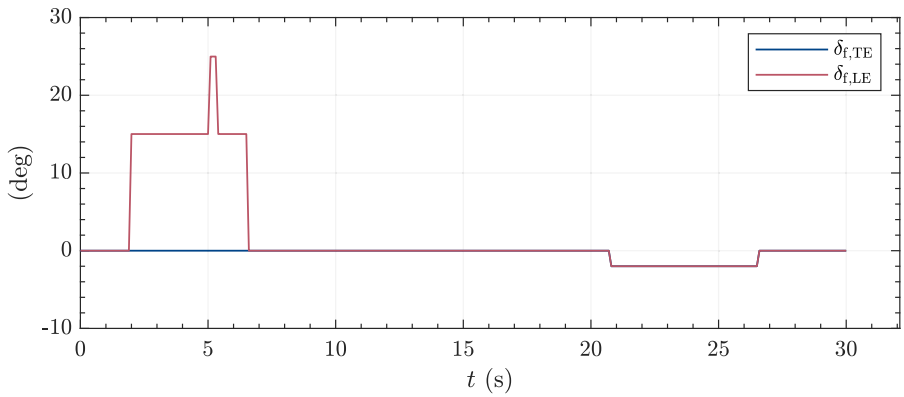


Fig. 15 F-16 agent-controlled heading and altitude following simulation scenario. Aerosurface deflections $\delta_{\mathrm{f,TE}}$ and $\delta_{\mathrm{f,LE}}$ generated by the FCS logics. See also Fig. 5

The waypoints form a discrete sequence $\{(l_{\mathrm{c}}, \mu_{\mathrm{c}}, h_{\mathrm{c}})_i \mid i = 1, 2, \ldots, n\}$ of n locations in space of geographic coordinates (l, μ) and altitude h generated at subsequent time instants $t_1, t_2, \ldots, t_n$. In the example presented here, a sequence of $n = 10$ waypoints has been considered.

Starting from a random initial flight condition, with casual heading, speed and altitude, the goal is to reach the next random waypoint, and successively one by one all the other waypoints as they reveal themselves to the agent along the way. For $0 = t_0 \leq t < t_1$ the aircraft points to waypoint $(l_{\mathrm{c}}, \mu_{\mathrm{c}}, h_{\mathrm{c}})_1$ until at $t = t_1$ the vehicle is labelled as sufficiently close to the first target, the waypoint $(l_{\mathrm{c}}, \mu_{\mathrm{c}}, h_{\mathrm{c}})_2$ is generated and pursued for $t_1 \leq t < t_2$; the scheme repeats itself until the last waypoint is reached after time t_n. The geographic coordinates and the general aeroplane position tracking are handled through the JSBSim internal Earth model.

In the generic instant t of the simulation, with the aircraft gravity centre having geographic longitude $l(t)$ and latitude $\mu(t)$, once the vehicle is commanded to fly towards the next ith waypoint, the heading $\psi_{\mathrm{c},i}$ that the agent is required to follow after time t_{i-1} is calculated as:

$$\psi_{\mathrm{c},i}(t) = \operatorname{atan2}\Big\{ \sin\big[l_{\mathrm{c},i} - l(t)\big] \cos\mu(t), \cos\mu(t) \sin\mu_{\mathrm{c},i} - \sin\mu(t) \cos\mu_{\mathrm{c},i} \cos\big[l_{\mathrm{c},i} - l(t)\big]\Big\} \tag{44}$$

for $t_{i-1} \leq t < t_i$. Therefore, the altitude and heading error considered in the reward calculation function become:

$$\epsilon_h(t) = h_{\mathrm{c},i} - h(t), \quad \epsilon_\psi(t) = \psi_{\mathrm{c},i} - \psi(t) \tag{45}$$

for $t_{i-1} \leq t < t_i$ and $i = 1, \ldots, n$. The above formulas represent the time-varying references provided to the control agent.

The agent looks at one waypoint at a time in this particular test case. When the aircraft reaches an assigned threshold distance from the current pointed waypoint, the next one is generated and passed to the agent. The instantaneous distance from the aircraft gravity centre to the current waypoint is calculated with the Haversine formula [38].

The behaviour of the agent in the multiple waypoints following task can be figured out by the simulation

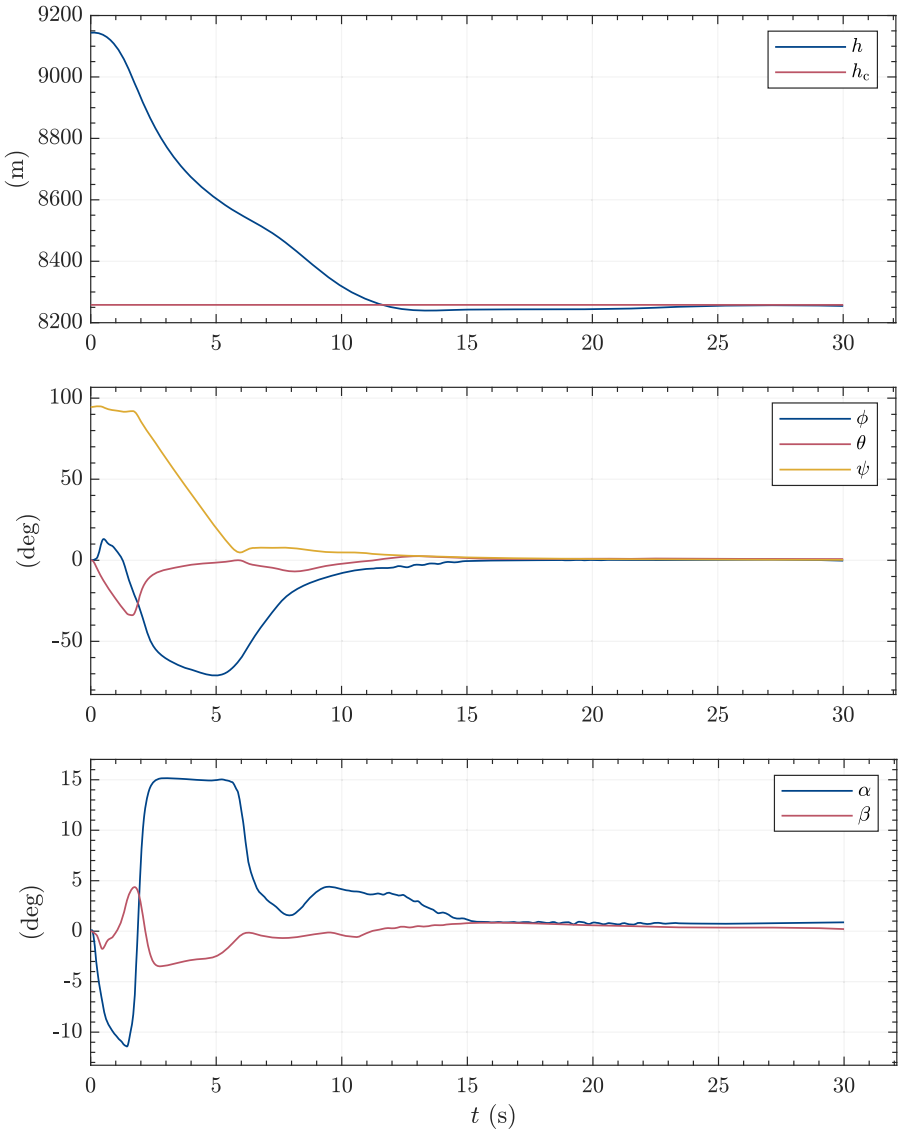


Fig. 16 F-16 agent-controlled heading and altitude following simulation scenario. Time histories of altitude, attitude angles and aerodynamic angles

results presented below. The agent's inputs as well as the FCS outputs are plotted as normalized flight commands time histories in Fig. 19. The time histories of the primary aerosurface deflections, as actual inputs to the aircraft dynamics model, are shown in Fig. 20. Time histories of aircraft state variables, such as attitude angles and aerodynamic angles, velocity components, normal load factor, Mach number and angular velocity components, are reported in Figs. 21 and 22.

Figure 23 reports the actual and commanded values of heading ψ, longitude l and latitude μ. This scenario is also represented on the map of Fig. 24 reporting the ground track of the aircraft trajectory, the sequence of 10 waypoints and the commanded waypoint altitudes. Finally, the three-dimensional flight path and aircraft body attitude evolution are shown in Fig. 25.

4.2.3 Varying target with sensor noise

This test case is set up to investigate the agent's behaviour when external disturbances are injected into the environment. The simulation scenario is similar to the heading and altitude following example presented in Sect. 4.2.1, but in this case the commanded altitude h_c and heading ψ_c do change in time according to assigned stepwise constant functions. In addition, the

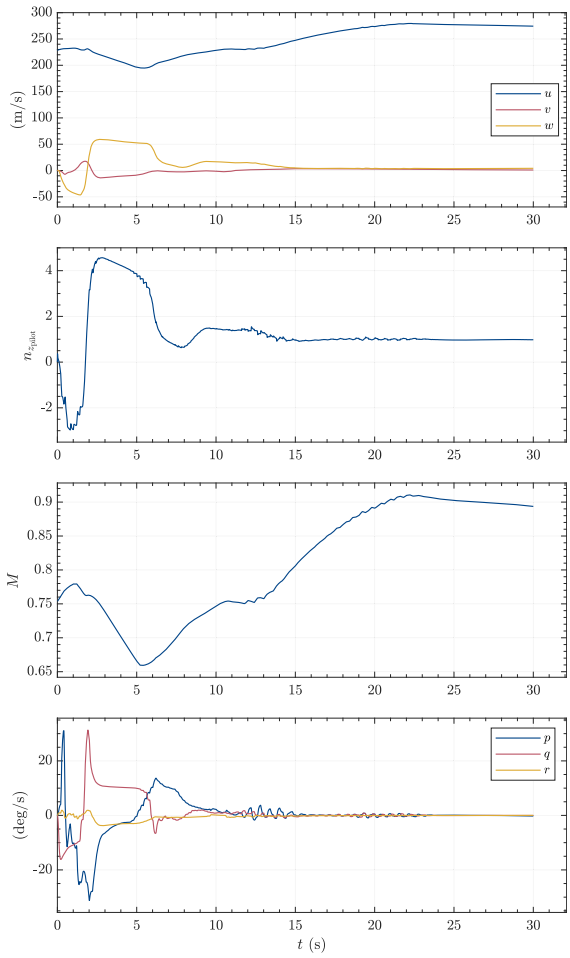


Fig. 17 F-16 agent-controlled heading and altitude following simulation scenario. Time histories of velocity components, normal load factor, Mach number and angular velocity components

state signals that define the agent's observation vector (43) and that are sent also as feedback to the FCS are perturbed with prescribed noisy disturbances.

In particular, a set of random (Gaussian) normally distributed noise signals are generated and then added to some state variables in order to simulate an uncertainty on data available to the controllers. The results of a simulation case with zero-mean perturbations—assuming an accurately calibrated set of sensors, which is typically expected from the class of aircraft considered in this study—are reported in Fig. 26. Table 6 lists the main parameters of the additive noisy signals.

The results of this simulation case are similar to those reported in previous examples. The outcome of the agent's control actions is represented by the time histories plotted in Fig. 26, which shows the instantaneous aircraft altitude and heading beside their corresponding commanded values. The same figure also reports the aerosurface deflection angles as provided to the aircraft FDM.

4.2.4 Prey–chaser scenario

This test case evolves from the waypoint following scenario presented in Sect. 4.2.2 and provides a basis for possible applications of the present agent-based control approach to the field of military fighter pilot training (to dogfight and formation flight, for instance). In a prey–chaser scenario, two aeroplane models coexist and share the same flight environment where the first aircraft acts as the prey, being chased by the second one, i.e. the chaser.

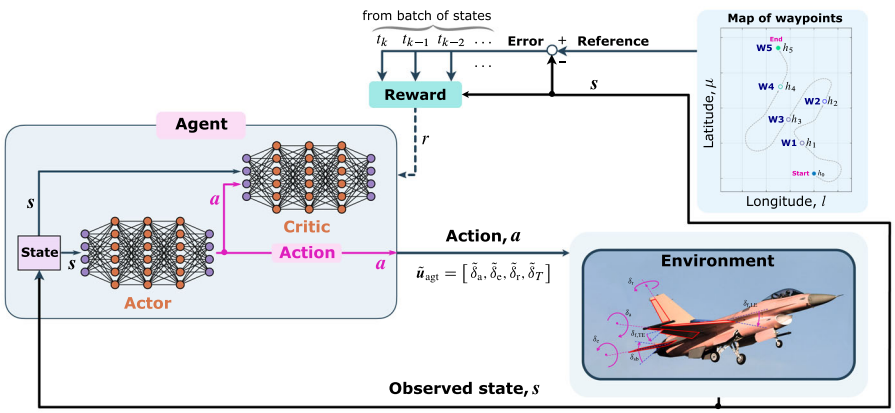


Fig. 18 Control scheme of a waypoint following scenario. The reference given by successive waypoints and altitudes are injected into the reward calculation

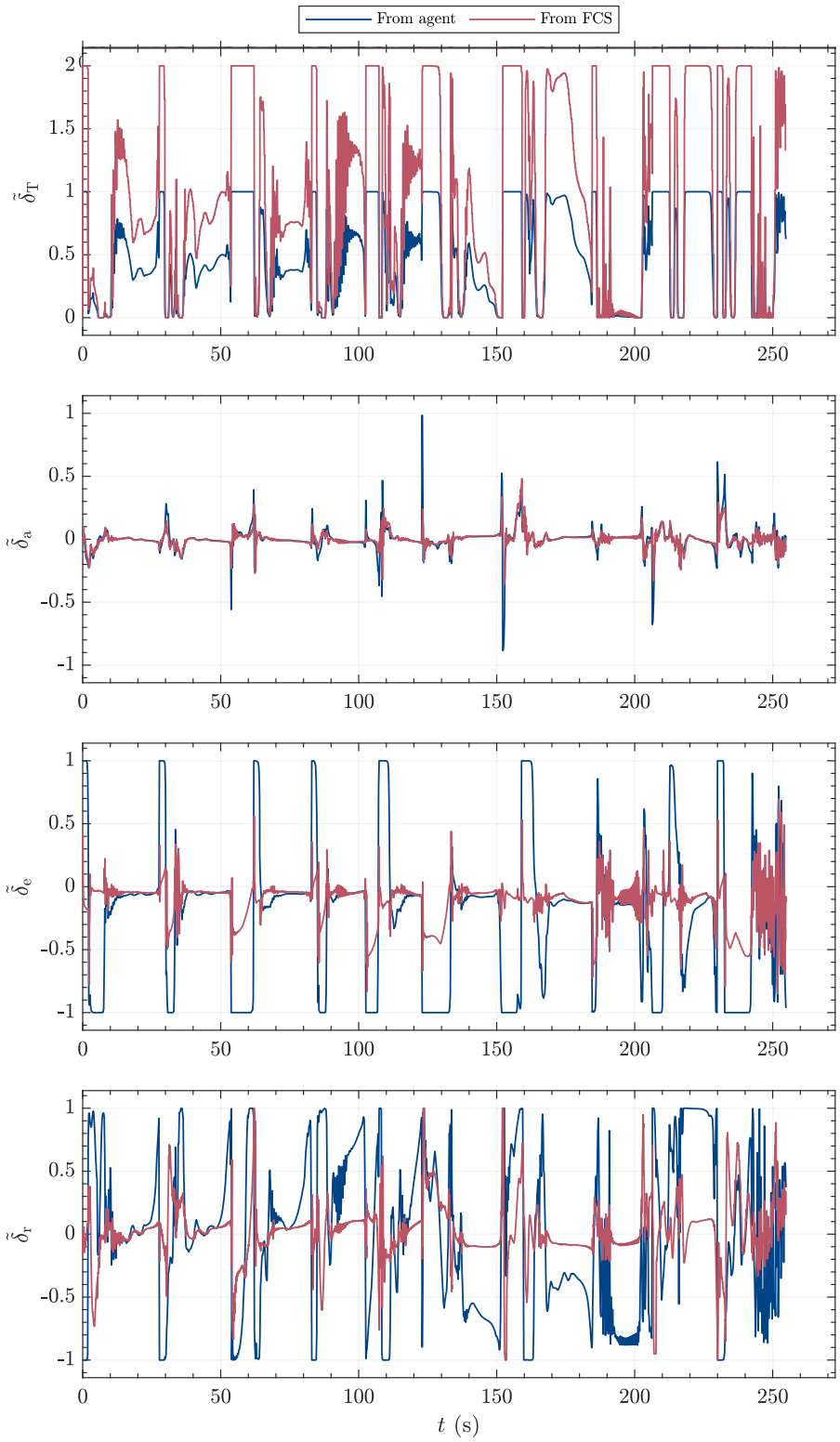


Fig. 19 F-16 agent-controlled waypoint following simulation scenario. Normalized flight commands histories, as provided by the agent and filtered by the FCS

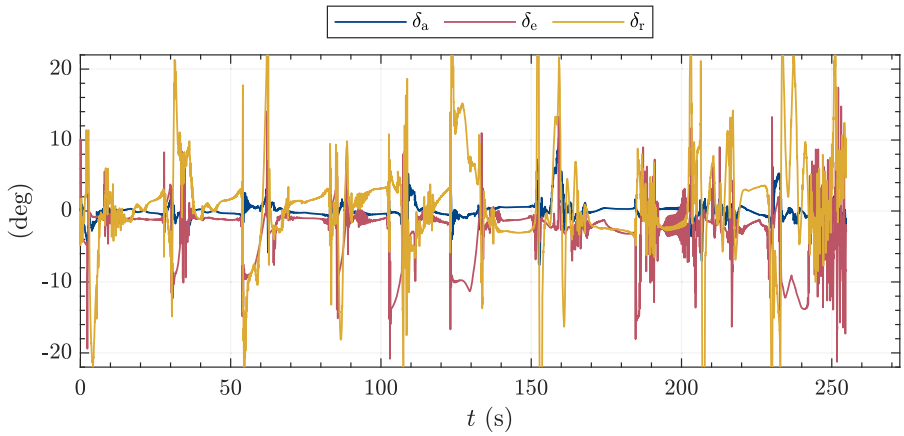


Fig. 20 F-16 agent-controlled waypoint following simulation scenario. Actual primary aerosurface deflections corresponding to the command inputs coming from the FCS

In the particular test case presented here, both the chaser and the prey are piloted by an RL-based agent. In the simulation environment, there are two replicas of the same F-16 model, both of them piloted by two identical trained agents (Agent vs. Agent, the same presented in Sect. 4.1). The first agent controls the prey aircraft to follow a given sequence of random waypoints, much as the previous multiple waypoints following example. The second agent controls the chaser aircraft by acquiring the successive positions of the prey with a given frequency and follows them as if they were virtual waypoints.

The prey–chaser interaction within the waypoint subsequence 1 to 8 is shown in detail by the maps of Fig. 27. The full picture for the full waypoint sequence 1 to 10 is shown in Fig. 28, where also the random initial positions of the two aeroplanes are marked.

This particular simulation demonstrates the ability of the chaser agent to tighten its trajectory when appropriate, for instance, when the prey passes from waypoint 5 to waypoint 6. Another interesting behaviour is observed when the chaser aircraft overtakes the prey when the waypoint 8 is reached and surpassed. In this case, the agent-controlled chaser aircraft performs a complete turn to position itself behind the prey and continue the target following task.

5 Discussion

All simulation examples introduced in the previous section demonstrate the validity of the trained agent when it is directed to execute different control tasks with a progressive level of difficulty.

5.1 Path control with fixed reference

The simplest example showing a case of path control with a fixed reference is presented in Sect. 4.2.1. The assigned control task is precisely what the agent was trained for, i.e. a case of exploitation. The aircraft is able to reach a target heading angle and an assigned new flight altitude, starting from a randomly generated initial flight condition (random heading, altitude, and speed), within a flight time of 30 s. The agent achieves this result by providing the input actions—normalized flight commands—shown in time histories of Fig. 13. The agent inputs are filtered by the FCS, whose output signals are also plotted in the same figure. In particular, it is seen that to reach the assigned goal as quickly as possible the agent demands a high thrust level, thus requiring the use of the jet engine afterburner (see the topmost plot of Fig. 13, where the output $\tilde{\delta}_T$ from the FCS, for 12 s $\leq t \leq$ 22 s, becomes higher than 1). Time histories of the other primary and secondary controls as well as of the aircraft state variables clearly show an initial left turn, combined with a dive, to reach a prescribed lower altitude, North-pointing flight path. A typical FCS behaviour for this type of high-performance fighter jet is observed in the plot at the bottom of Fig. 13: the output $\tilde{\delta}_r$ from the FCS for 12 s $\leq t \leq$ 22 s exhibits severe filtering of the agent's rudder command in order to keep the sideslip angle as low as possible (see β within the same time interval in Fig. 16). The manoeuvre is confirmed by the time histories of primary aerosurface deflections shown in Fig. 14 and of wing leading-edge/trailing-edge flap deflections reported in Fig. 15. The left turn results from the negative (right) aileron deflection δ_a

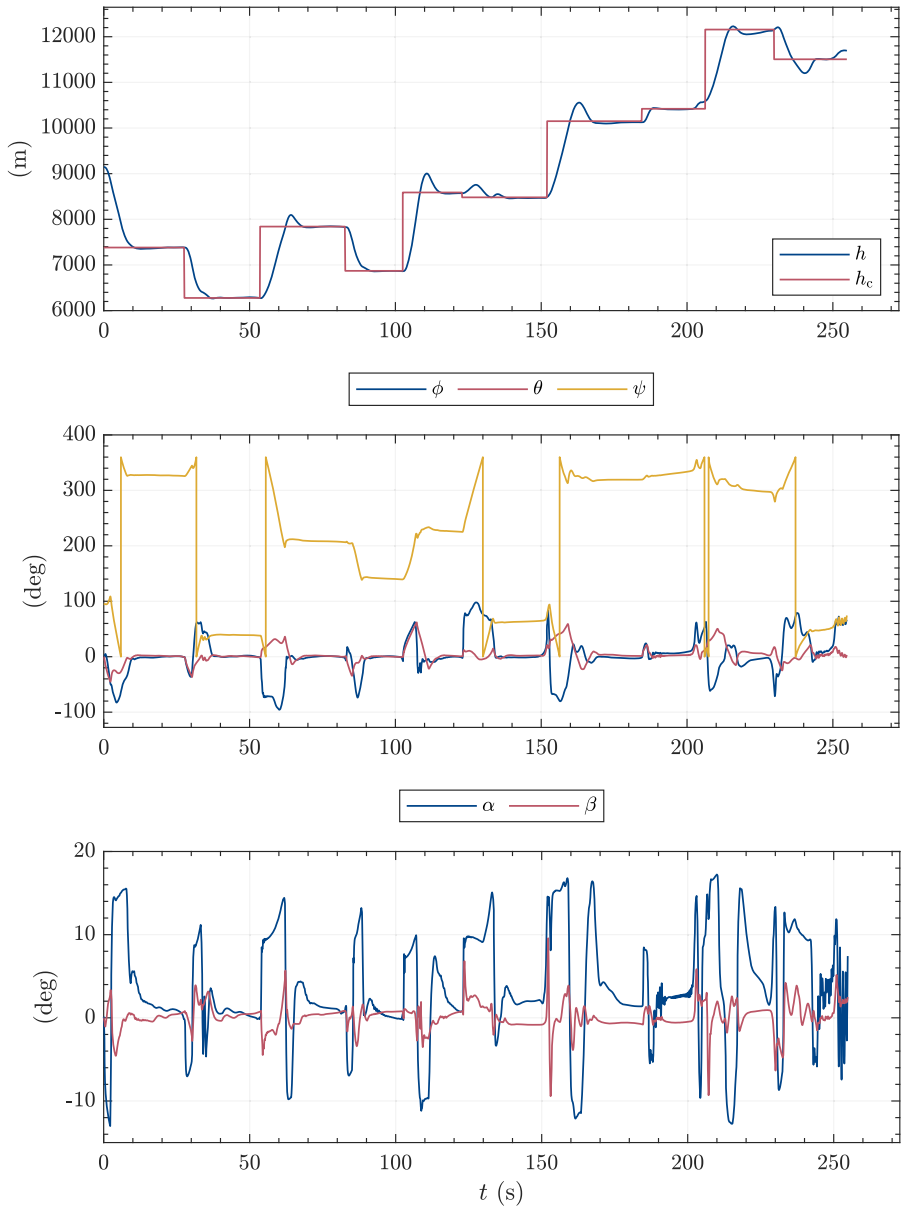


Fig. 21 F-16 agent-controlled waypoint following simulation scenario. Time histories of altitude, attitude angles and aerodynamic angles

(right aileron down, left aileron up), for 1 s $\leq t \leq$ 6 s). The initial dive results from the combined action of tail and wing leading-edge flap deflections, δ_e and $\delta_{f,LE}$, respectively, for 0 s $\leq t \leq$ 6 s, leading to a pitch-down manoeuvre (see negative θ within the same time interval in Fig. 16). Figure 15 reveals an inherent complexity of such a controlled flight example, showing time-varying deflections of leading-edge and trailing-edge flaps mounted on the main wing. These deflections are due to the high-fidelity implementation of FCS control logics that can trigger the actuation of high-lift devices when some aircraft state variables do fall within prescribed ranges (see Appendix C). Time histories of aircraft state variables—such as altitude, attitude angles, aerodynamic angles reported in Fig. 16, velocity components, normal load factor, flight Mach number and angular velocity components reported in Fig. 17—confirm the left turn/dive manoeuvre to reach the prescribed terminal state. From Fig. 16, in particular, the errors $\epsilon_h = h_c - h$, $\epsilon_\psi = -\psi$, $\epsilon_\theta = -\theta$, $\epsilon_\phi = -\phi$ and their vanishing behaviour with time are easily deduced and confirm the effectiveness of the

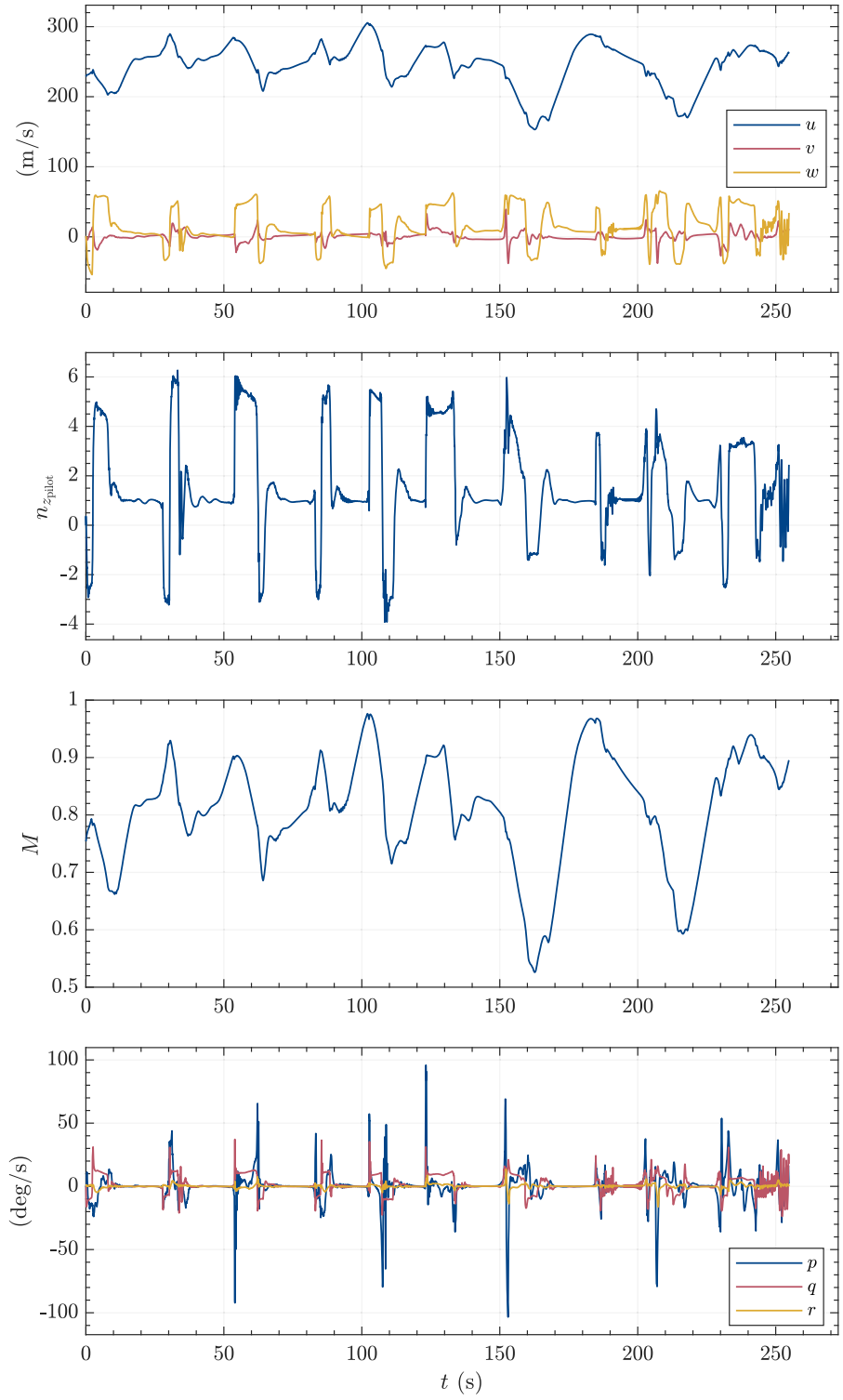


Fig. 22 F-16 agent-controlled waypoint following simulation scenario. Time histories of velocity components, normal load factor, Mach number and angular velocity components

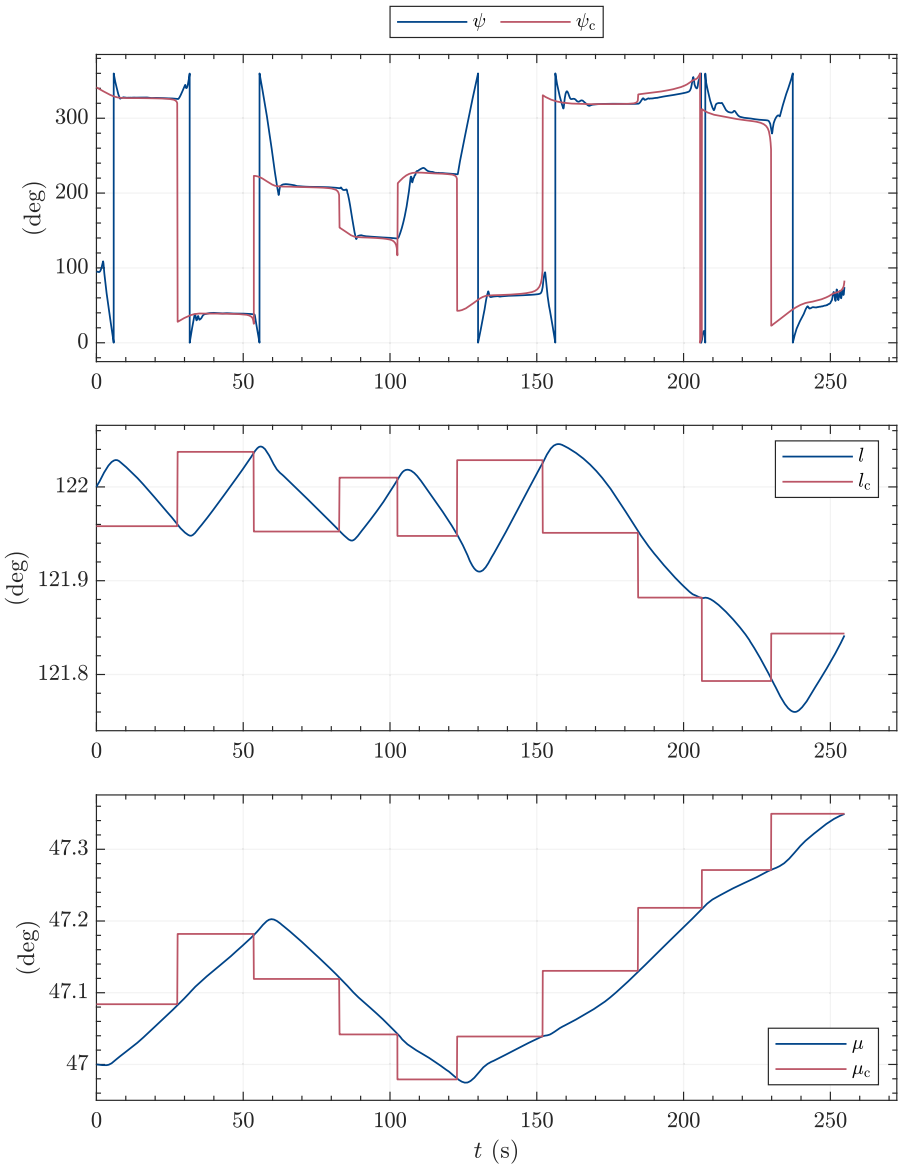


Fig. 23 F-16 agent-controlled waypoint following simulation scenario. Actual and commanded values of heading ψ, longitude l and latitude μ. See also Eq. (44)

agent's control actions. This test case clearly shows a valid AI-based control in the presence of nonlinear effects. For instance, the significant variations of altitude, flight Mach number and angle of attack do trigger all those nonlinearities accurately modelled in the aircraft FDM (see Appendices A, and B).

5.2 Path control with varying reference

A case of multiple waypoints following task is introduced in Sect. 4.2.2. This scenario provides an interesting example of controlled flight with a moving reference. The simulation is performed by using the same control agent that was trained for the simpler one-reference heading/altitude following task. Interestingly, the simulation results presented in Figs. 19 to 23, demonstrate that also the multiple waypoints following exercise is successfully achieved. The agent reacts appropriately to each randomly generated new reference and effectively enacts its policy to control the plant dynamics. The ground track reported in Fig. 24 and the three-dimensional trajectory presented in Fig. 25 show the sequence of turns, dives and climbs that are flown to accomplish the assigned task.

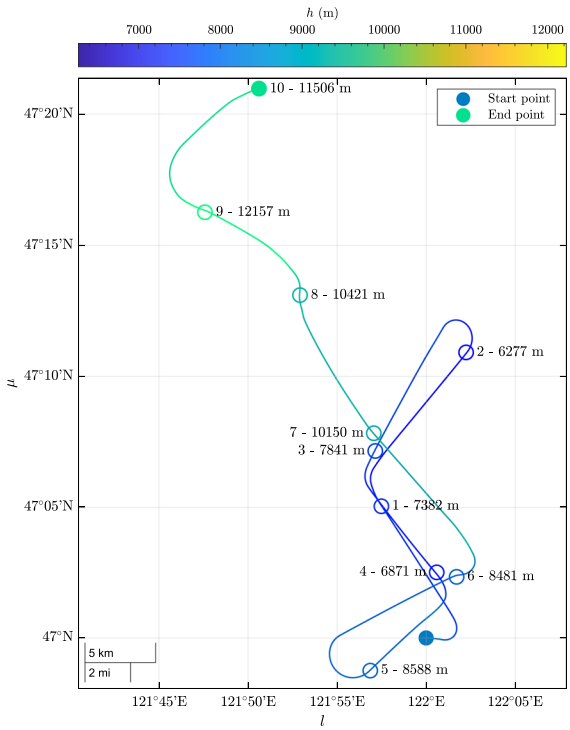


Fig. 24 F-16 agent-controlled waypoint following simulation scenario. Ground track of the aircraft trajectory, sequence of assigned waypoints and commanded waypoint altitudes

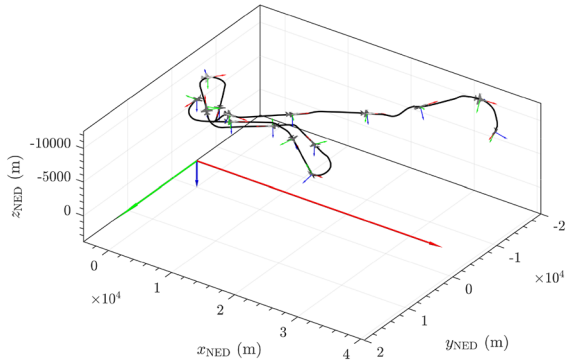


Fig. 25 F-16 agent-controlled waypoint following simulation scenario. Three-dimensional flight path and aircraft body attitude evolution. Aircraft geometry not in scale (magnified 1250 times for clarity)

Table 6 Aircraft states variables affected by additive zero-mean noisy disturbances in the test case of Sect. 4.2.3

Perturbed states	Unit	Variance (σ^2)
p, q, r	rad/s	0.01
h	m	1.0
ψ, θ, ϕ	rad	0.01
α, β	rad	0.01

5.3 Control validation in the presence of noise

Section4.2.3 reports an example of how the trained agent behaves when the state observations are perturbed by additive noisy signals. This test case, with the simulation results shown in Fig. 26, proves the robustness of the agent's control actions with respect to external disturbances, within the assumption of well-calibrated sensors.

5.4 Simulation of a prey–chaser scenario

Finally, Sect. 4.2.4 presents an interesting prey–chaser simulation scenario, demonstrated by the ground tracks reported in Figs. 27 and 28. Two instances of the same trained agent direct two instances, respectively, of the same model of flying vehicle simulating an air engagement. A virtual simulation environment based on the FlightGear flight simulation software[1] has been set up as a means to visualize the air combat in a proper scenery. The scheme of Fig. 29 depicts how the two aircraft instances and their states are represented in the chosen airspace. Four successive screen captures of the virtual simulation environment are represented in Fig. 30, while the prey aircraft, after having reached waypoint 3, is pursuing waypoint 4. The figure shows two camera views on the left, taken following the prey and the chaser aircraft at fixed distances, respectively. The views are synchronized to the evolving ground tracks shown on the right. As seen in this excerpt of simulation, the chasing fighter enters into the field of view of the first camera as it flies at a higher Mach number, while the leading aircraft is still pursuing waypoint 4. Eventually, the chaser reaches its target arriving at the prey's tail. This test case is one example of several other interesting simulation possibilities. In fact, assuming, for example, that the chaser is piloted by the agent discussed here, the prey can be piloted by a completely different agent instructed to accomplish a prescribed manoeuvre or, as an alternative, the prey can be a human-in-the-loop piloted model. These are possible applications of the flight control approach presented in this study that have the potential to enhance pilot training procedures by means of AI-augmented simulation environments.

[1] www.flightgear.org.

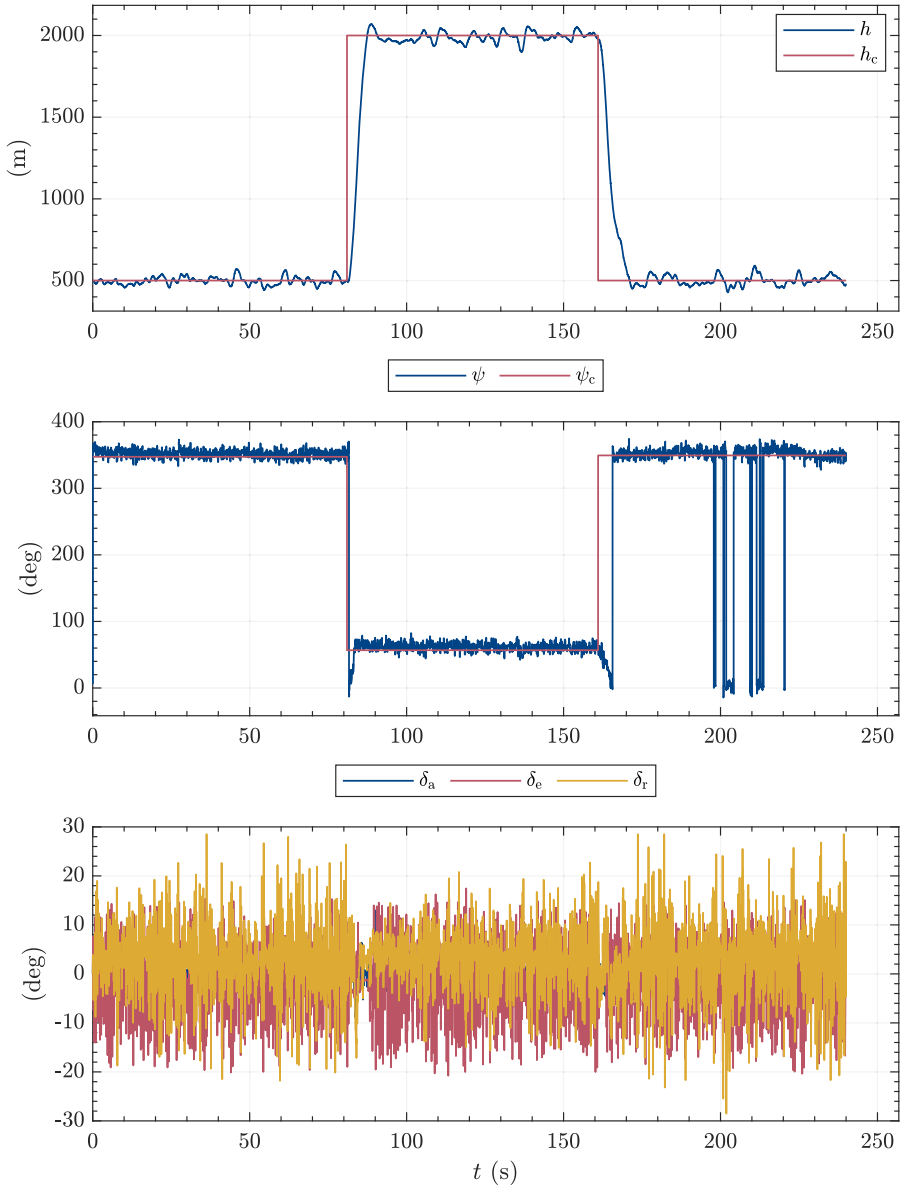


Fig. 26 F-16 agent-controlled simulation scenario with varying commanded heading and altitude, and zero-mean sensor noise. Time histories of altitude, heading and aerosurface deflection angles

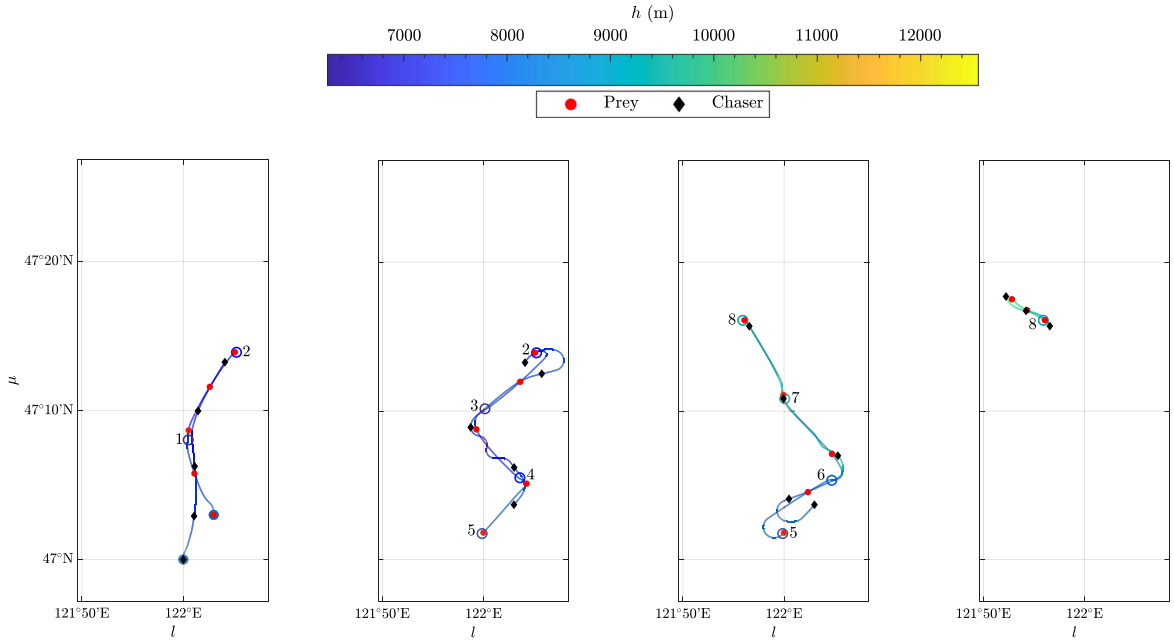
6 Conclusion

This research presents a high-performance aircraft flight control technique based on reinforcement learning and provides an example of how AI can generate a valid controller. The proposed approach is validated by using a reference simulation environment where the nonlinear, high-fidelity flight dynamics model of a military fighter jet is used to train an agent for a selected set of controlled flight tasks. The simulation results show the control effectiveness to make certain manoeuvres fully automatic in highly dynamic scenarios, even in the presence of sensor noise and atmospheric disturbances.

A future research direction that should evolve from this study is the comparison of the proposed AI-based controller to other standard types of flight control.

Supplementary information

This article has no accompanying supplementary file.



(a) $t \in [1, 50.33]$ (s) (b) $t \in [50.34, 145.77]$ (s) (c) $t \in [145.78, 233.21]$ (s) (d) $t \in [233.22, 246.11]$ (s)

Fig. 27 Details of the waypoint sequence 1–8, and of the two ground tracks in the prey–chaser simulation scenario

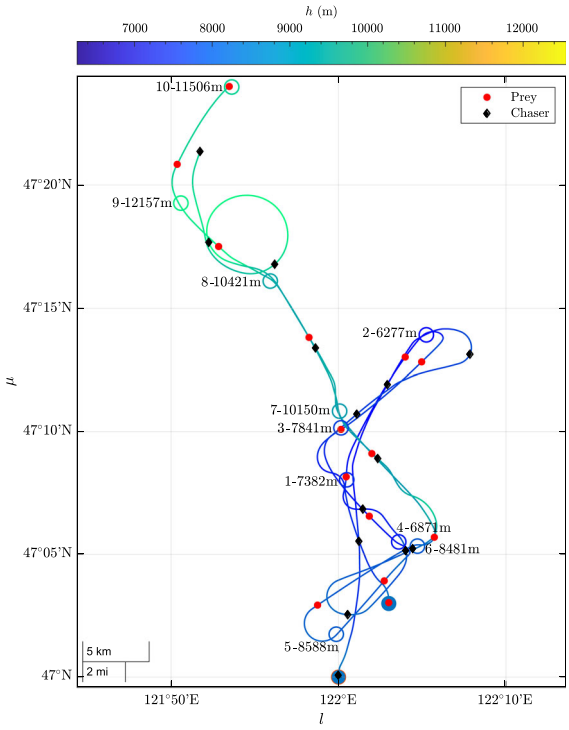


Fig. 28 Map projection of aircraft trajectory, and assigned waypoints, by a prey–chaser scenario simulation

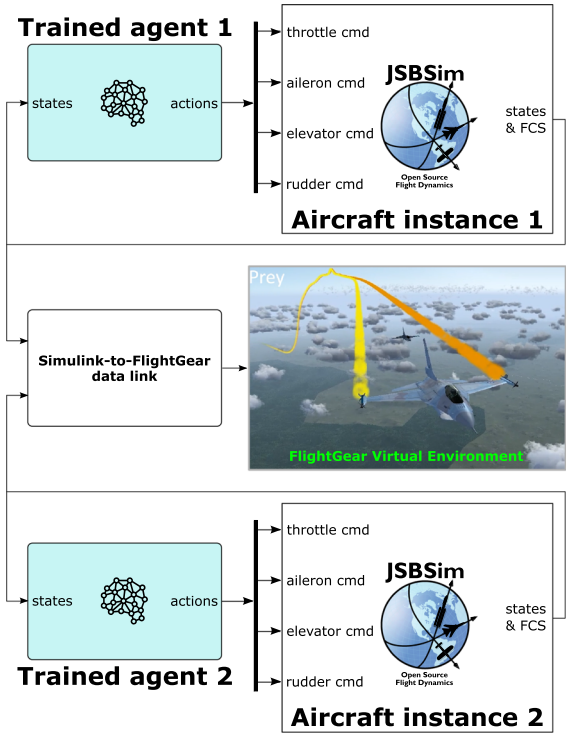


Fig. 29 Virtual simulation environment based on JSBSim, MATLAB/Simulink and FlightGear

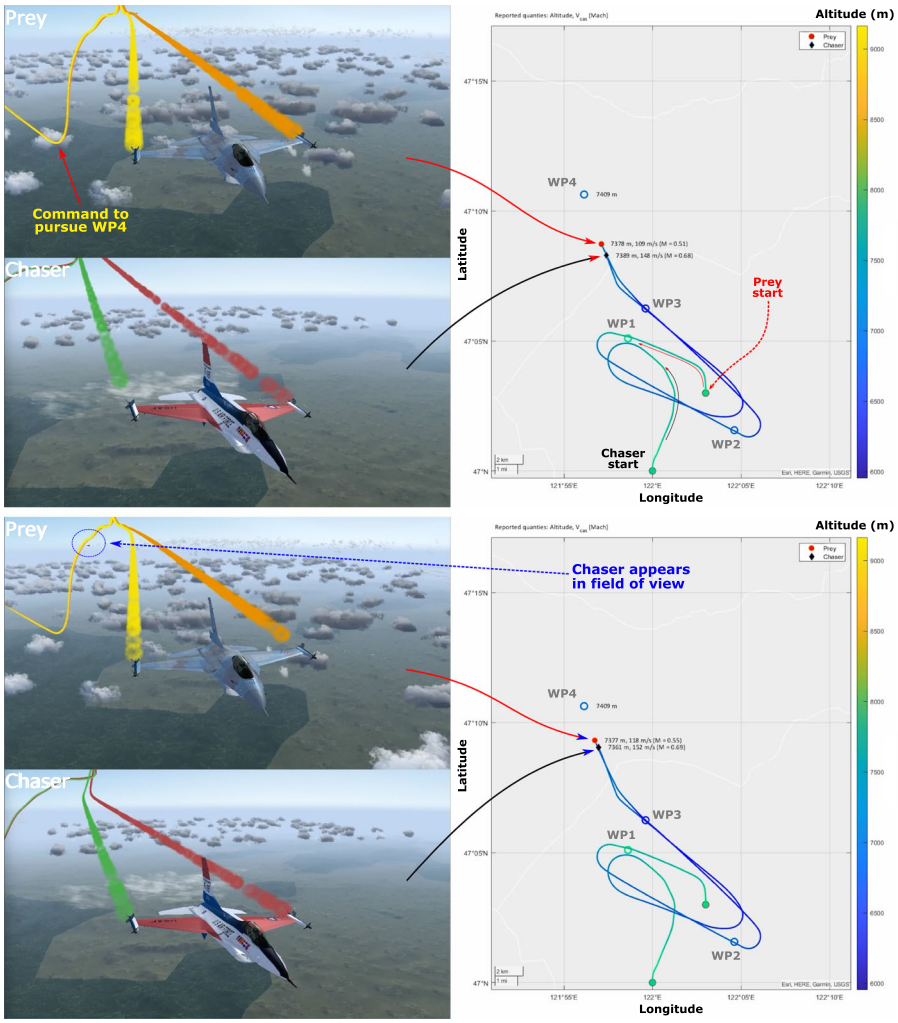
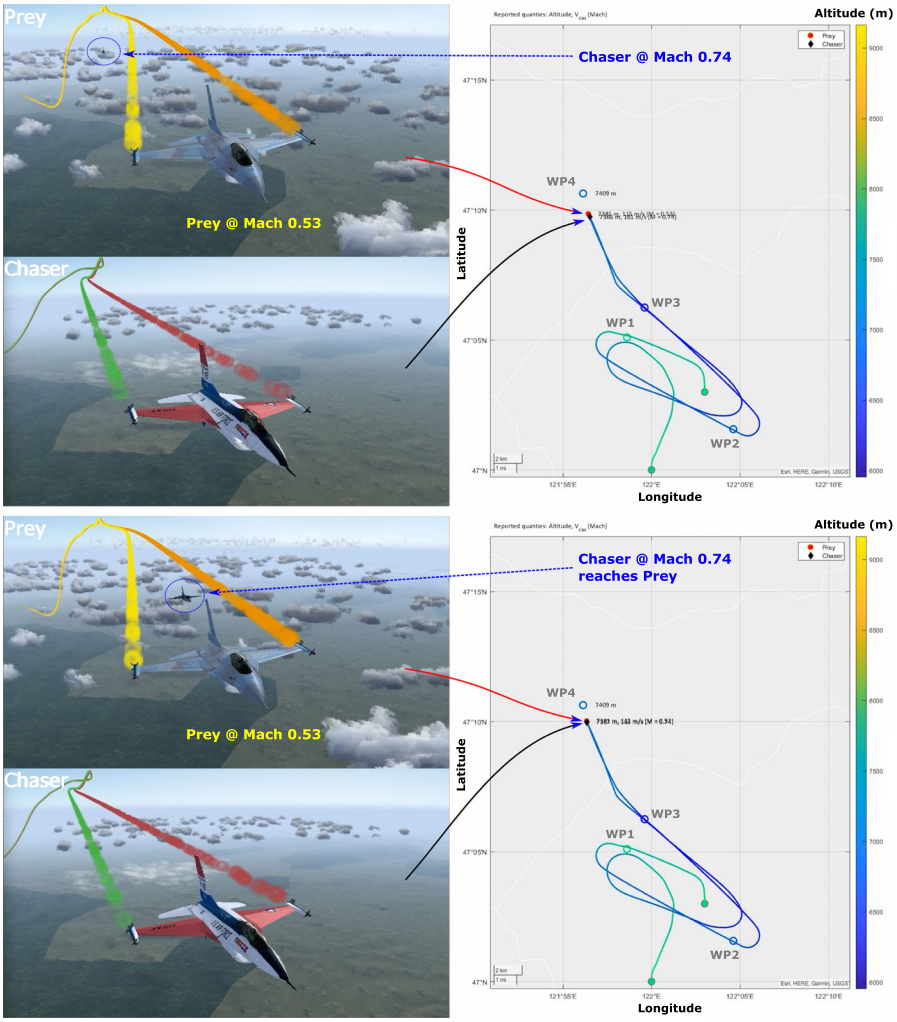


Fig. 30 Screen captures at four successive instants of the one-to-one air combat virtual simulation (times 1 and 2) and (times 3 and 4)

Fig. 30 continued



Acknowledgements The authors would like to thank the editors and reviewers of Nonlinear Dynamics for their valuable efforts in the review of this paper.

Funding Open access funding provided by UniversitÃ degli Studi di Napoli Federico II within the CRUI-CARE Agreement. No funding was received for conducting this study.

Data availability The data that support the findings of this study are available from the corresponding authors, ADM or SM, upon reasonable request.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Consent for publication The authors grant the Journal of Nonlinear Dynamics the authority to publish this work.

Appendix A: Aerodynamic model of the General Dynamics F-16 Fighting Falcon

The high-performance aircraft FDM selected for this research incorporates the nonlinear JSBSim aerodynamic model [30], which we summarize in this appendix. With reference to the definitions given by Eqs. (4), (9), (13) and (14), the aerodynamic coefficients are implemented as functions of the aircraft state and input variables, $\boldsymbol{x}$ and $\boldsymbol{u}$. Each coefficient is expressed as a sum of nonlinear terms, commonly known as the aerodynamic build-up formula. The detailed build-up formulas for the F-16 fighter are reported below, where all angles are in radians.

Lift coefficient

$$\begin{aligned} C_L = k_{C_{L,\mathrm{ge}}}\left(\frac{h}{b}\right)\Big\{ & C_L^{\mathrm{bas}}(\alpha,\delta_{\mathrm{e}}) + C_{L_{\delta_{\mathrm{f,LE}}}}(\alpha)\,\delta_{\mathrm{f,LE}} \\ & + C_{L_{\delta_{\mathrm{f,TE}}}}\delta_{\mathrm{f,TE}} + C_{L_{\delta_{\mathrm{sb}}}}(\alpha)\,\delta_{\mathrm{sb}} \\ & + \left[C_{L_q}(\alpha) + k_{\delta_{\mathrm{sb}}}(\alpha)\,\delta_{\mathrm{sb}}\right]\frac{qc}{2V}\Big\} \end{aligned} \tag{A1}$$

where the functions $k_{C_{L,\mathrm{ge}}}(h/b)$, $C_L^{\mathrm{bas}}(\alpha,\delta_{\mathrm{e}})$, $C_{L_{\delta_{\mathrm{f,LE}}}}(\alpha)$, $C_{L_{\delta_{\mathrm{sb}}}}(\alpha)$, $C_{L_q}(\alpha)$, $k_{\delta_{\mathrm{sb}}}(\alpha)$ are reported, for the sake of example, in Figs. 31 and 32.

Drag coefficient

$$\begin{aligned} C_D = {} & C_D^{\mathrm{bas}}(\alpha,\delta_{\mathrm{e}}) + \Delta C_D(M) + C_{D_{\delta_{\mathrm{f,LE}}}}(\alpha)\,\delta_{\mathrm{f,LE}} \\ & + C_{D_{\delta_{\mathrm{f,TE}}}}\,\delta_{\mathrm{f,TE}} \\ & + C_{D_{\delta_{\mathrm{sb}}}}(\alpha)\,\delta_{\mathrm{sb}} + C_{D_{\tilde{\delta}_{\mathrm{gear}}}}\tilde{\delta}_{\mathrm{gear}} \\ & + \left[C_{D_q}(\alpha) + k_{\delta_{\mathrm{f,LE}}}(\alpha)\,\delta_{\mathrm{f,LE}}\right]\frac{qc}{2V} \end{aligned} \tag{A2}$$

$\tilde{\delta}_{\mathrm{gear}}$: normalized gear position (0: retracted; 1: fully deployed)

Cross force coefficient

The cross force coefficient C_C that appears in Eq. (9) is also called C_Y by many authors. For the F-16, the side force coefficient is modelled as follows:

$$\begin{aligned} C_Y = {} & \left[C_{Y_\beta} + \Delta C_{Y_\beta}(M)\right]\beta + C_{Y_{\delta_{\mathrm{a}}}}\delta_{\mathrm{a}} + C_{Y_{\delta_{\mathrm{r}}}}\delta_{\mathrm{r}} \\ & + \left[C_{Y_p}(\alpha)\,p + C_{Y_r}(\alpha)\,r\right]\frac{b}{2V} \end{aligned} \tag{A3}$$

Roll moment coefficient

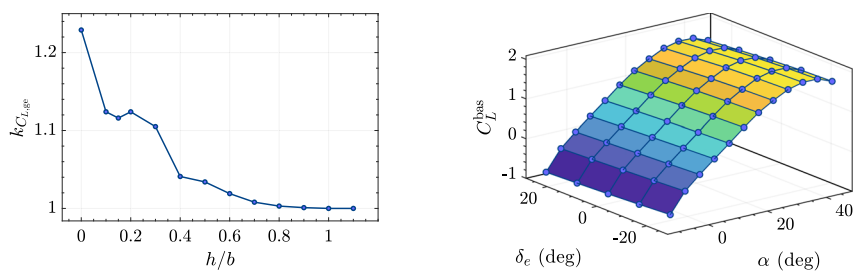
$$\begin{aligned} C_{\mathcal{L}} = {} & C_{\mathcal{L}_\beta}^{\mathrm{bas}}(\alpha,\beta) + \Delta C_{\mathcal{L}_\beta}(M)\,\beta \\ & + \left[C_{\mathcal{L}_p}(\alpha)\,p + C_{\mathcal{L}_r}(\alpha)\,r\right]\frac{b}{2V} \\ & + \left[C_{\mathcal{L}_{\delta_{\mathrm{a}}}}(\alpha,\beta) + \Delta C_{\mathcal{L}_{\delta_{\mathrm{a}}}}(M)\,\alpha\right]\delta_{\mathrm{a}} \\ & + \left[C_{\mathcal{L}_{\delta_r}}(\alpha,\beta) + \Delta C_{\mathcal{L}_{\delta_r}}(M)\,\alpha\right]\delta_r \end{aligned} \tag{A4}$$

Pitch moment coefficient

$$\begin{aligned} C_{\mathcal{M}} = {} & C_{\mathcal{M}}^{\mathrm{bas}}(\alpha,\delta_{\mathrm{e}}) + \Delta C_{\mathcal{M}_\alpha}(M)\,\alpha \\ & + C_{\mathcal{M}_{\delta_{\mathrm{sb}}}}(\alpha)\,\delta_{\mathrm{sb}} + C_{\mathcal{M}_q}(\alpha)\,\frac{qc}{2V} \end{aligned} \tag{A5}$$

Yaw moment coefficient

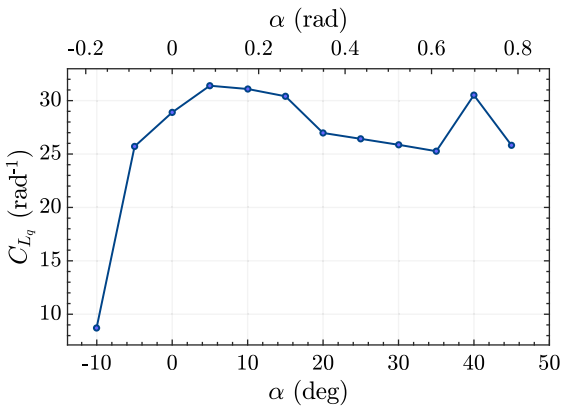
$$C_{\mathcal{N}} = C_{\mathcal{N}}^{\mathrm{bas}}(\alpha,\beta) + \Delta C_{\mathcal{N}_\beta}(M)\beta$$



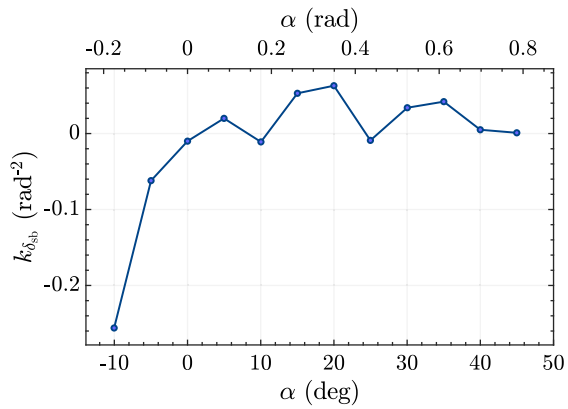
(a) Gain of lift coefficient due to the ground effect

(b) Basic lift coefficient contribution, as a function of angle of attack and elevator deflection

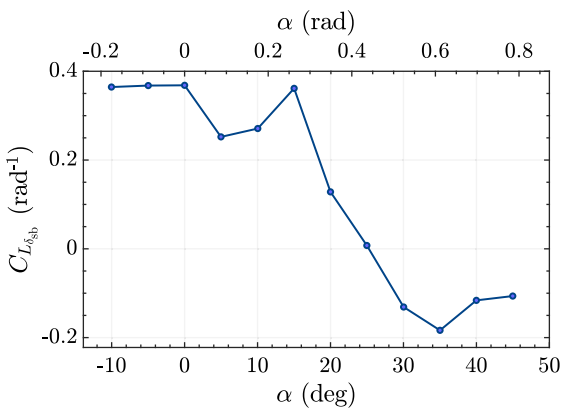
Fig. 31 Mapped lift coefficient terms of General Dynamic F-16 JSBSim model



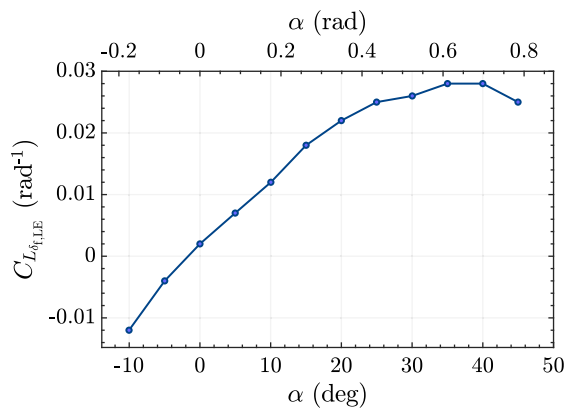
(a) Lift coefficient gradient w.r.t. the nondimensional pitch rate $qc/(2V)$, as a function of the angle of attack



(b) Change $k_{\delta_{sb}}$ of lift coefficient gradient w.r.t. the nondimensional pitch rate $qc/(2V)$, as a function of the angle of attack



(c) Lift coefficient gradient w.r.t. speed brake deflection, as a function of the angle of attack



(d) Lift coefficient gradient w.r.t leading edge flap deflection, as a function of the angle of attack

Fig. 32 Mapped lift coefficient terms of General Dynamic F-16 JSBSim model

$$
\begin{aligned}
&+\left[C_{\mathcal{N}_p}(\alpha)\, p + C_{\mathcal{N}_r}(\alpha)\, r\right]\frac{b}{2V}\\
&+\left[C_{\mathcal{N}_{\delta_a}}(\alpha,\beta) + \Delta C_{\mathcal{N}_{\delta_a}}(M)\right]\delta_a\\
&+\left[C_{\mathcal{N}_{\delta_r}}(\alpha,\beta) + \Delta C_{\mathcal{N}_{\delta_r}}(M)\,\alpha\right]\delta_r
\end{aligned}
\tag{A6}
$$

For a complete description of the F-16 aerodynamics model, the reader can refer to the official JSBSim repository https://github.com/JSBSim-Team/jsbsim exploring the folder `aircraft/f16` and the `<aerodynamics/>` block in the configuration file `aircraft/f16/f16.xml`.

Appendix B: Thrust model of the General Dynamics F-16 Fighting Falcon

With reference to definitions (10), (13) and (15), the instantaneous thrust is modelled in JSBSim [30,39] as follows:

$$T = \delta_T\, T_{\max,\mathrm{SL}}\, \tilde{T}_{\max}\big(h_{\mathrm{DA}}, M\big) \tag{B7}$$

where the normalized thrust $\tilde{T}_{\max}$ is a function of density altitude h_{DA} and flight Mach number M (see also Table 7). The density altitude h_{DA} is one of the quanti-

Table 7 Pratt and Whitney F100-PW-229 JSBSim model property

Property	Symbol	Value
Maximum static thrust at sea level	$T_{\max,\mathrm{SL}}$	17,800.0 lbf (79,178.3 N)
Maximum static thrust, with afterburner, at sea level	$T_{\max,\mathrm{ab,SL}}$	29,000.0 lbf (128,998 N)
Bypass ratio	BPR	0.4
Thrust-specific fuel consumption at cruise	TSFC	$0.74\,\frac{\mathrm{lb}}{\mathrm{lbf\,h}}\left(0.075\,\frac{\mathrm{kg}}{\mathrm{N\,h}}\right)$
Thrust-specific fuel consumption at cruise, with afterburner	ATSFC	$2.05\,\frac{\mathrm{lb}}{\mathrm{lbf\,h}}\left(0.209\,\frac{\mathrm{kg}}{\mathrm{N\,h}}\right)$
Angle of incidence of the thrust axis	μ_T	0.0 deg
Thrust line eccentricity	e_T	0.0 m

ties defined by the ICAO International Standard Atmosphere (ISA) model as functions of the altitude h (m) above the mean sea level (MSL) [31].

For known values of altitude h and airspeed V, the density altitude h_{DA} and the flight Mach number $M = V/a(h)$ allow the calculation of a normalized maximum available thrust in various engine conditions by interpolating the data points shown in Fig. 33.

For a complete description of the F-16 propulsive model, the reader can refer to the official JSBSim repository https://github.com/JSBSim-Team/jsbsim exploring the folder `aircraft/f16`, the `<propulsion/>` block in the configuration file `aircraft/f16/f16.xml`, and the engine configuration file `engine/F100-PW-229.xml`.

Appendix C: Flight control system of the General Dynamics F-16 Fighting Falcon (an excerpt)

The FCS within the JSBSim FDM [30,39] of the F-16 implements several ideal/parallel PID controllers acting on different input channels (see Fig. 5). Each controller attempts to minimize a given error function $e(t)$ over time by adjusting a control variable $u(t)$ to a new value determined by the following weighted sum:

$$u(t) = K_{\mathrm{p}}\, e(t) + K_{\mathrm{i}} \int_0^t e(\tau)\,\mathrm{d}\tau + K_{\mathrm{d}} \frac{\mathrm{d}e(t)}{\mathrm{d}t} \tag{C8}$$

For the roll channel, the FCS features a PID controller that minimizes the following nondimensional error function:

$$e_{\mathrm{roll}}(t) = \tilde{\delta}_{\mathrm{a}}(t) - \hat{p}(t) \tag{C9}$$

where $\tilde{\delta}_{\mathrm{a}}$ is the aileron normalized command input (in the interval $[-1, 1]$), and $\hat{p} = G_p\, p$ is a nondimensional roll rate, with G_p a constant gain (in s/rad). The error (C9) is a difference between the commanded roll rate and the actual roll rate, which is named `fcs/roll-trim-error` in JSBSim input/configuration meta-language and implemented by the following XML fragment:

Listing 1 F-16 roll channel FCS logic, PID error function.

```
<pure_gain name="fcs/roll-rate-norm">
    → p̂(t) := ...
   <input>velocities/p-aero-rad_sec</input>
        → p(t)
   <gain>0.31821</gain> → G_p (s/rad)
</pure_gain>

<summer name="fcs/roll-trim-error">
    → e_roll(t) := ...
   <input>fcs/aileron-cmd-norm</input> → δ̃_a(t)
   <input>-fcs/roll-rate-norm</input> → −p̂(t)
</summer>
```

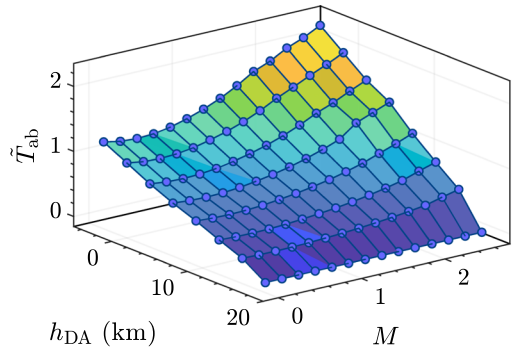
This roll channel controller is named `fcs/roll-rate-pid` in Listing 2 and becomes active in flight whenever the calibrated airspeed $V_{\mathrm{CAS}} \geq 20$ kts (`fcs/aileron-pid-trigger` set to 1).

Listing 2 F-16 roll channel FCS logic, as defined in JSBSim input/configuration meta-language in XML format.

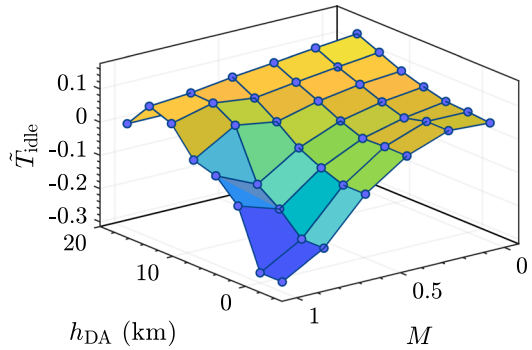
```
<switch name="fcs/aileron-pid-trigger">
   <default value="1"/>
   <test value="0">
       velocities/vc-kts lt 20.0
   </test>
</switch>

<pid name="fcs/roll-rate-pid"> → u_roll(t) := ...
   <trigger>fcs/aileron-pid-trigger</trigger>
       0 or 1
   <input>fcs/roll-trim-error</input>
   <kp>  3.00000 </kp> → K_p
   <ki>  0.00050 </ki> → K_i
   <kd> -0.00125 </kd> → K_d
</pid>
```

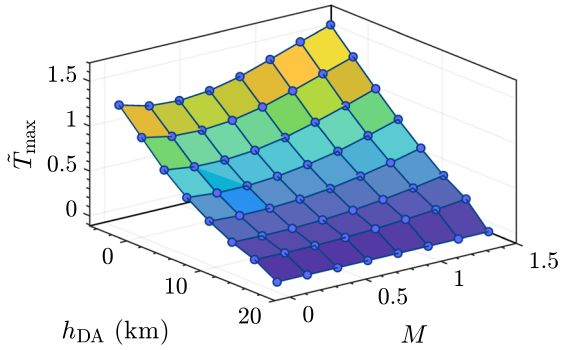
Finally, for the roll channel, the closed-loop input command $\tilde{\delta}'_{\mathrm{a}}$ is defined and then converted into an aerosurface deflection angle $\delta_{\mathrm{a}}(t) \in [\delta_{\mathrm{a,min}}, \delta_{\mathrm{a,max}}]$, named



(a) Normalized augmented thrust, unusing afterburner, as a function of the Mach number and the Density-Altitude



(b) Normalized idle thrust as a function of the Mach number and the Density-Altitude



(c) Normalized maximum thrust as a function of the Mach number and the Density-Altitude

Fig. 33 Mapped normalized thrust of General Dynamic F-16 JSBSim model

`fcs/aileron-pos-rad` as shown by the following XML input fragment:

Listing 3 F-16 roll channel FCS logic, as defined in JSBSim input/configuration meta-language in XML format.

```
<summer name="fcs/roll-rate-command">
    → δ̃'_a := δ̃_a + u_roll
  <input>fcs/roll-rate-pid</input>
  <input>fcs/aileron-cmd-norm</input>
  <clipto>
      <min>-1</min>
      <max>1</max>
  </clipto>
</summer>

<aerosurface_scale
    name="fcs/aileron-control">
  <input>fcs/roll-rate-command</input>
      → δ̃'_a ∈ [−1, 1]
  <range>
      <min>-0.375</min>  → δ_a,min
      <max> 0.375</max>  → δ_a,max
  </range>
  <output>fcs/aileron-pos-rad</output>
      → δ_a(t) ∈ [δ_a,min, δ_a,max]
</aerosurface_scale>
```

The FCS control laws for the remaining input channels are defined in a similar manner. For the pitch channel, the FCS control logic features a cascade of interconnected blocks. The high-level control logic is given by a PID controller assuming the following error function:

$$e_{\mathrm{pitch}}(t) = \hat{q}_0 - G_{\delta_{\mathrm{e}}}(t) - \hat{q}(t) - n_{z_{\mathrm{B}}}(t) \tag{C10}$$

that is, the difference between a commanded normalized pitch rate $\hat{q}_0$, a nondimensional elevator scheduler gain $G_{\delta_{\mathrm{e}}}$ (as a function of the instantaneous angle of

attack α), the instantaneous aircraft normalized pitch rate $\hat{q}(t)$, and the normal load factor $n_{z_B} = (g_{z_B} - a_{z_B})/g$, also known as g-load. The pitch channel PID controller is active whenever $V_{CAS} \geq 5$ kts.

A low-level FCS logic for the pitch channel defines a limiter on the elevator command. The F-16, in fact, has a maximum g-load allowable limit of $n_{z_B,\max} = 9$, and a minimum limit of $n_{z_B,\min} = -4$. Moreover, the pitch control logic defines a particular behaviour of the elevator according to the instantaneous value of the angle of attack $\alpha(t)$. If α approaches 30 deg, the FCS will command full down elevator deflection, overriding the pilot's commanded deflection, to prevent stalling. This articulated behaviour determines the elevator scheduler gain $G_{\delta_e}(t)$ in definition (C10).

For the yaw channel, the FCS features a PID controller that minimizes the following error function:

$$e_{\text{yaw}}(t) = \tilde{\delta}_r(t) + \hat{r}(t) + \frac{1}{4} n_{y_B}(t) \tag{C11}$$

where $\tilde{\delta}_r$ is the rudder normalized command input (in the interval $[-1, 1]$), $\hat{r} = G_r(t)\, r$ is a nondimensional yaw rate, $G_r(t)$ is a variable gain (in s/rad, and function of the instantaneous airspeed), and $n_{y_B} = (g_{y_B} - a_{y_B})/g$ is the lateral load factor. The PID controller is active whenever the aircraft calibrated $V_{CAS} \geq 10$ kts.

For the throttle input channel, the FCS simply maps the normalized throttle command $\tilde{\delta}_T \in [0, 1]$ into a signal $\delta_T \in [0, 2]$ assuming that when $\delta_T \geq 1$ the jet engine afterburner becomes active.

The speed brake channel is used by the FCS to prevent deep stall. This control commands speed brake deflections at high angles of attack and low speeds. This will provide just enough pitch-down moment to keep the aircraft under control. Moreover, speed brake maximum deflection is of $\delta_{sb,\max} = 60$ deg and is limited to 43 deg when the gear is extended to prevent physical speed brake damage on touchdown.

For a complete description of the F-16 flight dynamics model and its FCS, including control laws for the wing trailing-edge and leading-edge flap deflections, the reader can refer to the official JSBSim repository https://github.com/JSBSim-Team/jsbsim exploring the folder `aircraft/f16`, the `<flight_control/>` block in the configuration file `aircraft/f16/f16.xml` and the folder `systems`.

References

1. Stevens, B.L., Lewis, F.L.: Aircraft Control and Simulation. Wiley-Interscience, Hoboken (2003)
2. Dally, K., Kampen, E.-J.V.: Soft actor-critic deep reinforcement learning for fault tolerant flight control. In: AIAA SCITECH 2022 Forum. American Institute of Aeronautics and Astronautics, Reston, VA, USA (2022). https://doi.org/10.2514/6.2022-2078
3. Wang, H., Liu, S., Yang, X.: Adaptive neural control for non-strict-feedback nonlinear systems with input delay. Inf. Sci. **514**, 605–616 (2020)
4. Huo, X., Ma, L., Zhao, X., Niu, B., Zong, G.: Observer-based adaptive fuzzy tracking control of mimo switched nonlinear systems preceded by unknown backlash-like hysteresis. Inf. Sci. **490**, 369–386 (2019)
5. Xia, R., Chen, M., Wu, Q., Wang, Y.: Neural network based integral sliding mode optimal flight control of near space hypersonic vehicle. Neurocomputing **379**, 41–52 (2020)
6. Zhao, H.-W., Liang, Y.: Prescribed performance dynamic neural network control for a flexible hypersonic vehicle with unknown control directions. Adv. Mech. Eng. **11**(4), 1687814019841489 (2019)
7. Luo, C., Lei, H., Li, J., Zhou, C.: A new adaptive neural control scheme for hypersonic vehicle with actuators multiple constraints. Nonlinear Dyn. **100**(4), 3529–3553 (2020). https://doi.org/10.1007/s11071-020-05707-2
8. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. MIT Press, Cambridge (2018)
9. Reddy, G., Wong-Ng, J., Celani, A., Sejnowski, T.J., Vergassola, M.: Glider soaring via reinforcement learning in the field. Nature **562**(7726), 236–239 (2018)
10. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. In: Neural Information Processing Systems Deep Learning Workshop (2013). arXiv https://doi.org/10.48550/ARXIV.1312.5602
11. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: International Conference on Learning Representations (2015). arXiv https://doi.org/10.48550/ARXIV.1509.02971
12. Tsourdos, A., Dharma Permana, I.A., Budiarti, D.H., Shin, H.-S., Lee, C.-H.: Developing flight control policy using deep deterministic policy gradient. In: 2019 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES), pp. 1–7 (2019). https://doi.org/10.1109/ICARES.2019.8914343
13. Koch, W., Mancuso, R., West, R., Bestavros, A.: Reinforcement learning for UAV attitude control. ACM Trans. Cyber-Phys. Syst. **3**(2), 3301273 (2019). https://doi.org/10.1145/3301273
14. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms (2017). arXiv https://doi.org/10.48550/ARXIV.1707.06347
15. Bøhn, E., Coates, E.M., Moe, S., Johansen, T.A.: Deep reinforcement learning attitude control of fixed-wing UAVs using proximal policy optimization. In: 2019 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE,

Atlanta, GA, USA (2019). https://doi.org/10.1109/icuas.2019.8798254
16. Su, Z.-q, Zhou, M., Han, F.-f, Zhu, Y.-w, Song, D.-l, Guo, T.-t: Attitude control of underwater glider combined reinforcement learning with active disturbance rejection control. J. Mar. Sci. Technol. **24**(3), 686–704 (2019). https://doi.org/10.1007/s00773-018-0582-y
17. Mishra, A., Ghosh, S.: Variable gain gradient descent-based reinforcement learning for robust optimal tracking control of uncertain nonlinear system with input constraints. Nonlinear Dyn. **107**(3), 2195–2214 (2022). https://doi.org/10.1007/s11071-021-06908-z
18. Zhang, H., Huang, C.: Maneuver decision-making of deep learning for UCAV thorough azimuth angles. IEEE Access **8**, 12976–12987 (2020)
19. Lee, D., Kim, S., Suk, J.: Formation flight of unmanned aerial vehicles using track guidance. Aerosp. Sci. Technol. **76**, 412–420 (2018). https://doi.org/10.1016/j.ast.2018.01.026
20. Li, Y.-f, Shi, J.-p, Jiang, W., Zhang, W.-g, Lyu, Y.-x: Autonomous maneuver decision-making for a UCAV in short-range aerial combat based on an MS-DDQN algorithm. Def. Technol. **18**(9), 1697–1714 (2022)
21. Cambone, S.A., Krieg, K., Pace, P., Linton, W.: Unmanned aircraft systems roadmap 2005–2030. Off. Secr. Def. **8**, 4–15 (2005)
22. Wang, H., Liu, P.X., Bao, J., Xie, X.-J., Li, S.: Adaptive neural output-feedback decentralized control for large-scale nonlinear systems with stochastic disturbances. IEEE Trans. Neural Netw. Learn. Syst. **31**(3), 972–983 (2019)
23. Yuksek, B., Inalhan, G.: Reinforcement learning based closed-loop reference model adaptive flight control system design. Int. J. Adapt. Control Signal Process. **35**(3), 420–440 (2021). https://doi.org/10.1002/acs.3181
24. McGrew, J.S., How, J.P., Williams, B., Roy, N.: Air-combat strategy using approximate dynamic programming. J. Guid. Control. Dyn. **33**(5), 1641–1654 (2010). https://doi.org/10.2514/1.46815
25. Liu, X., Yin, Y., Su, Y., Ming, R.: A multi-UCAV cooperative decision-making method based on an MAPPO algorithm for beyond-visual-range air combat. Aerospace **9**(10), 563 (2022). https://doi.org/10.3390/aerospace9100563
26. Hu, D., Yang, R., Zuo, J., Zhang, Z., Wu, J., Wang, Y.: Application of deep reinforcement learning in maneuver planning of beyond-visual-range air combat. IEEE Access **9**, 32282–32297 (2021)
27. Wang, M., Wang, L., Yue, T., Liu, H.: Influence of unmanned combat aerial vehicle agility on short-range aerial combat effectiveness. Aerosp. Sci. Technol. **96**, 105534 (2020). https://doi.org/10.1016/j.ast.2019.105534
28. Yang, Q., Zhu, Y., Zhang, J., Qiao, S., Liu, J.: UAV air combat autonomous maneuver decision based on DDPG algorithm. In: 2019 IEEE 15th International Conference on Control and Automation (ICCA), pp. 37–42 (2019). IEEE
29. Shin, H., Lee, J., Kim, H., Hyunchul Shim, D.: An autonomous aerial combat framework for two-on-two engagements based on basic fighter maneuvers. Aerosp. Sci. Technol. **72**, 305–315 (2018). https://doi.org/10.1016/j.ast.2017.11.014
30. Berndt, J., De Marco, A.: Progress on and usage of the open source flight dynamics model software library, JSBSim. In: AIAA Modeling and Simulation Technologies Conference, 10–13 August 2009, Chicago, Illinois. American Institute of Aeronautics and Astronautics, Reston, VA, USA (2009). https://doi.org/10.2514/6.2009-5699
31. United States Committee on Extension to the Standard Atmosphere, National Aeronautics and Space Administration, National Oceanic and Atmospheric Administration, U.S. Air Force: U.S. Standard Atmosphere, 1976. NOAA-SIT 76-1562. National Oceanic and Amospheric Administration, Washington, DC, USA (1976)
32. Janota, A., Šimák, V., Nemec, D., Hrbček, J.: Improving the precision and speed of Euler angles computation from low-cost rotation sensor data. Sensors **15**(3), 7016–7039 (2015). https://doi.org/10.3390/s150307016
33. Brunton, S.L., Kutz, J.N.: Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control, 2nd edn. Cambridge University Press, Cambridge (2022). https://doi.org/10.1017/9781009089517
34. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
35. Peters, J., Schaal, S.: Reinforcement learning of motor skills with policy gradients. Neural Networks **21**(4), 682–697 (2008). https://doi.org/10.1016/j.neunet.2008.02.003. (**Robotics and Neuroscience**)
36. Hafner, R., Riedmiller, M.: Reinforcement learning in feedback control. Mach. Learn. **84**, 137–169 (2011). https://doi.org/10.1007/s10994-011-5235-x
37. Nicolosi, F., De Marco, A., Sabetta, V., Della Vecchia, P.: Roll performance assessment of a light aircraft: flight simulations and flight tests. Aerosp. Sci. Technol. **76**, 471–483 (2018). https://doi.org/10.1016/j.ast.2018.01.041
38. The Cosine-Haversine formula: American Mathematical Monthly **64**(1), 38 (1957). https://doi.org/10.2307/2309088
39. Snell, S., Enns, D., Garrard, W., Jr.: Nonlinear control of a supermaneuverable aircraft. J. Guid. Control. Dyn. **15**(4), 976–984 (1992). https://doi.org/10.2514/6.1989-3486