Online Deep Learning for Improved Trajectory Tracking of Unmanned Aerial Vehicles Using Expert Knowledge

Andriy Sarabakha[1] and Erdal Kayacan[2]

Abstract— **This work presents an online learning-based control method for improved trajectory tracking of unmanned aerial vehicles using both deep learning and expert knowledge. The proposed method does not require the exact model of the system to be controlled, and it is robust against variations in system dynamics as well as operational uncertainties. The learning is divided into two phases: offline (pre-)training and online (post-)training. In the former, a conventional controller performs a set of trajectories and, based on the input-output dataset, the deep neural network (DNN)-based controller is trained. In the latter, the trained DNN, which mimics the conventional controller, controls the system. Unlike the existing papers in the literature, the network is still being trained for different sets of trajectories which are not used in the training phase of DNN. Thanks to the rule-base, which contains the expert knowledge, the proposed framework learns the system dynamics and operational uncertainties in real-time. The experimental results show that the proposed online learning-based approach gives better trajectory tracking performance when compared to the only offline trained network.**

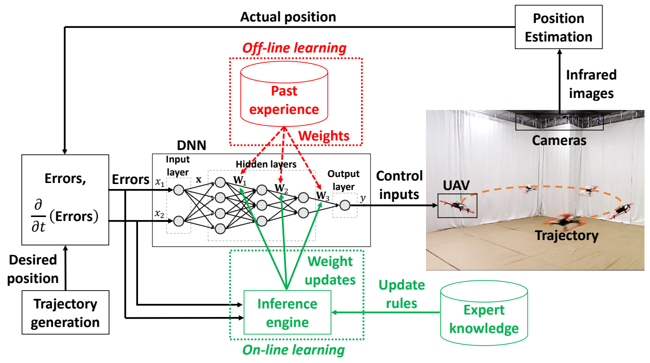


Fig. 1. Illustration of the proposed online learning-based control scheme. During the offline learning phase, a conventional controller performs a set of trajectories and the input-output dataset is used for the offline training of DNN. During the online learning phase, DNN controls the UAV and, by using the expert knowledge, improves further the performance in real-time.

I. INTRODUCTION

Research activities to develop increased autonomy in unmanned aerial vehicles (UAVs) have taken a centre stage in the recent years due to their usefulness in providing cost-effective solutions to dangerous, dirty and dull tasks, such as aerial grasping [1], emergency evacuation [2] and building inspection [3]. In these applications, it is crucial for UAVs to be able to fly autonomously in uncertain environments with variations in operating conditions [4]. Therefore, in such conditions, adaptability is a must rather than a choice.

Given the ability of artificial neural networks (ANNs) to generalise knowledge from training samples, an ANN-based controller can be used to control nonlinear dynamic systems [5]. On the other hand, deep neural networks (DNNs) can approximate non-linear functions with exponentially lower number of training parameters and higher sample complexity when compared to ANNs [6]. Therefore, DNNs propose a novel approach to enhance the control strategies [7].

In the literature, ANNs have successfully been integrated with control system design to improve tracking performance in uncertain environments [8]. In [9], the unknown part of the dynamical model of a quadcopter is modelled by DNN. In [10], DNN is used for direct inverse control of the quadrotor in simulation. In [11] and [12], DNNs are used to learn the dynamics of helicopter and multicopter, respectively. In [13], DNN pre-cascaded module is used to improve the performance of UAV in tracking arbitrary hand-drawn trajectory. However, in all these works, DNNs are trained offline and, then, used online without further learning. In other words, while the dynamics are learnt in the training phase, the controller is not updated in the testing phase – DNN simply mimics the conventional controller – and the operational uncertainties are no longer learnt.

Unlike the traditional use of DNNs in literature, in this work, we propose an online DNN-based approach for improving trajectory tracking performance of UAVs. After an offline pre-training phase with past flight data, a DNN-based controller is used in real-time to control the UAV. Without any prior knowledge of the system, besides the training data, the proposed approach shows its capability to reduce the trajectory tracking error online by compensating for internal uncertainties and external disturbances. Moreover, it is shown that the DNN module is computationally suitable for real-time operations and adequate for arbitrary trajectory, making it applicable to the real-world tasks. Furthermore, the proposed approach employs the expert knowledge for the online training. The overall control architecture and its training process are depicted in Fig. 1.

This work is organised as follows. The problem is formulated in Section II. Section III introduces the proposed approach. Then, Section IV presents the experimental setup. Section V provides real-time experiments with quadcopter UAV, to validate the proposed method. Finally, Section VI summarises this work with conclusions and future work.

[1]Andriy Sarabakha is with School of Mechanical and Aerospace Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore, 639798. `andriy001@e.ntu.edu.sg`

[2]Erdal Kayacan is with Department of Engineering, Aarhus University, Aabogade 34, Aarhus N, 8200, Denmark. `erdal@eng.au.dk`

II. PROBLEM FORMULATION

In this work, we consider a problem of designing a learning feedback control algorithm for a dynamical system, such as UAV. Our objective is to learn a control strategy of the system to achieve a high-accuracy tracking. To describe the problem, we introduce the dynamical model of UAV first.

A. Dynamical Model of Unmanned Aerial Vehicle

The world-fixed reference frame is $\mathcal{F}_W = \{\vec{\mathbf{x}}_W, \vec{\mathbf{y}}_W, \vec{\mathbf{z}}_W\}$ and the body frame is $\mathcal{F}_B = \{\vec{\mathbf{x}}_B, \vec{\mathbf{y}}_B, \vec{\mathbf{z}}_B\}$. The absolute position of UAV $\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is given by three Cartesian coordinates at its center of gravity in $\mathcal{F}_W$, and its attitude $\mathbf{o} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$ is given by three Euler angles. The rotation matrix from $\mathcal{F}_B$ to $\mathcal{F}_W$ is given by the combination of three single rotation matrices around ϕ, θ and ψ. The time derivative of the position gives the linear velocity $\mathbf{v} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$ of UAV expressed in $\mathcal{F}_W$. Equivalently, the time derivative of the attitude $\boldsymbol{\omega} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$ gives the angular velocity in $\mathcal{F}_W$ and $\boldsymbol{\omega}_B = \begin{bmatrix} p & q & r \end{bmatrix}^T$ is the angular velocity in $\mathcal{F}_B$.

The vector of control inputs $\mathbf{u}$ is chosen as:

$$\mathbf{u} = \begin{bmatrix} T & \tau_\phi & \tau_\theta & \tau_\psi \end{bmatrix}^T, \tag{1}$$

where T is the total thrust along $\vec{\mathbf{z}}_B$, whereas τ_ϕ, τ_θ and τ_ψ are moments around $\vec{\mathbf{x}}_B$, $\vec{\mathbf{y}}_B$ and $\vec{\mathbf{z}}_B$, respectively. Finally, the dynamical model of UAV is given as in [14]:

$$\begin{cases} \dot{x} = v_x & \dot{u} = \frac{1}{m}\left(c_\phi c_\psi s_\theta + s_\phi s_\psi\right) T \\ \dot{y} = v_y & \dot{v} = \frac{1}{m}\left(c_\phi s_\psi s_\theta - c_\psi s_\phi\right) T \\ \dot{z} = v_z & \dot{w} = -g + \frac{1}{m} c_\phi c_\theta T \\ \dot{\phi} = p + s_\phi t_\theta q + c_\phi t_\theta r & \dot{p} = \frac{I_y - I_z}{I_x} qr + \frac{1}{I_x}\tau_\phi \\ \dot{\theta} = c_\phi q - s_\phi r & \dot{q} = \frac{I_z - I_x}{I_y} pr + \frac{1}{I_y}\tau_\theta \\ \dot{\psi} = \frac{s_\phi}{c_\theta} q + \frac{c_\phi}{c_\theta} r & \dot{r} = \frac{I_x - I_y}{I_z} pq + \frac{1}{I_z}\tau_\psi, \end{cases} \tag{2}$$

where m is the mass of UAV, g is the gravity acceleration constant, $\mathbf{I} = \text{diag}(I_x, I_y, I_z)$ is the inertia matrix, $c_\star$, $s_\star$ and $t_\star$ denote $\cos(\star)$, $\sin(\star)$ and $\tan(\star)$, respectively.

Remark 1: The dynamical system in (2) is nonlinear, coupled and underactuated. Therefore, an advanced controller is required.

The system in (2) can be written in a general form as:

$$\begin{cases} \dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u} + d \\ \mathbf{y} = h(\mathbf{x}), \end{cases} \tag{3}$$

where $\mathbf{x} = \begin{bmatrix} x & y & z & \phi & \theta & \psi & u & v & w & p & q & r \end{bmatrix}^T$, d is the disturbance term, $h(\mathbf{x}) = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^T$ and $\mathbf{u}$ is defined in (1).

B. Problem Description

If a precise model of the system exists, then the inversion of the system can be computed. Let $h \circ f$ denote the composition of functions h and f; while f^i denote the i-th composition of function f, i.e., $f^0(\mathbf{x}) = \mathbf{x}$ and $f^i(\mathbf{x}) = f^{i-1} \circ f(\mathbf{x}) \quad \forall i \in \mathbb{N}^+$ [15]. Let n define the dimension of the system's input, i.e., $\mathbf{u} \in \mathbb{R}^n$, and let $\mathbf{r}$ define the vector of relative degrees of the system, s.t. $\arg\min_{\mathbf{r}_i} \frac{\partial}{\partial \mathbf{u}_i}\left(h \circ f^{\mathbf{r}_i - 1} \circ (f(\mathbf{x}) + g(\mathbf{x})\mathbf{u})\right) \neq 0 \quad \forall i \in [1, n]$. Then, the input and the output of the system are related by

$$\mathbf{y}_{k+\mathbf{r}_i} = h \circ f^{\mathbf{r}_i - 1} \circ (f(\mathbf{x}_k) + g(\mathbf{x}_k)\mathbf{u}_k). \tag{4}$$

If $\mathbf{y}$ is affine in $\mathbf{u}$, then (4) becomes

$$\mathbf{y}_{k+\mathbf{r}_i} = F(\mathbf{x}_k) + G(\mathbf{x}_k)\mathbf{u}_k, \tag{5}$$

where $F_i(\mathbf{x}_k) = h \circ f^{\mathbf{r}_i}(\mathbf{x}_k)$ and $G_i(\mathbf{x}_k) = \frac{\partial}{\partial \mathbf{u}_{k,i}}\left(h \circ f^{\mathbf{r}_i - 1} \circ (f(\mathbf{x}_k) + g(\mathbf{x}_k)\mathbf{u}_k)\right)$ are the decoupling matrices. Finally, the control law at time k to track the desired output of the system $\mathbf{y}^*$ can be written as in [16]:

$$\mathbf{u}_{k,i} = [G(\mathbf{x}_k)]^{-1}\left(\mathbf{y}^*_{k+\mathbf{r}_i} - F(\mathbf{x}_k)\right). \tag{6}$$

However, in a real system, the system's parameters might be unknown and difficult to estimate, e.g., moments of inertia. What is more, these parameters might change during the operation of the system, e.g., mass. Moreover, it is not always possible to predict the external disturbance term. Therefore, an adaptive controller which can learn online is required. Our objective is to learn the control of the system by only looking at the performance of the system, i.e., in our case, the tracking error:

$$\mathbf{e}_k = \mathbf{y}^*_k - \mathbf{y}_k, \tag{7}$$

and its time derivative:

$$\dot{\mathbf{e}}_k = \dot{\mathbf{y}}^*_k - \dot{\mathbf{y}}_k. \tag{8}$$

Thus, $\mathbf{y}_k$ and $\dot{\mathbf{y}}_k$ is the only required information about the system.

III. METHODOLOGY

By their nature, DNNs are distinguished from more common single-hidden-layer ANNs by their depth. The neurons are organised in input, multiple-hidden and output layers. In DNN, like in classical ANNs, the weights are modified using a learning process governed by the training rules.

$$f(\mathbf{x}) = \begin{bmatrix} u & v & w & p + s_\phi t_\theta q + c_\phi t_\theta r & c_\phi q - s_\phi r & \frac{s_\phi}{c_\theta} q + \frac{c_\phi}{c_\theta} r & 0 & 0 & g & \frac{I_y - I_z}{I_x} qr & \frac{I_z - I_x}{I_y} pr & \frac{I_x - I_y}{I_z} pq \end{bmatrix}^T,$$

$$g(\mathbf{x}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{m}\left(c_\phi c_\psi s_\theta + s_\phi s_\psi\right) & -\frac{1}{m}\left(c_\phi s_\psi s_\theta - c_\psi s_\phi\right) & -\frac{1}{m} c_\phi c_\theta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix}^T,$$

A. *Offline Pre-Training*

During the offline pre-training phase, a supervised learning approach is used, in which a feed-forward DNN learns to control the system from a conventional controller – proportional-integral-derivative (PID) controller, in our case. In this control scheme, shown in Fig. 2a, PID controller controls the system alone. Hence, it is utilized as an ordinary feedback controller to ensure the global asymptotic stability of the system and provide labelled training samples for DNN. The training of DNN requires the availability of a large number of labelled training samples. Each labelled training sample consists of an input and expected output pair $< \{\mathbf{e}_k, \dot{\mathbf{e}}_k\}, \{\mathbf{u}_k\} >$. The training of DNN involves back-propagation to minimize the loss over all training examples. After the training, DNN can approximate the mapping from the training inputs to the outputs. The pseudo-code of offline pre-training is provided in Algorithm 1.

Algorithm 1: Offline pre-training of DNN.

```
Input: -
Output: Pre-trained DNN$_0$
begin
    while $k < MaxSamples$ do
        Get $\mathbf{y}_k$, $\mathbf{y}^*_k$, $\dot{\mathbf{y}}_k$, $\dot{\mathbf{y}}^*_k$ and $\mathbf{u}_k$
        $\mathbf{e}_k \leftarrow \mathbf{y}^*_k - \mathbf{y}_k$ by using (7)
        $\dot{\mathbf{e}}_k \leftarrow \dot{\mathbf{y}}^*_k - \dot{\mathbf{y}}_k$ by using (8)
        Collect $< \{\mathbf{e}_k, \dot{\mathbf{e}}_k\}, \{\mathbf{u}_k\} >$
    end
    DNN$_0$ $\leftarrow$ ConstructNetworkLayers()
    $\mathbf{w} \leftarrow$ InitializeWeights()
    Train DNN$_0$ on $< \{\mathbf{e}, \dot{\mathbf{e}}\}, \{\mathbf{u}\} >$
end
```

B. *Online Training*

During the online training phase, DNN controls the system, and, at the same time, learns how to improve the control performances. Since DNN training requires supervised learning, another process has to provide a feedback about its performances. In our case, fuzzy logic system (FLS) is used to provide this information. By definition, FLS incorporates the expert knowledge in form of rules and uses this knowledge to provide some useful information [17]. The control structure for online training is illustrated in Fig. 2b.

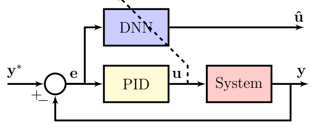
In our approach, FLS observes the behaviour of the system controlled by DNN, and, depending on the situation, corrects the action of DNN. The possible evolutions of the error are depicted in Fig. 3. If the error is positive, i.e, $e_i > 0$, and its time derivative is also positive, i.e., $\dot{e}_i > 0$, then the system diverges (top red curve in Fig. 3). In this case, FLS will force DNN to decrease the control signal u_i significantly to stabilize the system, i.e., $\Delta u_i \ll 0$. In another possible case, if the error is negative, i.e., $e_i < 0$, and its time derivative is zero, i.e., $\dot{e}_i = 0$, then the error is steady (bottom blue line in Fig. 3). In this case, DNN falls down in a local minimum and FLS will give a small positive perturbation, i.e., $\Delta u_i > 0$. Finally, if the error is zero, i.e., $e_i = 0$, and its time derivative is also zero, i.e., $\dot{e}_i = 0$, then, this is the optimal case (green line in Fig. 3) and no action has to be taken, i.e., $\Delta u_i = 0$.

These empirical rules can be formally described by a Mamdani FLS with triangular membership functions to represent the fuzzy sets. The rules for each possible case are summarized by the rule-base in Table I. The inputs to the FLC are selected to be the tracking error and its time derivative, i.e., e_i and $\dot{e}_i$; while the output is the correction signal, i.e., Δu_i. The input is represented by three fuzzy sets: negative, zero and positive; while the output can belong to five fuzzy sets: big decrease, small decrease, no changes, small increase and big increase.

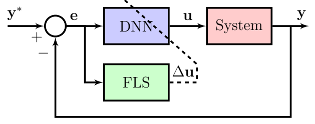
However, FLS requires operations among fuzzy sets which are time-consuming. Therefore, by using a similar approach to the one described in [18], a fuzzy mapping which represents the FLS in Table I can be generated for a general multidimensional case:

$$\Delta \mathbf{u}_k = -\alpha \left(\frac{1}{2}\mathbf{e}_k + \dot{\mathbf{e}}_k - \frac{1}{2}\text{abs}(\mathbf{e}_k) \odot \dot{\mathbf{e}}_k \right), \qquad (9)$$

where $\odot$ denotes Hadamard product and α is the adaptation rate. The fuzzy mapping reduces significantly the computation time which makes this approach suitable for real-time systems [19]. The pseudo-code of online training is provided in Algorithm 2.



(a) Block diagram of the offline pre-training of DNN by PID.



(b) Block diagram of the online post-training of DNN by FLS.

Fig. 2. Block diagrams of two control paradigms: offline pre-training and online post-training.

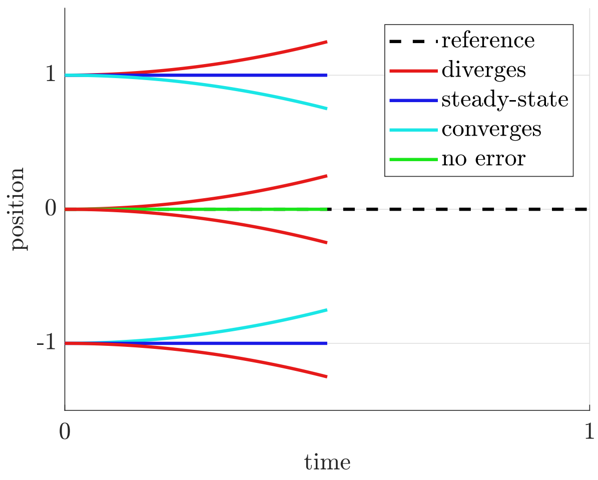


Fig. 3. Possible evolution of the tracking error in a dynamical system. The system can diverge (red curves), converge (cyan curves), it can have a steady-state error (blue lines), or the error can be zero (green line).

TABLE I
RULE-BASE FOR THE UPDATES OF u_i.

e_i	$\dot{e}_i$		
	Negative	Zero	Positive
Negative	**Big decrease**	**Small decrease**	**No changes**
Zero	**Big decrease**	**No changes**	**Big increase**
Positive	**No changes**	**Small increase**	**Big increase**

IV. EXPERIMENTAL SETUP

The experimental platform used in this work is Parrot Bebop 2 quadcopter UAV. This UAV is controlled via a Wi-Fi connection and the robot operating system (ROS) is used to communicate with UAV. The motion capture system provides the UAV's real-time position at 240Hz. This position is fed into the ground station computer (CPU: 2.6GHz, 64bit, quad-core; GPU: 4GB; RAM: 16GB DDR4) where the algorithms are executed. Once the control signal is computed, it is sent to the UAV at 100Hz rate.

For the attitude/velocity tracking, the onboard nonlinear geometric controller on $\mathsf{SE}(3)$ is used [20]. The attitude controller is responsible for mapping the high-level control inputs, i.e., $\begin{bmatrix}\theta_k^* & \phi_k^* & w_k^*\end{bmatrix}$, to the low-level control commands, i.e., $\mathbf{u}_k$ in (1).

A. Deep Neural Network Structure

Three feed-forward DNNs with hyperbolic tangent ($\tanh$) activation functions are used to learn the control mapping for each controlled axis: x, y and z. The inputs to DNN for the x-axis are the errors and their time derivatives on the x-axis, $\{e_{x,k}, e_{x,k-1}, e_{x,k-2}, \dot{e}_{x,k}, \dot{e}_{x,k-1}, \dot{e}_{x,k-2}\}$, and the output is the desired pitch angle, $\{\theta_k^*\}$. Similarly, the inputs to DNN for the y-axis are the errors and their time derivatives on the y-axis, $\{e_{y,k}, e_{y,k-1}, e_{y,k-2}, \dot{e}_{y,k}, \dot{e}_{y,k-1}, \dot{e}_{y,k-2}\}$, and the output is the desired roll angle, $\{\phi_k^*\}$. Finally, the inputs to DNN for the z-axis are the errors and their time derivatives on the z-axis, $\{e_{z,k}, e_{z,k-1}, e_{z,k-2}, \dot{e}_{z,k}, \dot{e}_{z,k-1}, \dot{e}_{z,k-2}\}$, and the output is the desired vertical velocity, $\{w_k^*\}$.

Remark 2: Both DNN controllers with and without online learning consist of three parallel sub-networks for x, y and z axes.

In our case, after some heuristic analysis and experimental trials, the architecture of each network is chosen to consist of 6 input neurons ($n_I = 6$), 6 scaling neurons, 2 fully connected hidden layers ($n_L = 2$) with 6 neurons in each layer ($n_H = 6$), 1 unscaling neuron and 1 output neuron ($n_O = 1$). From the asymptotic analysis, the runtime complexity for the forward-propagation is $\mathrm{O}(n_L \cdot n_H^3 + n_L \cdot n_H) \equiv \mathrm{O}(n_L \cdot n_H^3)$. While the runtime complexity for the back-propagation is $\mathrm{O}(n_{QN} \cdot n_L \cdot n_H^4 + n_L \cdot n_H^3) \equiv \mathrm{O}(n_{QN} \cdot n_L \cdot n_H^4)$, where n_{QN} is the number of iterations in the quasi-Newton method. Moreover, the runtime complexity for the fuzzy mapping in (9) is constant w.r.t. the architecture of the network, i.e., $\mathrm{O}(1)$. The dominant operation in DNN$_0$ is the forward-propagation; therefore, the runtime complexity of DNN$_0$ is polynomial. However, DNN with online learning involves both forward-propagation and back-propagation; therefore, the runtime complexity of DNN is also polynomial but asymptotic to $\mathrm{O}(n_{QN} \cdot n_L \cdot n_H^4)$. Therefore, the proposed architecture was chosen as a compromise between the learning capability of the neural network and the update time through the back-propagation.

The error type is an important term in the loss index, and, in our case, it is chosen as the normalized squared error. The initialization algorithm is used to bring the neural network to a stable region of the loss function, and, in our case, it is selected as the random search. The training algorithm is the core part of the training, and, in our case, the quasi-Newton method is the most suitable choice for both offline and online training.

Algorithm 2: Online post-training of DNN.

```
Input: Pre-trained DNN$_0$
Output: Trained DNN
Result: Learns and controls the system online
begin
    DNN $\leftarrow$ DNN$_0$
    repeat
        Get $\mathbf{y}_k$, $\mathbf{y}_k^*$, $\dot{\mathbf{y}}_k$ and $\dot{\mathbf{y}}_k^*$
        $\mathbf{e}_k \leftarrow \mathbf{y}_k^* - \mathbf{y}_k$ by using (7)
        $\dot{\mathbf{e}}_k \leftarrow \dot{\mathbf{y}}_k^* - \dot{\mathbf{y}}_k$ by using (8)
        $\Delta\mathbf{u}_k \leftarrow$ FLS$(\mathbf{e}_k, \dot{\mathbf{e}}_k)$ by using (9)
        Calculate $\mathbf{u}_k$ by forward-propagation
        Update by back-propagation
          $< \{\mathbf{e}_k, \dot{\mathbf{e}}_k\}, \{\mathbf{u}_k + \Delta\mathbf{u}_k\} >$
    until landing
end
```

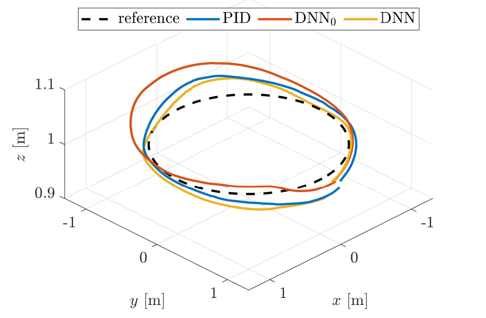
B. Data Collection

To prepare the training samples of the flight data, the system was controlled by a conventional controller alone, while the position errors and their time derivatives were collected as training inputs, and the control signal was saved as the labelled output. By using PID controller, $100'000$ instances have been collected in the training dataset for each axis. This dataset is large enough for our application, however, the proposed method does not have any limitations on the dataset size. The training data include slow circular and eight-shaped trajectories on xy-, xz- and yz-planes with the reference speed of $1\mathrm{m/s}$.

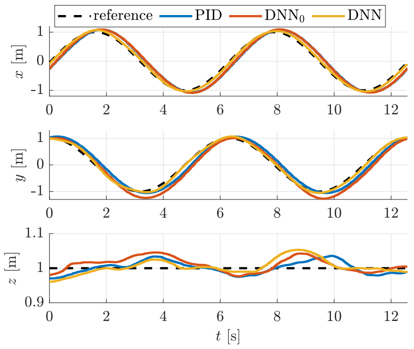
V. EXPERIMENTAL RESULTS

In order to validate the capabilities of the proposed controller in Section III, the trajectory following problem of a quadcopter UAV is considered. The proposed control architecture and its training process are depicted in Fig. 1. Three different types of trajectories have been tested: slow circular, fast circular and square-shaped. In order to show the efficiency and efficacy of the DNN-based controller, it is compared with a well-tuned PID controller (used during the offline pre-training) and DNN controller without online training, DNN$_0$.

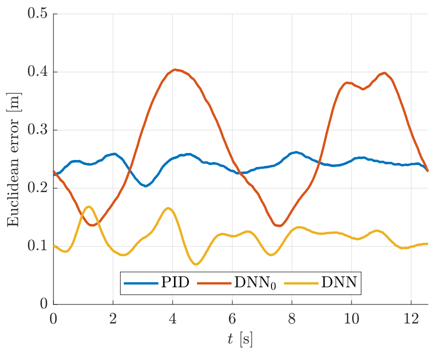
The first study case is the tracking of the slow circular trajectory with radius $1\mathrm{m}$ at $1\mathrm{m/s}$ which has been used during the pre-training phase. Fig. 4a shows the results of the 3D trajectory tracking for the first case. The projections on x, y and z axes of this portion of the trajectory are shown on Fig. 4b. The evolution of the Euclidean error for the tested controllers is illustrated in Fig. 4c. The second study case is the tracking of the fast circular trajectory with radius $1\mathrm{m}$ at $2\mathrm{m/s}$ which has not been used during the pre-training phase. Fig. 4d shows the results of the 3D trajectory tracking of the second case. The projections on x, y and z axes of this portion of the trajectory are shown on Fig. 4e. The evolution of the Euclidean error for the tested controllers is illustrated in Fig. 4f. The third study case is the tracking of the square-shaped trajectory with side length $2\mathrm{m}$ at $1\mathrm{m/s}$ which also has not been used during the pre-training phase. Fig. 4g shows the results of the 3D trajectory tracking of the third case. The projections on x, y and z axes of this portion of the trajectory are shown on Fig. 4h. The evolution of the Euclidean error for the tested controllers is illustrated in Fig. 4i.



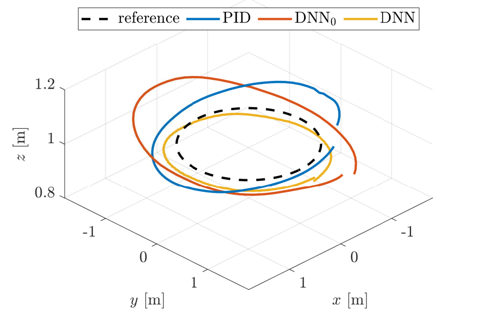
(a) 3D view for the tracking of the slow circular trajectory.



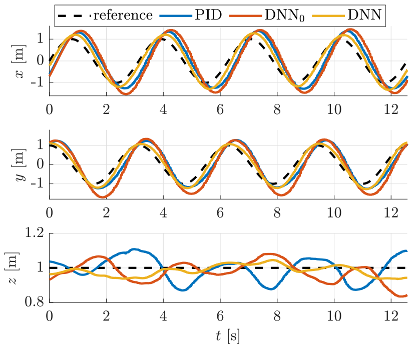
(b) Projection of the slow circular trajectory tracking on x, y and z axes.



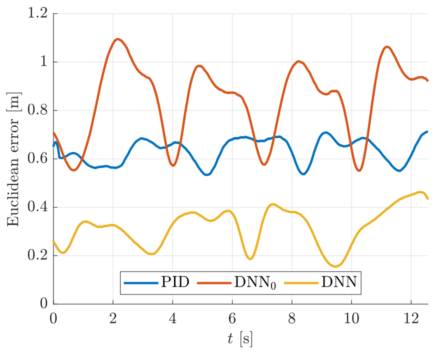
(c) Euclidean error for the slow circular trajectory tracking.



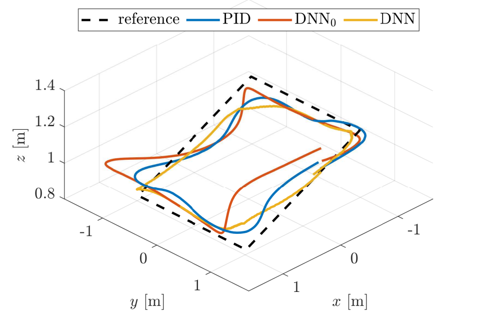
(d) 3D view for the tracking of the fast circular trajectory.



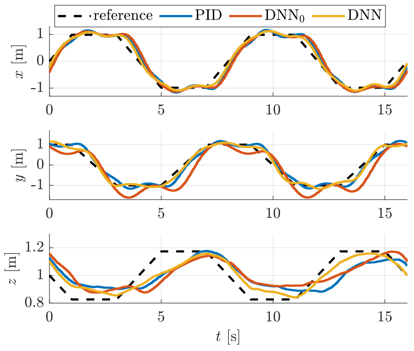
(e) Projection of the fast circular trajectory tracking on x, y and z axes.



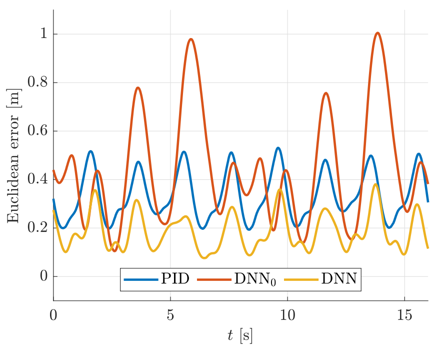
(f) Euclidean error for the fact circular trajectory tracking.



(g) 3D view for the tracking of the square-shaped trajectory.



(h) Projection of the square-shaped trajectory tracking on x, y and z axes.



(i) Euclidean error for the square-shaped trajectory tracking.

Fig. 4. Experimental results for different controllers (PID, DNN$_0$ and DNN) on three trajectories (slow circular, fast circular and square-shaped). The slow circular trajectory has been used for the offline pre-training of DNN$_0$, while fast circular and square-shaped trajectories have not been used during the pre-training. It is possible to observe that DNN controller with online training is able to learn the system dynamics and improve the tracking performances on all tested trajectories.

A. *Discussion*

A sample of experimental results for three controllers (PID, DNN_0 and DNN) on three trajectories (slow circular, fast circular and square-shaped) are illustrated on Figs. 4. It is possible to observe that DNN controller with online training is able to learn the system dynamics and decrease the tracking error over time on all tested trajectories. As visualized from Figs. 4b, 4e and 4h, DNN has faster responses, since it is able to estimate the desired control signal in (6) and predict the evolution of the system dynamics. It has to be emphasised that online DNN evolves from pre-trained DNN_0 during the learning process. Moreover, as expected, DNN_0 without online learning has poor performances on the trajectories which have not been used for its training.

For a statistical analysis of control performances, the experiments are repeated five times for each trajectory-controller combination under the same conditions. Fig. 5 presents a box-plot to compare the tracking performances of three different controllers on three tested trajectories. It is possible to observe that on average DNN controller with online learning outperforms other controllers on the tested trajectories. In addition, the maximum absolute error is also lower for the online DNN-based controller, even for previously unseen trajectory. Finally, the variance of the error is similar for PID and DNN with online learning controllers.

As can be seen from Table II, the DNN-based controller with online learning outperforms both PID and DNN_0 for all tested trajectory in terms of mean absolute error (MAE). Averaged results from numerous experiments depict that the overall improvement of 60%, 61% and 46% in MSE is achieved as compared to a well-tuned PID controller for slow circular, fast circular and square-based trajectories, respectively. While this ratio goes up to 62%, 70% and 64% when compared with pre-trained DNN_0 for the same trajectories.

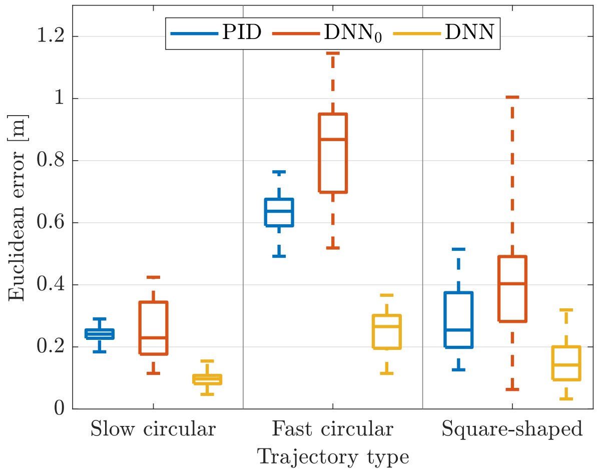


Fig. 5. Box-plot of the tracking performances of different controllers on three trajectories. For each trajectory, the experiments are repeated five times under the same conditions. It is possible to observe that in average the DNN controller with online learning outperforms other controllers on the tested trajectories.

TABLE II
COMPARISON OF DIFFERENT CONTROLLERS IN TERMS OF MAE [m].

Trajectory	**PID**	**DNN_0**	**DNN**
Slow circle	0.241	0.254	0.097
Fast circle	0.632	0.833	0.250
Square-shaped	0.284	0.431	0.154

Though the online DNN-based controllers can learn promptly how to control the system, the computing time is still the main drawback of this controller with online back-propagation. The computing time is polynomially proportional to the number of hidden layers and the number of neurons in each layer. Therefore, deeper is the network, more complex functions it can learn but more computational power it requires. The average experimental computation time for DNN with online back-propagation is around 5.4ms, while for PID and DNN_0 without online learning this time is only $8\mu\text{s}$ and $16\mu\text{s}$, respectively. However, 5.4ms is still an acceptable time for real-time applications, which allows the controller to run at almost 200Hz.

VI. CONCLUSIONS

In this work, we have presented a novel approach for a high-level control of UAV that improves online the trajectory tracking performances by using deep learning and expert knowledge. The learning is subdivided into two phases: offline pre-training and online training. During the offline learning phase, a conventional controller performs a set of trajectories and the batch of training samples is collected. Then, DNN-based controller, DNN_0, is pre-trained on the collected data samples. However, DNN_0 cannot adapt to the new flying conditions unseen during the pre-training; therefore, the online training is required. During the online learning phase, DNN controls the system and adapts the control input to improve the tracking performance. The expert knowledge encoded into the rule-base, thanks to the fuzzy mapping, provides the adaptation information to DNN allowing the real-time learning. Once DNNs are trained during the flight on UAV, the experimental results show that the proposed approach improves the performance by around 50%. We believe that the results of this study will open the doors to a wider use of DNN-based controllers with online training in real-world control applications as the proposed structure is suitable to deploy in real-time control systems.

In the future, we will test the DNN-based controller for the aerial transportation where the system dynamics change drastically. In addition, we will extensively analyse the parameters and architecture of DNN and their performances. Moreover, the analytical stability proof of the proposed approach will be provided.

ACKNOWLEDGMENT

This research was partially supported by the Singapore Ministry of Education (RG185/17).

References

[1] G. Loianno, V. Spurny, J. Thomas, T. Baca, D. Thakur, D. Hert, R. Penicka, T. Krajnik, A. Zhou, A. Cho, M. Saska, and V. Kumar, "Localization, grasping, and transportation of magnetic objects by a team of mavs in challenging desert-like environments," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1576–1583, July 2018.

[2] A. Sarabakha and E. Kayacan, "Y6 Tricopter Autonomous Evacuation in an Indoor Environment Using Q-Learning Algorithm," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec 2016, pp. 5992–5997.

[3] L. Teixeira and M. Chli, "Real-time local 3d reconstruction for aerial inspection using superpixel expansion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 4560–4567.

[4] N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermüller, and Y. Aloimonos, "GapFlyt: Active Vision Based Minimalist Structure-Less Gap Detection For Quadrotor Flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2799–2806, Oct 2018.

[5] E. Kayacan, E. Kayacan, H. Ramon, and W. Saeys, "Adaptive Neuro-Fuzzy Control of a Spherical Rolling Robot Using Sliding-Mode-Control-Theory-Based Online Learning Algorithm," *IEEE Transactions on Cybernetics*, vol. 43, no. 1, pp. 170–179, Feb 2013.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[7] S. Zhou, M. K. Helwa, and A. P. Schoellig, "An Inversion-Based Learning Approach for Improving Impromptu Trajectory Tracking of Robots With Non-Minimum Phase Dynamics," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1663–1670, July 2018.

[8] B. J. Emran and H. Najjaran, "Adaptive neural network control of quadrotor system under the presence of actuator constraints," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2017, pp. 2619–2624.

[9] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, Dec 2016, pp. 4653–4660.

[10] S. A. Nivison and P. P. Khargonekar, "Development of a robust deep recurrent neural network controller for flight applications," in *2017 American Control Conference (ACC)*, May 2017, pp. 5336–5342.

[11] A. Punjani and P. Abbeel, "Deep learning helicopter dynamics models," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 3223–3230.

[12] N. Mohajerin and S. L. Waslander, "Modular Deep Recurrent Neural Network: Application to Quadrotors," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2014, pp. 1374–1379.

[13] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5183–5189.

[14] R. Mahony, V. Kumar, and P. Corke, "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor," *Robotics Automation Magazine, IEEE*, vol. 19, no. 3, pp. 20–32, 2012.

[15] M. Sun and D. Wang, "Analysis of Nonlinear Discrete-Time Systems with Higher-Order Iterative Learning Control," *Dynamics and Control*, vol. 11, no. 1, pp. 81–96, Jan 2001.

[16] S. Zhou, M. K. Helwa, and A. P. Schoellig, "Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, Dec 2017, pp. 5201–5207.

[17] J. M. Mendel, *Type-1 Fuzzy Systems*. Springer International Publishing, 2017, pp. 101–159.

[18] A. Sarabakha, C. Fu, E. Kayacan, and T. Kumbasar, "Type-2 Fuzzy Logic Controllers Made Even Simpler: From Design to Deployment for UAVs," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 6, pp. 5069–5077, June 2018.

[19] A. Sarabakha, C. Fu, and E. Kayacan, "Double-Input Interval Type-2 Fuzzy Logic Controllers: Analysis and Design," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, July 2017, pp. 1–6.

[20] T. Lee, M. Leok, and N. H. McClamroch, "Nonlinear Robust Tracking Control of a Quadrotor UAV on SE(3)," *Asian Journal of Control*, vol. 15, no. 2, pp. 391–408, 2013.