Event-based neural learning for quadrotor control

Esteban Carvalho, Pierre Susbielle, Nicolas Marchand, Ahmad Hably, Jilles Dibangoye

Event-based Neural Learning for Quadrotor Control

Estéban Carvalho[1,2*], Pierre Susbielle[1], Nicolas Marchand[1], Ahmad Hably[1] and Jilles S. Dibangoye[2]

[1*]Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, F-38000, Grenoble, France.
[2]Univ. Lyon, INSA Lyon, CITI, F-69621, Villeurbanne, France.

*Corresponding author(s). E-mail(s): esteban.carvalho.ec@gmail.com;
Contributing authors: pierre.susbielle@grenoble-inp.fr; nicolas.marchand@grenoble-inp.fr;
ahmad.hably@grenoble-inp.fr; jilles.dibangoye@inria.fr;

Abstract

The design of a simple and adaptive flight controller is a real challenge in aerial robotics. A simple flight controller often generates a poor flight tracking performance. Furthermore, adaptive algorithms might be costly in time and resources or deep learning based methods may cause instability problems, for instance in presence of disturbances. In this paper, we propose an event-based neural learning control strategy that combines the use of a standard cascaded flight controller enhanced by a deep neural network that learns the disturbances in order to improve the tracking performance. The strategy relies on two events: one allowing the improvement of tracking errors and the second to ensure closed-loop system stability. After a validation of the proposed strategy in a ROS/Gazebo simulation environment, its effectiveness is confirmed in real experiments in the presence of wind disturbance.

Keywords: Quadrotor, event-based control, trajectory tracking, deep neural network, online learning.

1 Introduction

Designing efficient and reactive autopilots for Unmanned Aerial Vehicles (UAVs) is still an active research trend. This is due to the increasing number of applications becoming imaginable due to the quadrotor's small size and flight abilities. To mention few of these various applications, we can cite: industrial surveillance (Silano et al, 2021), infrastructure inspections (Gu et al, 2020), cinematography (Torres-González et al, 2017), merchandise transport (Schneider, 2020) or aerial manipulation (Kim et al, 2013). Through these applications, quadrotors have proven their efficiency but are still very often supervised by an experienced pilot. This is because of the highly nonlinear dynamics of UAVs (Bangura and Mahony, 2012), as well as the various disturbances, such as wind gusts or the presence of suspended payload, that standard autopilot controllers can hardly handle. These challenges require the implementation of more efficient and more reliable autopilot algorithms than the existing ones. Advanced control techniques have been developed over the last years in order to push the capabilities of quadrotors to their limits. As a result, the following techniques have been employed: adaptive control (Dydek et al, 2013), Model Predictive Control (MPC) (Bangura and Mahony, 2014), event-based control (Durand et al, 2018), or the use of disturbance estimators along the lines of (Castillo et al, 2019; Wang and Shirinzadeh, 2015). Furthermore, to better improve controllers, we have seen the emergence of the

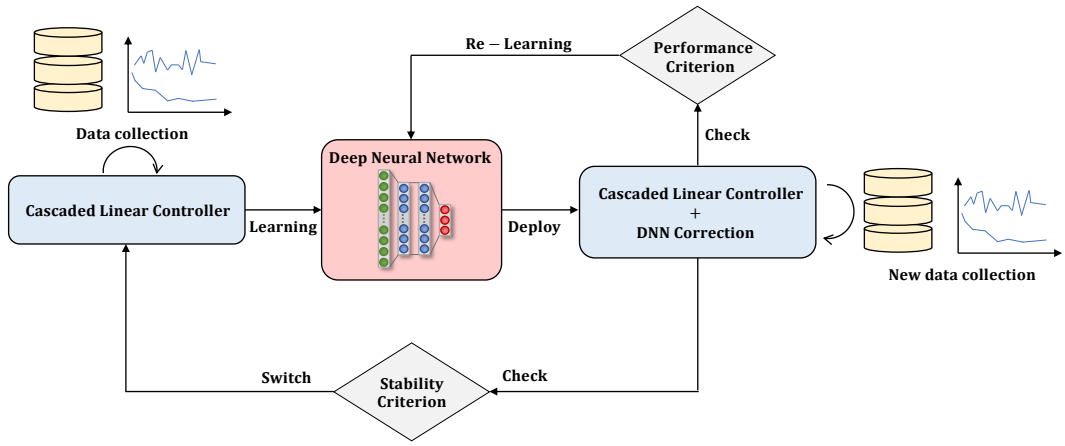

Fig. 1 Proposed event-based neural learning control strategy. At starting, an initial cascaded linear controller is used. It is enhanced with DNN learning after data collection. Two criteria are used to ensure stability and flight performance of the enhanced controller. If stability is faulty, the control is switch to the initial controller.

use of artificial intelligence (AI) in robotics (Anh et al, 2018). That is, data-driven methods mixed with control algorithms have been developed to perform such tasks and further improve flight tracking performance of quadrotors. Indeed, the use of AI is no longer limited to image and video processing and it is increasingly used for modeling and control purposes. In our previous work (Carvalho et al, 2022), we combined a Deep Neural Network (DNN) with a cascaded linear controller to correct the flight tracking errors, to fit the initially desired linear behavior and to overcome disturbances. In (Shi et al, 2019), the ground effect disturbance is entirely learned using DNNs and is then compensated with the control. In (Torrente et al, 2021), Gaussian processes are used to model high speed disturbances and are combined with MPC to achieve high precision tracking. Others approaches focus on highly trendy Reinforcement Learning (RL) to replace standard controllers. In particular, one can mention works of (Hwangbo et al, 2017; Lambert et al, 2019; Pi et al, 2020).

The problem with this type of AI-based approaches is that they require a huge amount of data to train a model. These methods may also experience a lot of trials and errors to efficiently learn a model. The data collection must be carefully operated so that it contains enough information to correctly learn the model. These exploration phases, sometimes done randomly (Sutton and Barto, 2018), can be critical for real-world applications including autonomous navigation in robotic systems. This is known as the exploration–exploitation dilemma. The exploration phase is very critical for a controller, as the quadrotor is not allowed to crash in real test. Thus, it is difficult to efficiently learn safe actions. Such approaches require the development of algorithms that safely explore the environment (Bura et al, 2021; Koller et al, 2019). In simulation, one needs first to explore a lot of possibilities and, then, to transfer the learning from simulation to experimentation (Zhao et al, 2020a). Such a task is not obvious and can become tedious when the model and the environment used in simulation differs from the reality. Finally, AI-based methods present the big issue of having the learning process, very often, done offline, due to the large amount of data required.

The emergence of data-driven approaches have consequently resulted in increasing-complex algorithms. As a result, proposed controllers became very specific to a type of application or work only for a specific quadrotors. They sometimes require a lot of tuning before achieving real flights. Thus, the test and implementation of new controllers, require a lot of time and effort before obtaining satisfactory results. For this reason, many industrial applications still use classical PID structures (Ang et al, 2005).

In order to solve these aforementioned problems we propose an event-based neural learning control strategy, which consists of a plug-and-play learning algorithm mixed with a standard cascaded controller. The latter is an improvement of our previously established work (Carvalho

et al, 2022). It is based on cascaded controller, here a PD for position control, which allows to rely on the cascaded architecture present in most open-source and industrial quadrotors. It allows a fast and easy initial tuning of the quadrotor flight controller. Once the traditional tuning step is done, the proposed controller automatically learns the corrections to be made to overcome the effects of model errors or disturbances. This approach avoids the tuning process of observers or other filter design to achieve good flight performance. Our approach is based on a succession of data collection, learning and correction phases. The correction is done using a deep neural network. Two criteria are implemented to ensure stability and tracking performance. The first one, the stability criterion, is based on Lyapunov stability theory and ensures that the DNN is not destabilizing the closed-loop system. The event related to the crossing of the threshold on this criterion, switches to the initial controller. The second one, the event based on a tracking performance criterion, proposes updates to the DNN to continuously improve the tracking. The overall scheme is a plug-and-play algorithm using standard cascaded controllers for quadrotors. The proposed event-based neural learning control methodology is summarized in Fig. 1. Note that coupling event-based strategies with neural learning has shown its efficiency (Zhao et al, 2020b).

Main contributions. In contrast to prior research, in this paper:

- We propose an event-based neural learning control strategy to overcome both residual dynamics and external disturbances,
- We guarantee both stability and tracking using two criteria in order to learn or re-learn a DNN.

The paper is organized as follows. The preliminaries, in section 2, introduces the quadrotor dynamics and the models used for control purpose. It also presents the initial cascaded control architecture at the basis of the proposed controller. Section 3 states the problem and objectives of this work. The proposed event-based neural learning control strategy is consequently presented in section 4. Section 5 presents the neural architecture used to model and estimate the error dynamics including residual dynamics and disturbances, used in the DNN-based controller. The validation of the proposed strategy is first performed via simulations and then tested in real experiments in section 6. Finally, a conclusion is given in section 7.

2 Preliminaries

In this section, a quadrotor nonlinear dynamical model is introduced. A first nonlinear model is proposed including unknown disturbances. Then, a linear model and the linear cascaded control used to pilot it is derived. Finally, a reformulation of the nonlinear model is introduced using the linear model. It will be used for the event-based neural learning control strategy.

2.1 Rigid-body dynamics

A quadrotor is made up of four rotors attached on a rigid cross-shaped frame. Motions in space are performed by differential control of each motor speed in a synchronized way.

Let $\{I\}$ be the inertial frame described by its unit vectors $\{\vec{i_1}, \vec{i_2}, \vec{i_3}\}$ and $\{B\}$ represents the frame attached to the rigid body, described by its unit vectors $\{\vec{b_1}, \vec{b_2}, \vec{b_3}\}$. The orientation of the quadrotor $\{B\}$ in $\{I\}$ can be described by a rotation matrix $\boldsymbol{R} \in SO(3)$. Using the Z-Y-X Euler formalism (see Fig. 2), the rotation matrix $\boldsymbol{R}$ is given by:

$$\begin{aligned} \boldsymbol{R} &:= \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\phi) \\ \boldsymbol{R} &:= \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \end{aligned} \quad (1)$$

where $c_\cdot := \cos(\cdot)$ and $s_\cdot := \sin(\cdot)$.

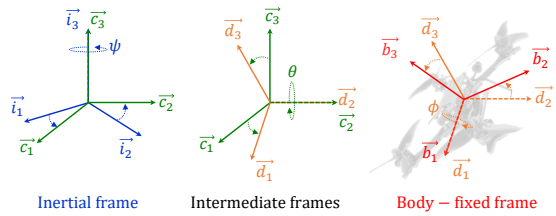

Fig. 2 Frames representation with Tait-Bryan angles. The inertial frame $\{I\}$ (left), intermediate frames (middle) and the body fixed frame $\{B\}$ (right).

The quadrotor center-of-mass position is denoted by $\xi := [x,\ y,\ z]^T \in \{I\}$. Its linear velocity, given in $\{I\}$, is $v := [v_x,\ v_y,\ v_z]^T$. Euler angles vector is $\zeta := [\phi,\ \theta,\ \psi]^T$, whose elements correspond to roll, pitch, and yaw angles, respectively. Vector $\Omega := [p,\ q,\ r]^T$ denotes the angular velocity expressed in the body frame $\{B\}$. $\Omega^\times$ is a skew-symmetric matrix associated to Ω verifying, for all $\boldsymbol{M}$ in $M_3(\mathbb{R})$, $\boldsymbol{M} \times \Omega = M\boldsymbol{\Omega}^\times$, where $\times$ denotes for the cross-product.

Let m be the total mass of the quadrotor and $\boldsymbol{J}$ be the inertia matrix expressed in the body fixed frame $\{B\}$. Thrust force value T is the total thrust generated by the four rotors and vector Γ represents the corresponding torques generated. The unmodeled and unknown dynamics of linear (translation) and angular (rotation) accelerations are, respectively, $\delta_t \in \mathbb{R}^3$ and $\delta_r \in \mathbb{R}^3$, they are usually called *residual dynamics*. These dynamics include: gyroscopic effects, battery discharge, ground effects, blade flapping, etc. As a result, the quadrotor rigid-body dynamics, governed by Newton-Euler equations (Mahony et al, 2012), are:

$$\begin{cases} \dot{\xi} = v, & \text{(2a)} \\ m\dot{v} = -mg\vec{i_3} + \boldsymbol{R}T\vec{b_3} + \delta_t, & \text{(2b)} \\ \dot{\boldsymbol{R}} = \boldsymbol{R}\boldsymbol{\Omega}^\times, & \text{(2c)} \\ \boldsymbol{J}\dot{\Omega} = -\boldsymbol{\Omega}^\times \boldsymbol{J}\Omega + \Gamma + \delta_r. & \text{(2d)} \end{cases}$$

For experimental purposes, we suppose that the angular rate controller is implemented and has a high convergence speed. As we are interested in conventional flight scenarios, acrobatic trajectories, such as doing barrel roll, are excluded. Thus, using Euler angles formalism is not a problem for the considered application. By making theses assumptions, one can consider the following system of equations:

$$\begin{cases} \dot{\xi} = v, & \text{(3a)} \\ m\dot{v} = -mg\vec{i_3} + \boldsymbol{R}T\vec{b_3} + \delta_t, & \text{(3b)} \\ \dot{\zeta} = \boldsymbol{W}\Omega, & \text{(3c)} \end{cases}$$

where $\boldsymbol{W}$ refers to the Wronskian matrix given by:

$$\boldsymbol{W} := \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi/c_\theta & c_\phi/c_\theta \end{bmatrix}. \quad (4)$$

Next, this model will be used to derive a linear model needed for control design.

2.2 Quadrotor linear model

The control strategy presented in this paper is based on a cascaded linear architecture. The linear model is obtained at hover conditions.

State and command vectors are respectively given by: $X := \left[\xi^T\ ,\ v^T\ ,\ \zeta^T\right]^T$ and $U := \left[T\ ,\ \Omega^T\right]^T$. Let $\mathbb{0}_3 := [0,\ 0,\ 0]$ and ξ_e be a given position point, the equilibrium state vector is denoted by $X_{eq} = \left[\xi_e^T\ ,\ \mathbb{0}_3\ ,\ \mathbb{0}_3\right]^T$ and the equilibrium command vector is $U_{eq} = \left[mg\ ,\ \mathbb{0}_3\right]^T$, which prevents the quadrotor from falling due to its weight.

Defining variation variables as $\tilde{X} := X - X_{eq}$ and $\tilde{U} := U - U_{eq}$ and assuming that δ_t is negligible, in the first instance, the following linear model is obtained using a first order Taylor expansion:

$$\begin{aligned} \dot{\tilde{X}} &= \begin{bmatrix} \boldsymbol{0_3} & \boldsymbol{I_3} & \boldsymbol{0_3} \\ \boldsymbol{0_3} & \boldsymbol{0_3} & \boldsymbol{A_{2,3}} \\ \boldsymbol{0_3} & \boldsymbol{0_3} & \boldsymbol{0_3} \end{bmatrix} \tilde{X} + \begin{bmatrix} \boldsymbol{0_{34}} \\ \boldsymbol{B_2} \\ \boldsymbol{B_3} \end{bmatrix} \tilde{U}, \\ \dot{\tilde{X}} &= \boldsymbol{A}\tilde{X} + \boldsymbol{B}\tilde{U}, \end{aligned} \quad (5)$$

where $\boldsymbol{0_3}$ refers to the square null matrix of order 3, $\boldsymbol{I_3}$ is the identity matrix of order 3, $\boldsymbol{0_{34}}$ is the null matrix composed of 3 lines and 4 columns, finally, $\boldsymbol{A_{2,3}}$, $\boldsymbol{B_2}$ and $\boldsymbol{B_3}$ are given by:

$$\boldsymbol{A_{2,3}} = \begin{bmatrix} 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{B_2} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ m^{-1} & 0 & 0 & 0 \end{bmatrix},$$
$$\boldsymbol{B_3} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The quadrotor linear model in eq. (5) is used to design a PD controller in next section 2.3. It will be enhanced with our event-based neural learning control strategy, presented in section 4.2.

2.3 Piloting the linear system

In this section, we describe the control architecture at the basis of the event-based neural learning control strategy. It is based on a linear cascaded architecture to achieve position, velocity and attitude control.

The proposed control architecture is composed of two linear cascaded controllers. The first one, defining the inner loop, is the attitude controller, which allows to stabilize the orientation of the quadrotor, ζ, to the desired angles. The second one, defining the outer loop, allows to reach the desired positions and speeds references, providing thrust and desired angles to the inner loop.

Attitude control:

To design the attitude control law, we use the linear model from (5): $\dot{\zeta} = \Omega$. Considering that Ω is the command input, we can directly choose it as:

$$\Omega = -\boldsymbol{K_\zeta}\left(\zeta - \zeta_{ref}\right), \tag{6}$$

where $\boldsymbol{K_\zeta}$ is the gain to obtain the desired closed-loop behavior and ζ_{ref} is the attitude target coming from the position and speed controller, explained in the sequel.

The gain $\boldsymbol{K_\zeta}$ is obtained through a linear quadratic regulation synthesis (LQR) which minimizes the quadratic cost expressed as follows:

$$\mathcal{J} := \int_0^{+\infty} (\zeta^T(t)\,\boldsymbol{Q_\zeta}\,\zeta(t) + \Omega^T(t)\,\boldsymbol{R_\zeta}\,\Omega)dt, \tag{7}$$

where $\boldsymbol{Q_\zeta}$ and $\boldsymbol{R_\zeta}$ are weighting matrices to be tuned to meet the desired tracking performance on attitude loop.

Position & velocity control:

The position and velocity control is denoted by $u_\zeta := [\tilde{\phi},\ \tilde{\theta},\ \tilde{T}]^T$. We can extract the position and velocity sub-model from (5) and write:

$$\begin{cases} \dot{\tilde{\xi}} = \tilde{v}, \\ \dot{\tilde{v}} = \begin{bmatrix} 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & m^{-1} \end{bmatrix} u_\zeta = \boldsymbol{B_\zeta} u_\zeta. \end{cases} \tag{8}$$

We now use the vector $\eta = \left[\eta_1^T,\ \eta_2^T\right]^T$ to define the errors : $\eta_1 := \tilde{\xi} - \tilde{\xi}_{ref}$ and $\eta_2 := \dot{\eta}_1 := \dot{\tilde{\xi}} - \dot{\tilde{\xi}}_{ref} = \tilde{v} - \tilde{v}_{ref}$, then (8) can be expressed as follows:

$$\dot{\eta} = \begin{bmatrix} \mathbf{0_3} & \boldsymbol{I_3} \\ \mathbf{0_3} & \mathbf{0_3} \end{bmatrix} \eta + \begin{bmatrix} \mathbf{0_3} \\ \boldsymbol{B_\zeta} \end{bmatrix} (u_\zeta - \boldsymbol{B_\zeta^{-1}} \dot{\tilde{v}}_{ref}). \tag{9}$$

For the following, we let $\overline{\boldsymbol{A}} = \left[\begin{smallmatrix} \mathbf{0_3} & \boldsymbol{I_3} \\ \mathbf{0_3} & \mathbf{0_3} \end{smallmatrix}\right]$ and $\overline{\boldsymbol{B}} = \left[\begin{smallmatrix} \mathbf{0_3} \\ \boldsymbol{B_\zeta} \end{smallmatrix}\right]$. Next, we introduce the following control law:

$$u_\zeta^* = -\boldsymbol{K_\eta}\eta + \boldsymbol{B_\zeta^{-1}} \dot{\tilde{v}}_{ref}, \tag{10}$$

where $\boldsymbol{K_\eta}$ is a gain to be tuned to get desired behavior in position and velocity tracking. This parameter design procedure is explained in section 4.2 and given by eq. (21).

2.4 Rewriting the nonlinear model

The model used for the event-based neural learning control strategy is a nonlinear model of the form given in eq. (11). It is obtained from (3) using the previously presented linear model in eq. (5). This model is subject to an additive error dynamic term, δ_t, which affects the dynamics. Such a term includes the residual dynamics, the linearized dynamics and external disturbances. The nonlinear dynamics of the quadrotor is given by:

$$\dot{\eta} = \overline{\boldsymbol{A}}\eta + \overline{\boldsymbol{B}}(u_\zeta - \boldsymbol{B_\zeta^{-1}} \dot{\tilde{v}}_{ref} + \boldsymbol{B_\zeta}^{-1}\delta_t). \tag{11}$$

It must be noted that now δ_t includes not only residual dynamics but also nonlinearities of the quadrotor system described in eq. (3).

3 Problem formulation

The aim of this work is to pilot the system given in eq. (11) under internal and external disturbances. For that, one proposes to rely on an initial cascaded controller which is easy to implement and tune using linear control tools, as presented in section 2.3. The ideal behavior given by η_{lin}, solution of $\dot{\eta} = \overline{\boldsymbol{A}} - \overline{\boldsymbol{B}}\boldsymbol{K_\eta}\eta$, corresponds to the desired tracking performance we want to obtain under disturbances. As the control is based on a linear controller, when testing it, it will not behave as intended due to the simplicity of the control. To automatically correct the trajectory tracking, we propose our event-based neural learning control strategy. It aims at minimizing the tracking error between:

$$\min_{u_\zeta} \left\|\eta_{lin} - \eta(u_\zeta)\right\|^2 \tag{12}$$

By minimizing (12), one reduces errors between the obtained behavior and the desired behavior. To achieve it, a deep neural network is used to

learn that error, δ_t. The controller is then updated using a DNN-based corrective term. This term aims at compensating δ_t in the command. It will be iteratively learned using the proposed strategy. In addition, as the closed-loop stability will depend on that neural term, the strategy will evaluate the stability in the Lyapunov sense and adapt the control if needed.

4 The proposed event-based neural control strategy

In this section, we describe the event-based neural learning control strategy used to track a reference trajectory under internal and external disturbances. It is based on a cascaded architecture to achieve position, velocity and attitude control. The linear controller at the basis of the strategy has been introduced in the preliminaries section 2.3. It is improved using an iteratively learned DNN corrective term to achieve satisfactory tracking. The DNN-based controller is presented in section 4.1. Afterwards, the event-based neural learning control strategy to ensure stability and tracking performance is exposed in 4.2. This strategy triggers a new learning when tracking errors become too large: the first event, defining the criterion named *performance criterion*. It switches controller if the DNN destabilizes the quadrotor. The second event is based on the value of a Lyapunov function, it is named *stability criterion*.

4.1 DNN-based controller

To derive the DNN-based controller, one relies on the nonlinear model given by eq. (11). This model is divided in two parts, a linear and a nonlinear one. The nonlinear part is ruled by δ_t term. One proposes to get an estimate $\hat{\delta}_t^k$ using a deep neural network. For the following sections, the superscript k denotes for the k-th learning of the neural network, meaning the k-th update of the DNN. Note that the $(k+1)$-th network has the same deep neural network structure as the k-th one, but it is re-learned using the newly created database. This means that weights and biases that compose the DNN are updated after each learning procedure, to get a better estimate. The learning strategy is given in section 4.2 and the DNN learning is explained in section 5. Let's assume now we have an estimate $\hat{\delta}_t^k$.

The proposed DNN-based controller is accordingly divided into two parts, namely:

$$u_\zeta = u_\zeta^* + \Delta u_\zeta^k, \tag{13}$$

where u_ζ^* is the linear controller in (10) and Δu_ζ^k is the feed-forward correction term based on the k-th learned neural network.

We can express the closed-loop of (11) using the chosen command (13):

$$\dot{\eta} = \underbrace{(\overline{\boldsymbol{A}} - \overline{\boldsymbol{B}}\boldsymbol{K_\eta})\eta}_{\substack{\text{Closed-loop linear} \\ \text{dynamics}}} + \underbrace{\overline{\boldsymbol{B}}(\Delta u_\zeta^k + \boldsymbol{B_\zeta^{-1}}\delta_t)}_{\text{Dynamic to cancel}}. \tag{14}$$

The closed-loop dynamics (14) is the sum of two terms. The first one describes the desired linear behavior. It is fully controlled using previously derived linear controller. The second one is a nonlinear term to be cancelled. If the term vanishes, the closed-loop dynamics tends towards the desired linear behavior.

The cancellation of the second term involves solving an optimization problem (15):

$$\min_{\Delta u_\zeta^k} \left\| \Delta u_\zeta^k + \boldsymbol{B_\zeta^{-1}}\delta_t \right\|^2. \tag{15}$$

However, the δ_t term is unknown, one can replace it by its estimate from the DNN:

$$\min_{\Delta u_\zeta^k} \left\| \Delta u_\zeta^k + \boldsymbol{B_\zeta^{-1}}\hat{\delta}_t^k(\mathrm{x}_t) \right\|^2. \tag{16}$$

The minimization problem (16) is a nonlinear problem since $\mathrm{x}_t = f(u_\zeta) = f(u_\zeta^* + \Delta u_\zeta^k)$. The input x_t depends on Δu_ζ^k which depends on $\hat{\delta}_t^k$. Solving such nonlinear, non convex, minimization problem can be very time consuming and may present several minima. The proposed solution consists in feeding the neural estimate with only the linear control input. It corresponds to replace u_ζ by u_ζ^* in the DNN input in eq. (16). Removing the correction term from the DNN input allows to directly deduce the solution:

$$\Delta u_\zeta^k = -\boldsymbol{B_\zeta^{-1}}\hat{\delta}_t^k(u_\zeta^*). \tag{17}$$

As $\boldsymbol{B_\zeta^{-1}}$ matrix is fully known, the correction to be applied to thrust T and ϕ_{ref} and θ_{ref} angles references from LQR can be quickly computed. The

cascaded architecture proposed is summarized in Fig. 3.

Here, the correction is only computed for position and velocity loop. The attitude loop corresponds to frame rotations and does not present disturbance to be estimated.

Now, as the control is completely defined, we will present the event-based strategy to learn or re-learn $\hat{\delta}_t^k$, according to stability and performance criteria.

4.2 The event-based neural learning strategy

The DNN-based control term given in eq. (13) depends on the estimate of $\hat{\delta}_t^k$. Hence, stability and performance of trajectory tracking rely on this estimation. This estimate will be refined frequently using small sets of collected data during the flight. As sets are small and do not necessarily contain enough data, for a good global approximation, an update of this term must be frequently performed using different criteria. As the DNN learning process is done automatically using a limited amount of data, the quadrotor cannot behave as expected. Two main events are expected, defining two criteria:

- Stability criterion: if the estimate $\hat{\delta}_t^k$ is not sufficiently accurate, stability might be compromised. One needs to first detect this critical situation using the stability criterion. Then, we propose to immediately switch to the initial linear controller, which does not involve the correction term, namely, $\Delta u_\zeta^k = \mathbb{0}_3^T$. After that, a new data collection procedure is done to estimate a new and more accurate DNN. The design of the stability criterion is explained in sub-section 4.2.1.

- Performance criterion: the predictions of the DNN cannot be as accurate as expected without causing any instability issues, for instance static errors. The proposed solution is to re-learn $\hat{\delta}_t^k$ getting a new estimate: $\hat{\delta}_t^{k+1}$ then one applies correction term $\Delta u_\zeta^{k+1} = -\boldsymbol{B_\zeta^{-1}}\hat{\delta}_t^{k+1}(u_\zeta^*)$. The performance criterion is explained in subsection4.2.2.

The scheme of our event-based neural learning control approach is also summarized in Fig. 3 and its methodology in Fig. 1. In the following, detailed explanations on each criterion, stability and tracking performance, are given.

4.2.1 Stability criterion

In order to establish stability-based switching criterion, it is needed to evaluate the closed-loop stability of the quadrotor. For this purpose, we have decided to use Lyapunov's stability definition. We consider the system given by eq. (11). Let the Lyapunov function be:

$$\mathcal{V} := \eta^T \boldsymbol{P} \eta, \tag{18}$$

where $\boldsymbol{P}$ is the solution of the Riccati equation (19):

$$\bar{\boldsymbol{A}}^T \boldsymbol{P} + \boldsymbol{P}\bar{\boldsymbol{A}} - 2\boldsymbol{P}\bar{\boldsymbol{B}}\bar{\boldsymbol{B}}^T\boldsymbol{P} = -\alpha \boldsymbol{P}, \tag{19}$$

or, equivalently by (20),

$$\left(\bar{\boldsymbol{A}} + \frac{\alpha}{2}\boldsymbol{I}\right)^T \boldsymbol{P} + \boldsymbol{P}\left(\bar{\boldsymbol{A}} + \frac{\alpha}{2}\boldsymbol{I}\right) - 2\boldsymbol{P}\bar{\boldsymbol{B}}\bar{\boldsymbol{B}}^T\boldsymbol{P} = 0. \tag{20}$$

Let $\boldsymbol{K_\eta}$ be given by:

$$\boldsymbol{K_\eta} := \bar{\boldsymbol{B}}^T \boldsymbol{P}, \tag{21}$$

and take the command as:

$$u_\zeta = u_\zeta^* + \Delta u_\zeta^k = -\boldsymbol{K_\eta}\eta + \boldsymbol{B_\zeta^{-1}}\dot{v}_{ref} + \Delta u_\zeta^k. \tag{22}$$

Substituting (22) in (11), we obtain:

$$\dot{\eta} = \overline{\boldsymbol{A}}\eta + \overline{\boldsymbol{B}}(-\boldsymbol{K_\eta}\eta + \Delta u_\zeta^k + \boldsymbol{B_\zeta}^{-1}\delta_t). \tag{23}$$

Using (17) and defining $\varepsilon_\eta := \delta_k - \hat{\delta}_t^k$, it follows from (23) that:

$$\dot{\eta} = (\overline{\boldsymbol{A}} - \overline{\boldsymbol{B}}\boldsymbol{K_\eta})\eta + \overline{\boldsymbol{B}}\boldsymbol{B_\zeta}^{-1}\varepsilon_\eta. \tag{24}$$

Hence, the time derivative of the Lyapunov function is then given by:

$$\begin{aligned}
\dot{\mathcal{V}}(t) &= \dot{\eta}^T\boldsymbol{P}\eta + \eta^T\boldsymbol{P}\dot{\eta}, &(25)\\
&= \eta^T\left((\bar{\boldsymbol{A}} - \bar{\boldsymbol{B}}\boldsymbol{K_\eta})^T\boldsymbol{P} + \boldsymbol{P}(\bar{\boldsymbol{A}} - \bar{\boldsymbol{B}}\boldsymbol{K_\eta})\right)\eta \\
&\quad + 2\eta^T P\bar{\boldsymbol{B}}\boldsymbol{B_\zeta^{-1}}\varepsilon_\eta, &(26)\\
\dot{\mathcal{V}}(t) &= -\alpha\mathcal{V} + 2\eta^T\boldsymbol{P}\bar{\boldsymbol{B}}\boldsymbol{B_\zeta^{-1}}\varepsilon_\eta. &(27)
\end{aligned}$$

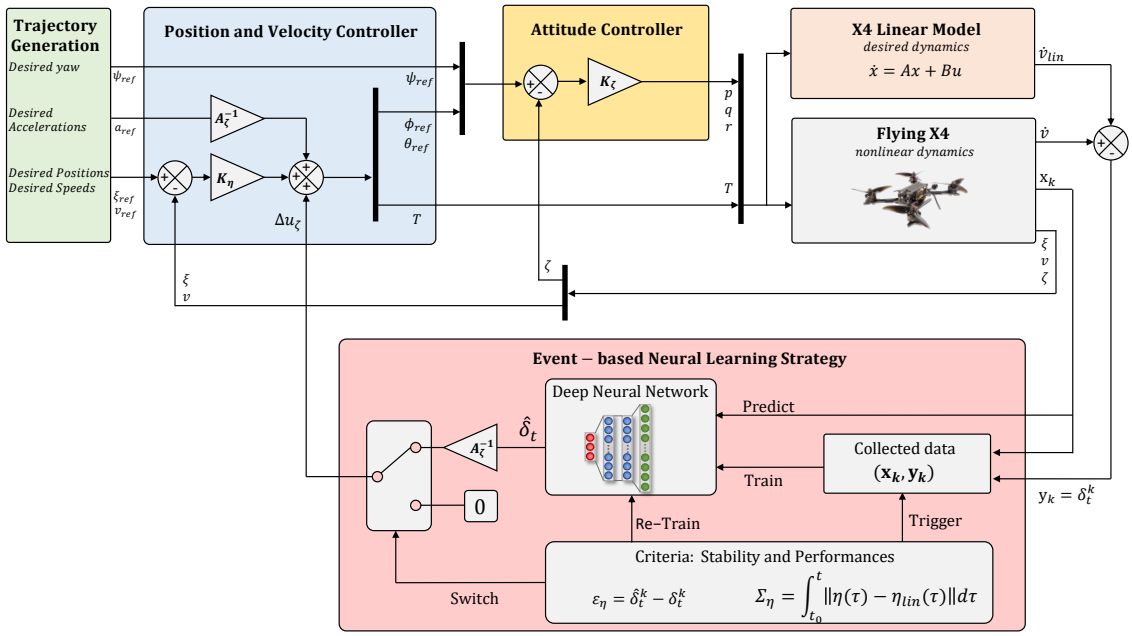

Fig. 3 The event-based neural learning control strategy.

Let $\boldsymbol{P}^{\frac{1}{2}}$ denote the square matrix such that $\boldsymbol{P}^{\frac{1}{2}}\boldsymbol{P}^{\frac{1}{2}} = \boldsymbol{P}$. Note that since the system is controllable, $\boldsymbol{P}$ is symmetric positive definite and, therefore, $\boldsymbol{P}^{\frac{1}{2}}$ always exists and can be chosen to be also symmetric positive definite. Let $w := \boldsymbol{P}^{\frac{1}{2}}\eta$. Then (27) becomes:

$$\dot{\mathcal{V}} = -\alpha w^T w + 2w^T \boldsymbol{P}^{\frac{1}{2}} \bar{\boldsymbol{B}} \boldsymbol{B}_{\boldsymbol{\zeta}}^{-1} \varepsilon_\eta, \tag{28}$$

$$\leq -\alpha \|w\|^2 + 2\lambda_{\max}(\boldsymbol{P}^{\frac{1}{2}} \bar{\boldsymbol{B}} \boldsymbol{B}_{\boldsymbol{\zeta}}^{-1}) \|w\| \|\varepsilon_\eta\|. \tag{29}$$

Therefore, as long as ε_η is sufficiently small, the Lyapunov function will be decreasing. More precisely, since $\|w\| = \sqrt{\mathcal{V}}$, if:

$$\|\varepsilon_\eta\| < \frac{\alpha}{2\lambda_{\max}(\boldsymbol{P}^{\frac{1}{2}} \bar{\boldsymbol{B}} \boldsymbol{B}_{\boldsymbol{\zeta}}^{-1})} \sqrt{\mathcal{V}}, \tag{30}$$

then $\mathcal{V}$ will be strictly decreasing. We can define a threshold variable as:

$$\varepsilon_m = \frac{\alpha}{2\lambda_{\max}(\boldsymbol{P}^{\frac{1}{2}} \bar{\boldsymbol{B}} \boldsymbol{B}_{\boldsymbol{\zeta}}^{-1})} \sqrt{\mathcal{V}}. \tag{31}$$

Then, if we get this matching condition:

$$\|\varepsilon_\eta\| - \varepsilon_m \geq 0, \tag{32}$$

the algorithm switches the controller to the initial one given in eq. (10) since the defined Lyapunov function (18) is not guaranteed to decrease anymore. In practice, switching controller is done by applying $\Delta u_\zeta^k = \mathbb{0}_3^T$. Then, a new data collection is performed with this linear controller. Once enough data is collected, a new DNN is learned and next applied using eq. (17). This event is based on the evolution of $\|\varepsilon_\eta\|$, which is computed thanks to the output of the DNN and the acceleration value obtained from the IMU. The value of ε_m is then restarted until the stability criterion is triggered again if the new DNN triggers again the criterion, meaning stability in the sense of Lyaupunov cannot be guaranteed.

4.2.2 Performance criterion

It is possible that some errors, generated by an inaccurate estimation of $\hat{\delta}_t^k$, are not affecting the quadrotor's closed-loop stability. A second criterion must be defined to handle this case. Indeed, the first criterion given in section 4.2.1 may not be triggered while the tracking performance is not satisfactory. For instance, in the presence of a static error that is not corrected by the neural network, the output of the DNN is constant and not triggering the stability threshold. Then, the quadrotor will continue to fly with a bad tracking performance. To overcome this problem, it is proposed to use a tracking performance criterion

based on the cumulative error to the linear model:

$$\Sigma_\eta(t) = \int_{t_s}^{t} \|\eta(\tau) - \eta_{lin}(\tau)\|^2 \, d\tau. \quad (33)$$

Then a new learning is triggered when:

$$\Sigma_\eta(t) > \Sigma_m, \quad (34)$$

where Σ_m is a threshold experimentally chosen so as to trigger a new learning when there is enough error to be corrected. This criterion is coupled with the fact that sufficient data must also be collected. To prevent *catastrophic forgetting* (Robins, 1995), meaning the complete forgetting of previously learned tasks, a small part of the previously build data base is kept and injected in the new one. That procedure is called *Rehearsal*. Moreover, the learning rate can be decreased using a multiplying factor at each new re-learning event.

In the next section, we present how δ_t is estimated using a deep neural network and we give some insights on its learning procedure.

5 Error dynamics learning

This section presents the proposed structure for the neural estimation of δ_t. It corresponds to an acceleration error term between the real quadrotor dynamics and the desired linear dynamics. It is given by eq. (5) and by isolating it we obtain:

$$\delta_t = \dot{v} - (\boldsymbol{A_{2,3}}\tilde{\zeta} + \boldsymbol{B_2}\tilde{U}). \quad (35)$$

Based on which, we propose to get an estimate, denoted by $\hat{\delta}_t$, using a deep neural network. They are powerful nonlinear estimators (Hornik et al, 1989). The universal approximation theorem and its extension to deep feedforward networks, guarantee that one can approximate any continuous functions using a DNN. Then, the error dynamics in (35) can be approximated using a DNN.

In this paper, we use a DNN to estimate the acceleration error term. This choice comes from the fact that, one wants to learn not only unmodeled dynamics but also disturbances, such as wind gusts for instance. As a result, a more complex network would better model all these dynamics. According to the nature of this estimation problem, here a regression of temporal data and the use of this kind of networks is well suited (Goodfellow et al, 2016).

The proposed DNN is composed of fully-connected layers (see Fig. 4). The rectified linear unit (ReLU) is used as activation function for all hidden layers and is denoted as h. It is defined element-wise by $h(\cdot) := max(\cdot, 0)$.

First, let's define weights and biases used to build the DNN: $\boldsymbol{\mathcal{A}_1} \in \mathbb{R}^{N_u, \#\mathrm{x}_t}$, $\boldsymbol{\mathcal{A}_2} \in \mathbb{R}^{N_u, N_u}$, $\boldsymbol{\mathcal{A}_3} \in \mathbb{R}^{N_u, 3}$ $\boldsymbol{\mathcal{B}_1} \in \mathbb{R}^{N_u}$, $\boldsymbol{\mathcal{B}_2} \in \mathbb{R}^{N_u}$ and $\boldsymbol{\mathcal{B}_3} \in \mathbb{R}^3$. Where N_u is the number of units per layer, $\#A$ denotes the cardinal of A and x_t denotes the neural network input.

Let x_t be the input vector of the DNN. The expression of the proposed deep neural network is then given by:

$$\hat{\delta}_t(\mathrm{x}_t) = \boldsymbol{\mathcal{A}_3}^T h(\boldsymbol{\mathcal{A}_2}^T h(\boldsymbol{\mathcal{A}_1}\mathrm{x}_t + \boldsymbol{\mathcal{B}_1}) + \boldsymbol{\mathcal{B}_2}) + \boldsymbol{\mathcal{B}_3}. \quad (36)$$

The number of layers is set to $N_l = 2$ and the number of units per layer is set to $N_u = 32$. A quick ablation study showed that it is enough to correctly model the error dynamics and to perform real-time computation.

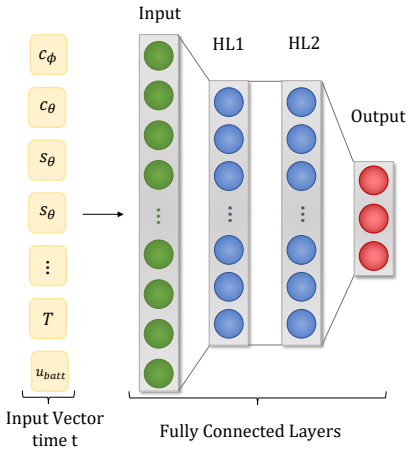

Fig. 4 Deep neural structure used to learn δ_t. Is is a feed-forward DNN composed of N_l hidden layers of N_u units, using ReLU as activation functions.

The choice of the input of the DNN is made considering eq. (2b): both sines and cosines of Euler angles are provided to the input as they appear in the rotation matrix $\boldsymbol{R}$. As the drag force is proportional to the square of velocities, one provides them to the network. The battery's voltage tension, u_{batt}, is provided to better learn the discharge profile. Finally, one provides the thrust, T,

as it is an important term in the linear acceleration dynamics. To summarize, the input vector for the proposed DNN at time t, x_t, is given by (37):

$$\mathrm{x}_t = \left[c_\phi,\ s_\phi,\ c_\theta,\ s_\theta,\ v_x^2,\ v_y^2,\ v_z^2,\ u_{batt},\ T\right]^T. \quad (37)$$

The Mean-Squared Error (MSE) is chosen as loss function $\mathcal{L}$ for training and validation. Here, the MSE is between δ_t^{mes}, computed using measured values of accelerations and expected linear accelerations, and predictions $\hat{\delta}_t$. Let's define ϑ as the optimization variable composed of all weights and biases of the DNN, given by (38):

$$\vartheta := \{\boldsymbol{\mathcal{A}_1} \quad \boldsymbol{\mathcal{A}_2} \quad \boldsymbol{\mathcal{A}_3} \quad \boldsymbol{\mathcal{B}_1} \quad \boldsymbol{\mathcal{B}_2} \quad \boldsymbol{\mathcal{B}_3}\}. \quad (38)$$

The network's training will try to find ϑ that minimizes $\mathcal{L}$:

$$\min_{\vartheta} \mathcal{L}(\vartheta) = \min_{\vartheta} \frac{1}{N} \sum_{k=1}^{N} \|\delta_t^{mes} - \hat{\delta}_t\|^2. \quad (39)$$

The solver used to solve (39) is the Nesterov-Accelerated Adaptive Moment Algorithm, known as NADAM algorithm. It is an improved version of the stochastic gradient descent algorithm.

All the network parameters and hyper-parameters used for training are given in the following Table 1. They are experimentally and empirically chosen to achieve fast and efficient convergence of the DNN.

Table 1 Parameters of the proposed deep neural network and training values.

Parameter	Value
Number of inputs	9
Number of hidden layers n_l	2
Number of units n_u	32
Batch	128
Epoch	250
Learning rate l_r	0.005
μ	0.9
ν	0.9999

The learning is done on several small databases created online during flights using the proposed event-based learning strategy given in section 4. Each training set is covering at least $t_c = 30s$ of collected data $(\mathbf{x_k}, \mathbf{y_k})$, .

In the next section, we assess the effectiveness of the event-based neural learning control strategy. It is both tested in simulation environment and in real flights tests.

6 Simulation and experimental results

In this section, the proposed strategy is first validated in a simulation environment in section 6.1 and it is then tested in experimental flights in section 6.3 using setup described in section 6.2.

For both simulations and experiments, we have worked with a ROS environment. Data collected from different sensors are stored in rosbag and used for the learning procedure. Main parameters for simulation and experimental flights are provided in Table 2.

Table 2 Parameters and their values for simulation and experimentation.

Parameter	Value
$g(m/s^-2)$	9.81
$m(kg)$	0.55
$\boldsymbol{K_\zeta}$	$\begin{bmatrix} 4.47 & 0 & 0 \\ 0 & 4.47 & 0 \\ 0 & 0 & 4.47 \end{bmatrix}$
$\boldsymbol{K_\eta}$	$\begin{bmatrix} 0.14 & 0 & 0 & 0.24 & 0 & 0 \\ 0 & -0.14 & 0 & 0 & -0.14 & 0 \\ 0 & 0 & 2.5 & 0 & 0 & 2 \end{bmatrix}$
$\Sigma_m(simu)$	1.0
$\Sigma_m(exp)$	0.7
$t_c(s)$	30

6.1 Simulations results

Simulations are performed using ROS and Gazebo environment (Koenig and Howard, 2004). To validate the proposed strategy, two independent scenarios are proposed:

1. A first scenario (I) consists in a repetition of circles in x/y plane,
2. A second one (II) requests the quadrotor to first perform circles in x/y plane and next to perform steps in y/z plane.

For both tests, no external disturbances are added to the simulation. Here, presented errors are

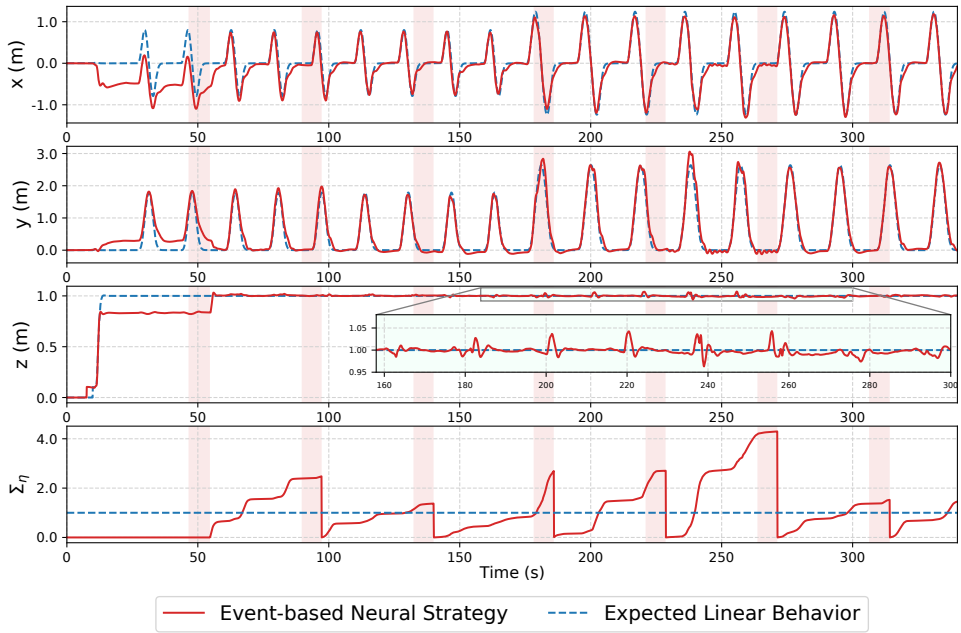

Fig. 5 Simulation scenario (I): circles motions. This scenario illustrate the performance criterion. Red areas represent learning and re-learning process.

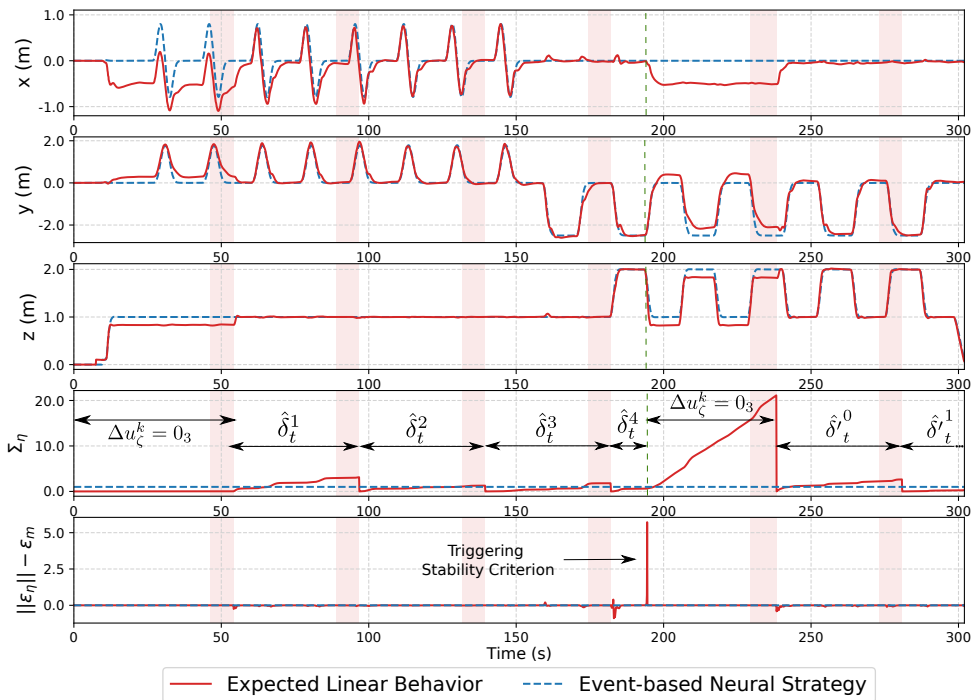

Fig. 6 Simulation scenario (II): circles followed by steps motions. This scenario illustrate stability and performance criteria. Red areas represent learning and re-learning process.

mainly due to the simplicity of the initial linear control (10), that does not contain an integral action. Through the scenario, the quadrotor will learn to correct its behavior via the proposed strategy.

6.1.1 Performance criterion analysis: scenario (I)

Simulations results of scenario (I), consisting in circles motions, are presented in Fig. 5. The first three graphs represent the three spatial motions. The fourth graph is representing the cumulative error (33). It is reset after each re-learning.

At the beginning of the scenario, one notices major static errors in all three components (x, y and z). A first data collection and a first DNN is proposed after few seconds of flight. It corrects the behavior and get closer to the linear desired behavior in blue dashed dot. It gets closer but can still be improved because the deep neural network has not yet fully learned the dynamics to be corrected. The performance criterion is here increasing directly after applying the first DNN. After a second data collection and when the performance threshold has been reached, a re-learning procedure is triggered. A second DNN, is then applied around 90s, and decreases the tracking errors in all components. At $t = 180s$, the trajectory generation asks the quadrotor to perform larger circles, as these references were not part of the training scenario, tracking performance is not as good as previously learned circles. Still, main static errors of initial controller are corrected. The learning procedure is triggered once more and improves the DNN proposed. At the end of the scenario, one notices that the behavior is significantly closer to the expected linear behavior.

6.1.2 Stability criterion analysis: scenario (II)

The results of the second scenario (II) are shown in Fig. 6 and illustrate the stability criterion. Here, we restarted the procedure from scratch, where only the initial linear cascaded controller is used at starting. For practical reasons, the criterion presented in (32) is implemented with a time-triggering threshold. Indeed, in order to avoid possible unwanted and unnecessary switching due to noise, the controller switch is activated if the stability criterion is activated during 10 samples. The first three curves represent the displacement on x, y and z. The second one is the cumulative error (33). The last curve represents the stability criterion (32).

This scenario is composed of eight circles motions in x/y plane followed by six steps in y/z plane after 160s of flight. As for previous test case scenario, the quadrotor starts with large static errors that are next corrected using successive improvement of $\hat{\delta}_t^1$ and the previously studied performance criterion. The DNN is re-learned up to $\hat{\delta}_t^4$. After 160s, the quadrotor is asked to perform successions of steps, a case that is not part of the training of the neural controller. It starts to well behave, however, at t=180s, the stability criterion (32) is triggered. It means that the last proposed DNN, $\hat{\delta}_t^4$, is destabilising the quadrotor system for the scenario. The initial linear cascaded controller is then next used to maintain stability and collect new useful data to learn a new DNN. After that training, the new proposed network, here denoted as $\hat{\delta'}_t^0$, is correcting the dynamics to fit the linear behavior correctly without destabilisation in step motions.

The choice to switch to the initial controller instead of the previously learned network is motivated by the fact that when testing that other solution, the proposed strategy was successively switching controller till the initial controller. Indeed, if the lastly learned network is not able to well behave, there is a high possibility that the previous one did not correctly learned the information.

Now, as we validated the proposed strategy in simulation, one can deploy it in real flight test. Before performing test, a description of the experimental aerial platform is presented in the following section.

6.2 Experimental setup

To experimentally validate our event-based neural learning control strategy we used a *Holybro Kopis 2* quadrotor. It is built with two on-board microcontrollers:

- One *Kakute F7* with PX4 firmware (PX4, 2021). It is used as low-level controller (PID for angular speed control) and it also performs state estimation using an Extended Kalman Filter, ECL EKF2.

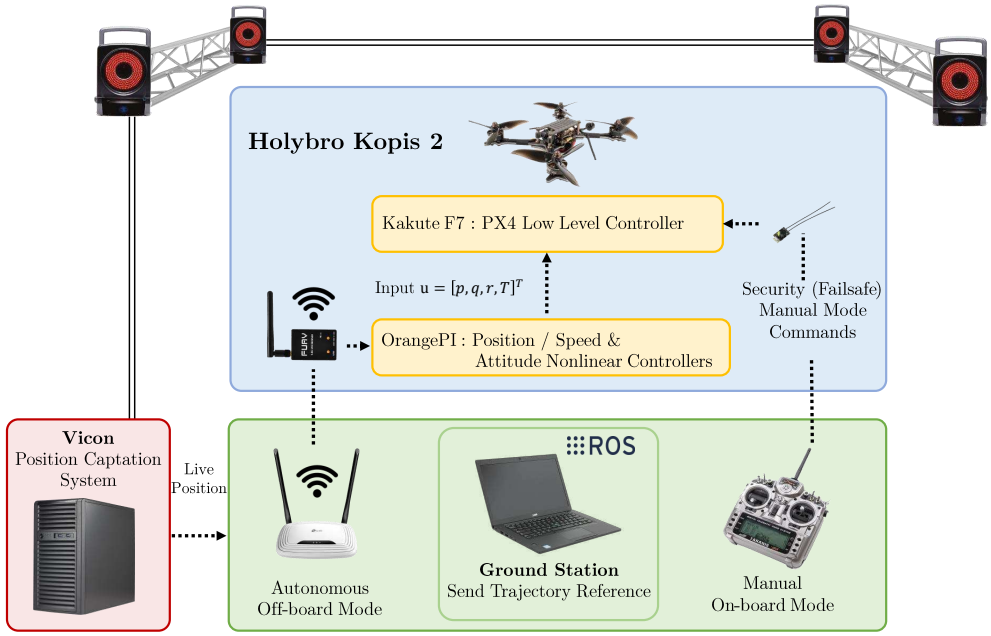

Fig. 7 The experimental motion capture room used for experimental tests of the event-based neural learning controller.

- One *Orange-pi*, it is used to perform our event-based neural learning controller. It publishes computed commands (13) to the Kakute F7. We use TensorFlow Lite for fast on-board neural predictions.

The quadrotor is connected to the ground station computer using a WiFi bridge, communication is done at 100Hz. References trajectories are generated from the ground station and are sent to the controller via WiFi. A motion capture system is used to obtain spatial positions and Euler angles of the quadrotor. The global setup is illustrated in Fig. 7. The experimental test and results are explained in following section.

6.3 Experimental results

To better assess the contribution of the proposed event-based neural strategy, we tested its tracking performance under external disturbances, in a real test case. Here, experimental tests are performed in a windy environment, using a wind indoor tunnel, see Fig. 8. The quadrotor is asked to face a sinusoidal wind doing back and forth motions. Results are compared to a well tuned PID controller coupled with a disturbance observer. That controller was previously tuned in order to fit the desired expected linear closed-loop behavior given by (9) with the PD controller without learning (13) in windy conditions. The experimental results are provided in Fig. 9. A video of the experimental results is available at: youtu.be/S4I2dlPDCJY

The last curve is representing the intensity of the wind applied in m/s. It is an approximate value calculated from the maximum speed of the wind generated by the wind indoor tunnel and the transmitted value in PWM. Applied disturbance is a sinus whose frequency is around 0.046Hz starting at $t = 10s$. It is important to notice that the wind frequency is not synchronized with the back and forth motion of the quadrotor.

When the wind starts, in part n°0 in the figure, the proposed controller is not able to reject the disturbance, generating large errors compared to the PID cascaded controller along with the disturbance observer. After a collection time and a first learning, the first DNN mainly corrects the wind disturbance effect, in portion n°1. In part n°2, the quadrotor has re-learned the deep neural network but has difficulty in z component compared to the other controller. Indeed the battery discharge is not yet completely learned. A change of amplitude

Fig. 8 The quadrotor facing the wind indoor tunnel.

and offset of the intensity of the wind is operated at $t = 145s$, at the end of part n°2. Here, notice that the proposed controller with the previously learned controller has difficulty to reach its desired target point, as the intensity of the wind is stronger and was not previously learned. After a new collection and re-learning process, it is able to get closer to the linear desired behavior in parts n°4 and n°5.

It is important to notice that when applying the same DNN, the behavior between each steps is not always exactly the same. Indeed conditions are varying at each step: the de-synchronized wind, the increased battery discharge, the wind disturbance generated in the whole room, etc. Thus, the output of the network is not exactly the same each time and the correction leads to a slightly different behavior at each new step. Root mean squared errors (RMSE) of all parts [0 up to 5] presented in the figure are computed for both controllers compared to the linear excepted behavior with PD controller in perfect simulated non windy case. RMSE are provided in Table 3. A global improvement of RMSE through flights, thus of flight tracking, can be observed with the proposed event-based neural learning control strategy. One notices that the behavior in z component of the PID along with then disturbance observer is globally better than the proposed solution. Indeed, the event-based neural strategy does not include integral action and is progressively learning the battery discharge profile. It needs more time to better learn the correction. However, at the end of the scenario, it is improved as more data as been used and introduced in the last DNN. Also, the PID controller with observer produces more low frequency oscillations during whole scenario in each component. Finally, one may notice that the stability criterion is very noisy, but via the filtered procedure, it is not triggered during the proposed scenario.

Table 3 RMSE (in meters) in each components during the proposed wind scenario for each controller.

	Event-based Neural strategy			PID+Observer		
Part	x	y	z	x	y	z
0	0.054	0.323	0.02	0.023	0.046	0.072
1	0.031	0.098	0.021	0.015	0.061	0.006
2	0.012	0.065	0.053	0.013	0.058	0.006
3	0.011	0.114	0.018	0.012	0.06	0.007
4	0.011	0.059	0.027	0.012	0.064	0.007
5	0.013	0.059	0.015	0.016	0.064	0.064

7 Conclusion

This paper presents an event-based neural learning control strategy for quadrotor trajectory tracking in different scenarios including internal and external disturbances. The objective of this controller is twofold: to allow an ease of implementation (PD gains) using well-spread cascaded control architecture and to provide a great capacity of adaptability to correct flight errors whether they are internal (model, unknown dynamics, etc.) or external (wind disturbances, etc.) while keeping the system stable. The proposed controller is based on a succession of data collections, learning procedures and correction phases. It allows an online application as it does not require huge computing time and it is fast enough to avoid landing for learning. It is based upon two criteria. The first one is used to re-learn the deep neural network, to adjust the correction proposed to the initially tuned linear cascaded controller, in order to improve flight performance, and get closer to the initial desired linear behavior. The second criterion, that is concerning the stability, is implemented to overcome the shortcoming of deep neural networks: the coherence of the output value in the case of unknown data. Simulation and experiments validate the benefits of such approach.

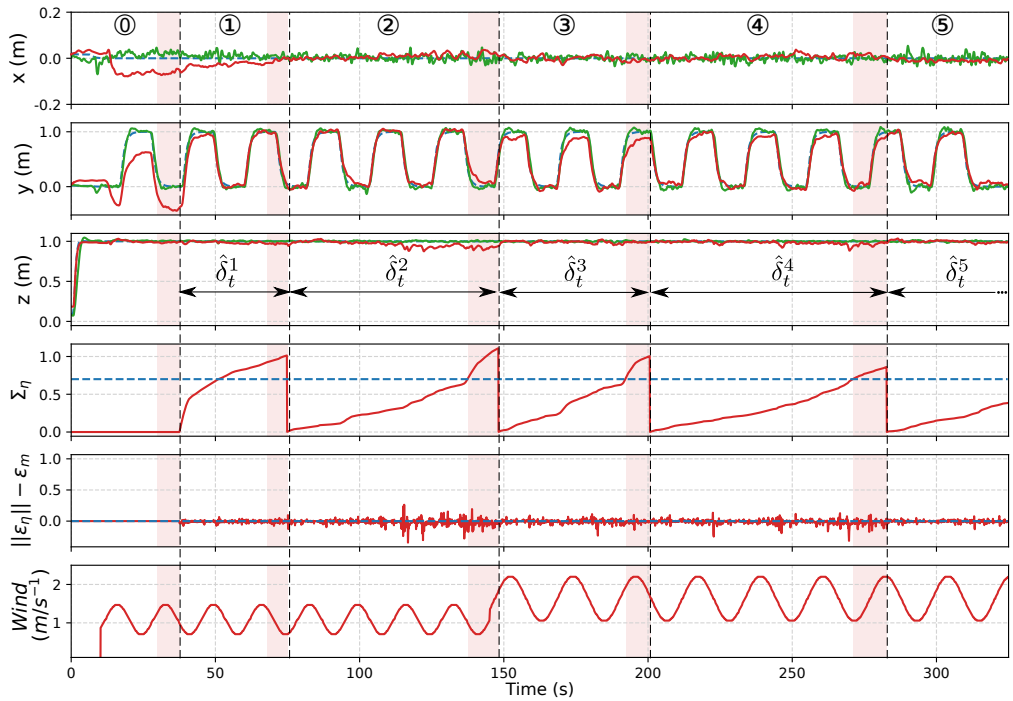

Fig. 9 Experimental test: back and forth motions in front of a sinusoidal varying wind. Red areas represent learning and re-learning process. For x, y and z, the red curve is the result with the event-based neural strategy. The green curve is the obtained result with the PID along with disturbance observer. The blue one is the expected linear behavior with the linear PD controller only.

Acknowledgments. This version of the article has been accepted for publication, after peer review but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://doi.org/10.1007/s10514-023-10115-7. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms.

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) and ROBOTEX 2.0 (Grants ROBOTEX ANR-10-EQPX-44-01 and TIRREX ANR-21-ESRE-0015) funded by the French program Investissements d'Avenir.

References

Ang KH, Chong G, Li Y (2005) PID control system analysis, design, and technology. IEEE Transactions on Control Systems Technology 13(4):559–576. https://doi.org/10.1109/tcst.2005.847331, URL https://doi.org/10.1109/tcst.2005.847331

Anh TT, Luong NC, Niyato D, et al (2018) Efficient training management for mobile crowd-machine learning: A deep reinforcement learning approach. https://doi.org/10.48550/ARXIV.1812.03633, URL https://arxiv.org/abs/1812.03633

Bangura M, Mahony R (2012) Nonlinear dynamic modeling for high performance control of a quadrotor. In: Proc. of the Australasian Conference on Robotics and Automation

Bangura M, Mahony R (2014) Real-time model predictive control for quadrotors. IFAC Proceedings Volumes 47(3):11,773–11,780. https://doi.org/10.3182/20140824-6-za-1003.00203, URL https://doi.org/10.3182/20140824-6-za-1003.00203

Bura A, HasanzadeZonuzy A, Kalathil D, et al (2021) Dope: Doubly optimistic and pessimistic exploration for safe reinforcement learning. https://doi.org/10.48550/ARXIV.2112.00885, URL https://arxiv.org/abs/2112.00885

Carvalho E, Susbielle P, Hably A, et al (2022) Neural enhanced control for quadrotor linear behavior fitting. In: 2022 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, https://doi.org/10.1109/icuas54217.2022.9836058, URL https://doi.org/10.1109/icuas54217.2022.9836058

Castillo A, Sanz R, Garcia P, et al (2019) Disturbance observer-based quadrotor attitude tracking control for aggressive maneuvers. Control Engineering Practice 82:14–23. https://doi.org/10.1016/j.conengprac.2018.09.016, URL https://doi.org/10.1016/j.conengprac.2018.09.016

Durand S, Boisseau B, Marchand N, et al (2018) Event-Based PID Control: Application to a Mini Quadrotor Helicopter. Journal of Control Engineering and Applied Informatics 20(1):36–47. URL https://hal.archives-ouvertes.fr/hal-01722845

Dydek ZT, Annaswamy AM, Lavretsky E (2013) Adaptive control of quadrotor UAVs: A design trade study with flight evaluations. IEEE Transactions on Control Systems Technology 21(4):1400–1406. https://doi.org/10.1109/tcst.2012.2200104, URL https://doi.org/10.1109/tcst.2012.2200104

Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT press

Gu W, Hu D, Cheng L, et al (2020) Autonomous wind turbine inspection using a quadrotor. In: 2020 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, https://doi.org/10.1109/icuas48674.2020.9214066, URL https://doi.org/10.1109/icuas48674.2020.9214066

Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Networks 2(5):359–366. https://doi.org/10.1016/0893-6080(89)90020-8, URL https://doi.org/10.1016/0893-6080(89)90020-8

Hwangbo J, Sa I, Siegwart R, et al (2017) Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters 2(4):2096–2103. https://doi.org/10.1109/lra.2017.2720851, URL https://doi.org/10.1109/lra.2017.2720851

Kim S, Choi S, Kim HJ (2013) Aerial manipulation using a quadrotor with a two DOF robotic arm. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, https://doi.org/10.1109/iros.2013.6697077, URL https://doi.org/10.1109/iros.2013.6697077

Koenig N, Howard A (2004) Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566). IEEE, https://doi.org/10.1109/iros.2004.1389727, URL https://doi.org/10.1109/iros.2004.1389727

Koller T, Berkenkamp F, Turchetta M, et al (2019) Learning-based model predictive control for safe exploration and reinforcement learning. https://doi.org/10.48550/ARXIV.1906.12189, URL https://arxiv.org/abs/1906.12189

Lambert NO, Drew DS, Yaconelli J, et al (2019) Low-level control of a quadrotor with deep model-based reinforcement learning. IEEE Robotics and Automation Letters 4(4):4224–4230. https://doi.org/10.1109/lra.2019.2930489, URL https://doi.org/10.1109/lra.2019.2930489

Mahony R, Kumar V, Corke P (2012) Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. IEEE Robotics & Automation Magazine 19(3):20–32. https://doi.org/10.1109/MRA.2012.2206474, URL http://

ieeexplore.ieee.org/document/6289431/

Pi CH, Hu KC, Cheng S, et al (2020) Low-level autonomous control and tracking of quadrotor using reinforcement learning. Control Engineering Practice 95:104,222. https://doi.org/10.1016/j.conengprac.2019.104222, URL https://doi.org/10.1016/j.conengprac.2019.104222

PX4 (2021) PX4/PX4-autopilot software. URL https://github.com/PX4/PX4-Autopilot

Robins A (1995) Catastrophic forgetting, rehearsal and pseudorehearsal. Connection Science 7(2):123–146. https://doi.org/10.1080/09540099550039318, URL https://doi.org/10.1080/09540099550039318

Schneider D (2020) The delivery drones are coming. IEEE Spectrum 57(1):28–29. https://doi.org/10.1109/mspec.2020.8946304, URL https://doi.org/10.1109/mspec.2020.8946304

Shi G, Shi X, O'Connell M, et al (2019) Neural lander: Stable drone landing control using learned dynamics. In: 2019 International Conference on Robotics and Automation (ICRA). IEEE, https://doi.org/10.1109/icra.2019.8794351, URL https://doi.org/10.1109/icra.2019.8794351

Silano G, Bednar J, Nascimento T, et al (2021) A multi-layer software architecture for aerial cognitive multi-robot systems in power line inspection tasks. In: 2021 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, https://doi.org/10.1109/icuas51884.2021.9476813, URL https://doi.org/10.1109/icuas51884.2021.9476813

Sutton RS, Barto AG (2018) Reinforcement Learning, 2nd edn. Adaptive Computation and Machine Learning series, Bradford Books, Cambridge, MA

Torrente G, Kaufmann E, Foehn P, et al (2021) Data-driven mpc for quadrotors. IEEE Robotics and Automation Letters https://doi.org/10.48550/ARXIV.2102.05773, URL https://arxiv.org/abs/2102.05773

Torres-González A, Capitán J, Cunha R, et al (2017) A multidrone approach for autonomous cinematography planning. In: ROBOT 2017: Third Iberian Robotics Conference. Springer International Publishing, pp 337–349, https://doi.org/10.1007/978-3-319-70833-1_28, URL https://doi.org/10.1007/978-3-319-70833-1_28

Wang X, Shirinzadeh B (2015) Nonlinear augmented observer design and application to quadrotor aircraft. Nonlinear Dynamics 80(3):1463–1481. https://doi.org/10.1007/s11071-015-1955-y, URL https://doi.org/10.1007/s11071-015-1955-y

Zhao W, Queralta JP, Westerlund T (2020a) Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, https://doi.org/10.1109/ssci47803.2020.9308468, URL https://doi.org/10.1109/ssci47803.2020.9308468

Zhao Z, Cerf S, Robu B, et al (2020b) Event-based control for online training of neural networks. IEEE Control Systems Letters 4(3):773–778. https://doi.org/https://doi.org/10.1109/LCSYS.2020.2981984, URL https://hal.archives-ouvertes.fr/hal-02509604, hal-02509604