Online Deep Fuzzy Learning for Control of Nonlinear Systems Using Expert Knowledge

Andriy Sarabakha, *Student Member, IEEE,* and Erdal Kayacan, *Senior Member, IEEE*

(Special Issue on "Deep Fuzzy Models")

***Abstract*—This work presents an online learning method for improved control of nonlinear systems by combining deep learning and fuzzy logic. Given the ability of deep learning to generalise knowledge from training samples, the proposed method requires minimum amount of information about the system to be controlled. However, in robotics, particularly in aerial robotics where the operating conditions may vary, online learning is required. In this study, fuzzy logic is preferred to provide supervising feedback to the deep model for adapting to variations in the system dynamics as well as new operational conditions. The learning method is divided into two phases: offline pre-training and online post-training. In the former, the system is controlled by a conventional controller and a deep fuzzy neural network (DFNN) is pre-trained based on the recorded input-output dataset, in order to approximate the inverse dynamical model of the system. In the latter, only the pre-trained DFNN is used to control the system. In this phase, the fuzzy logic, which encodes the expert knowledge, is utilized to observe the behaviour of the system and to correct the action of DFNN instantaneously. The experimental results show that the proposed online learning-based approach improves the trajectory tracking performance of the unmanned aerial vehicle.**

***Index Terms*—Deep learning, fuzzy logic, adaptive process control, nonlinear systems, aerial robotics.**

I. Introduction

In recent years, research activities to develop learning or adaptive controllers for nonlinear systems have taken a centre stage due to their usefulness in providing effective solutions in various robotic applications. Designing nonlinear controllers to achieve high-accuracy tracking is typically difficult, due to parameters uncertainties, external disturbance and other nonidealities in the systems. In such cases, the learning capability of the controller is a must rather than a choice.

Deep learning is based on multi-layered deep neural networks (DNNs) which are distinguished from the more commonplace single-hidden-layer artificial neural networks by their depth; that is the number of layers through which data must pass in a multi-step process [1]. DNNs can effectively be used to solve advanced tasks similar to or even better than human experts [2]. In many areas of machine learning, DNNs have made notable advances, e.g., image classification [3], speech recognition [4] and language translation [5]. On the other hand, fuzzy logic attempts to mathematically and systematically emulate human reasoning and decision making [6]. Moreover, fuzzy logic represents an excellent concept to create a connection between human logic and computational paradigms [7]. Fuzzy logic systems (FLSs) have also an exceptional ability to handle uncertainties [8]. In various fields, FLSs have been proposed to solve many practical problems efficiently, e.g., multicriteria decision-making [9], conditional density estimation [10] and system control [11].

Each of these perceptive techniques has distinct properties that make it suitable for particular problems and not for others. For instance, DNNs are good at approximating knowledge, but they do not explain how they take their decisions. Differently, FLSs are good at explaining their decisions, but generally, they are not good at acquiring new information. The limitations of these two techniques have been a driving force behind the creation of hybrid systems where the combination of DNN and FLC can overcome the drawbacks of each individual method [12]. In the literature, there are attempts to integrate strengths of learning capability of neural networks and reasoning ability provided by fuzzy logic, called fuzzy neural network (FNN), for emission prediction [13], movie classification [14] and robot control [15]. However, these approaches usually utilise the sequential learning paradigms [16]. For example, in [13], [15], first, the original inputs are fuzzified and, then, the fuzzy numbers are fed into the neural network. Contrarily, the method in [14], first, transforms the original data by using DNN and, then, the deep representation is fuzzified at the output layer. Correspondingly, one may ask whether a joint learning framework exists that fuses wisely these two methods.

In this work, we propose a novel online deep fuzzy neural network (DFNN) framework which profoundly fuses DNN and FLS. In our framework, the first layer of DFNN fuzzifies the original inputs and, then, the fuzzy numbers are fed into subsequent layers of DFNN. The learning method is divided into two phases: conventional offline training and proposed innovative online training. After the offline pre-training phase, where the past data samples collected on the controlled system are used, the DFNN-based controller controls the system in real-time, while fuzzy logic observes the behaviour of the system and corrects the action of DFNN. With minimum amount of prior knowledge about the system, the proposed approach shows its capability to reduce tracking error online by compensating for internal and external disturbances.

The contributions of this study can be summarized as:

- To the best of our knowledge, for the first time FLS is used as a supervisor for the online training of DFNN.
- For the first time the DFNN-framework is used for learning online the inverse dynamics of a system.
- It is shown that DFNN module is computationally adequate for real-time control applications.

Andriy Sarabakha is with the School of Mechanical and Aerospace Engineering, Nanyang Technological University, Singapore, 639798. Email: andriy001@e.ntu.edu.sg

Erdal Kayacan is with the Department of Engineering, Aarhus University, Aarhus N, 8200, Denmark. Email: erdal@eng.au.dk

This work is organised as follows. The control problem of nonlinear systems is formulated in Section II. Section III introduces the proposed framework based on DFNNs. Then, Section IV provides simulation results for a nonlinear system, to validate the proposed method in a deterministic environment. Whereupon, Section V provides experimental results with quadcopter unmanned aerial vehicle (UAV) to validate the proposed method in a real-life problem. Finally, Section VI summarises this work with conclusions and future work.

II. Problem Formulation

Let us consider a general nonlinear multi-input multi-output system represented by its state-space model:

$$\begin{cases} \mathbf{x}(k+1) & = f(\mathbf{x}(k)) + g(\mathbf{x}(k))\mathbf{u}(k) \\ \mathbf{y}(k) & = h(\mathbf{x}(k)), \end{cases} \tag{1}$$

where $k \in \mathbb{N}^+$ is the time step, $\mathbf{x} \in \mathbb{R}^{n_S}$ is the state of the system, $\mathbf{u} \in \mathbb{R}^{n_I}$ is the input to the system, $\mathbf{y} \in \mathbb{R}^{n_O}$ is the output from the system, and $f : \mathbb{R}^{n_S} \to \mathbb{R}^{n_S}$, $g : \mathbb{R}^{n_S} \to \mathbb{R}^{n_S} \times \mathbb{R}^{n_I}$ and $h : \mathbb{R}^{n_S} \to \mathbb{R}^{n_O}$ are system functions. Let $\mathbf{r} \in \mathbb{N}_0^{n_O}$ be the vector of relative degrees of the system, which is the number of times one has to differentiate the output to have at least one of the inputs explicitly appearing [17], i.e.:

$$\arg\max_{\mathbf{r}_i} \frac{\partial}{\partial \mathbf{u}} \left[h_i \left(f^{\mathbf{r}_i - 1} \left(f(\mathbf{x}) + g(\mathbf{x})\mathbf{u} \right) \right) \right] \neq \mathbf{0}, i \in [1, n_O]. \tag{2}$$

Assumption 1. *The dynamical system in (1) has well-defined relative degrees in (2) [18], i.e.:*

$$\begin{aligned} \forall i \in [1, n_O] \quad & \exists \mathbf{r}_i \in \mathbb{N}_0 \quad | \\ & \arg\max_{\mathbf{r}_i} \frac{\partial}{\partial \mathbf{u}} \left[h_i \left(f^{\mathbf{r}_i - 1} \left(f(\mathbf{x}) + g(\mathbf{x})\mathbf{u} \right) \right) \right] \neq \mathbf{0}. \end{aligned} \tag{3}$$

The input and the output of the system are related by

$$\begin{aligned} & \mathbf{y}_i(k + \mathbf{r}_i) = \\ & \quad h_i \left(f^{\mathbf{r}_i - 1} \left(f(\mathbf{x}(k)) + g(\mathbf{x}(k))\mathbf{u}(k) \right) \right), i \in [1, n_O]. \end{aligned} \tag{4}$$

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

$$\mathbf{y}_i(k + \mathbf{r}_i) = F(\mathbf{x}(k)) + G(\mathbf{x}(k))\mathbf{u}(k), i \in [1, n_O], \tag{5}$$

where $F_i(\mathbf{x}_k) = h_i \left(f^{\mathbf{r}_i}(\mathbf{x}(k)) \right) : \mathbb{R}^{n_S} \to \mathbb{R}^{n_O}$ and $G_i(\mathbf{x}(k)) = \frac{\partial}{\partial \mathbf{u}(k)} \left[h_i \left(f^{\mathbf{r}_i - 1} \left(f(\mathbf{x}(k)) + g(\mathbf{x}(k))\mathbf{u}(k) \right) \right) \right] : \mathbb{R}^{n_S} \to \mathbb{R}^{n_O} \times \mathbb{R}^{n_I}$ are decoupling functions. Finally, the control law at time k to track the desired output of the system $\mathbf{y}^* \in \mathbb{R}^{n_O}$ can be written as in [19]:

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

Assumption 2. *The desired output of the dynamical system in (1) is available [20], i.e.:*

$$\forall k \quad \exists \mathbf{y}^*(k) \in \mathbb{R}^{n_O}. \tag{7}$$

Assumption 3. *The dynamical system in (1) is input-to-output stable [21], i.e.:*

$$\| \mathbf{y}(k) \| \leq \gamma \left(\| \mathbf{u}(k) \| \right) \quad \forall k, \tag{8}$$

where γ is a gain function.

If a precise model of the system exists, the inversion of the system can be computed. However, the system's parameters might be unknown and difficult to estimate (e.g., moments of inertia). Besides, these parameters might change during the operation of the system (e.g., mass). In addition, it is difficult to predict the external disturbance term (e.g., wind gust). Furthermore, measurements from the system might come from a noisy sensor (e.g., monocular camera). Therefore, the control law in (6) cannot always be calculated precisely, and for a modified system a new control law has to be computed.

A. Internal Uncertainties

Let us consider a general nonlinear multi-input multi-output system with internal uncertainties:

$$\begin{cases} \mathbf{x}(k+1) & = \tilde{f}(\mathbf{x}(k)) + \tilde{g}(\mathbf{x}(k))\mathbf{u}(k) \\ \mathbf{y}(k) & = \tilde{h}(\mathbf{x}(k)), \end{cases} \tag{9}$$

where $\tilde{f} : \mathbb{R}^{n_S} \to \mathbb{R}^{n_S}$, $\tilde{g} : \mathbb{R}^{n_S} \to \mathbb{R}^{n_S} \times \mathbb{R}^{n_I}$ and $\tilde{h} : \mathbb{R}^{n_S} \to \mathbb{R}^{n_O}$ are new system functions. The control law at time step k to track the desired output $\mathbf{y}^*$ can be written as:

$$\mathbf{u}_i(k) = \left[\tilde{G}(\mathbf{x}(k)) \right]^{-1} \left(\mathbf{y}^*(k + \mathbf{r}_i) - \tilde{F}(\mathbf{x}(k)) \right), \tag{10}$$

where $\tilde{F}_i(\mathbf{x}_k) = \tilde{h}_i \left(\tilde{f}^{\mathbf{r}_i}(\mathbf{x}(k)) \right) : \mathbb{R}^{n_S} \to \mathbb{R}^{n_O}$ and $\tilde{G}_i(\mathbf{x}(k)) = \frac{\partial}{\partial \mathbf{u}(k)} \left[\tilde{h}_i \left(\tilde{f}^{\mathbf{r}_i - 1} \left(\tilde{f}(\mathbf{x}(k)) + \tilde{g}(\mathbf{x}(k))\mathbf{u}(k) \right) \right) \right] : \mathbb{R}^{n_S} \to \mathbb{R}^{n_O} \times \mathbb{R}^{n_I}$ are new decoupling functions. Therefore, for an exact tracking of $\mathbf{y}^*$, the exact values of new system functions $\tilde{f}$, $\tilde{g}$ and $\tilde{h}$ have to be known.

B. External Disturbance

Let us consider a general nonlinear multi-input multi-output system with external disturbances:

$$\begin{cases} \mathbf{x}(k+1) & = f(\mathbf{x}(k)) + g(\mathbf{x}(k))\mathbf{u}(k) + \mathbf{d}(k) \\ \mathbf{y}(k) & = h(\mathbf{x}(k)), \end{cases} \tag{11}$$

where $\mathbf{d} \in \mathbb{R}^{n_S}$ is the disturbance to the system. The control law at time step k to track the desired output $\mathbf{y}^*$ is:

$$\mathbf{u}_i(k) = \left[G(\mathbf{x}(k)) \right]^{-1} \left(\mathbf{y}^*(k + \mathbf{r}_i) - F(\mathbf{x}(k) - \mathbf{D}(k)) \right), \tag{12}$$

where $\mathbf{D}(k) = h_i \left(f^{\mathbf{r}_i - 1}(\mathbf{d}(k)) \right) \in \mathbb{R}^{n_O}$ is the disturbance decoupling matrix. Therefore, for an exact tracking of $\mathbf{y}^*$, the exact value of the disturbance $\mathbf{d}(k)$ has to be known.

C. Noisy Measurement

Let us consider a general nonlinear multi-input multi-output system with noisy measurements:

$$\begin{cases} \mathbf{x}(k+1) & = f(\mathbf{x}(k)) + g(\mathbf{x}(k))\mathbf{u}(k) \\ \mathbf{y}(k) & = h(\mathbf{x}(k)) + \mathcal{N}(k), \end{cases} \tag{13}$$

where $\mathcal{N} : \mathbb{R}^2 \to \mathbb{R}^{n_O}$ is an additive noise, e.g., additive white Gaussian noise, at time step k. The control law at time step k to track the desired output $\mathbf{y}^*$ can be written as:

$$\mathbf{u}_i(k) = \left[G(\mathbf{x}(k)) \right]^{-1} \left(\mathbf{y}^*(k + \mathbf{r}_i) - \mathcal{N}(k + \mathbf{r}_i) - F(\mathbf{x}(k)) \right). \tag{14}$$

Therefore, for an exact tracking of $\mathbf{y}^*$, the exact model of the noise $\mathcal{N}$ has to be known.

III. Deep Fuzzy Neural Network

To deal with the problems described in Section II, an adaptive controller which can learn system dynamics online and deal with uncertainties is required. DNNs are able to learn from input-output data, whereas fuzzy logic has an exceptional ability to handle noise and uncertainties. The DNN with one antecedent fuzzification layer, also called DFNN, can be used for learning control of nonlinear systems. The DFNN neurons are organised in input layer with N_I neurons, Gaussian fuzzification layer with N_F neurons, N_L hidden layers with (N_H+1) neurons in each layer, and output layer with N_O neurons, as shown in Fig 1. The input $\{\mathcal{I}_1,\ldots,\mathcal{I}_{N_I}\}$ to DFNN is fuzzified by three Gaussian membership functions (MFs):

$$\mu_{F^k}(\mathcal{I}_j)=\exp\left[-\frac{(\mathcal{I}_j-c_{F,k})^2}{2\sigma_{F,k}^2}\right],\quad \begin{matrix} j=1,\ldots,N_I \\ k=1,2,3, \end{matrix} \tag{15}$$

where $c_{F,1}=-1$, $c_{F,2}=0$, $c_{F,3}=1$, and $\sigma_{F,1}=\sigma_{F,1}=\sigma_{F,1}=1$, as depicted in Fig. 2. Then, the fuzzified input $\{\mu_{F^1}(\mathcal{I}_1),\ \mu_{F^2}(\mathcal{I}_1),\ \mu_{F^3}(\mathcal{I}_1),\ \ldots,\ \mu_{F^1}(\mathcal{I}_{N_I}),\ \mu_{F^2}(\mathcal{I}_{N_I}),\ \mu_{F^3}(\mathcal{I}_{N_I})\}$ is fed into the first hidden layer of DFNN through the network weights $\mathbf{W}_1$. The hidden layers in DFNN are organized in a fully connected structure with the network weights $\mathbf{W}_i$, $i=2,\ldots,N_L-1$. Finally, the output $\{\mathcal{O}_1,\ldots,\mathcal{O}_{N_O}\}$ is computed by applying the network weights $\mathbf{W}_{N_L}$ to the output from the last hidden layer.

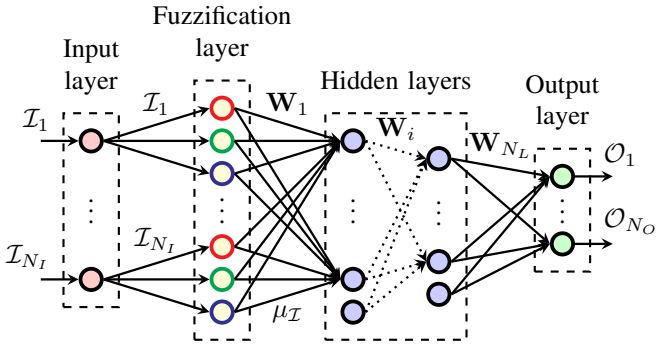


Fig. 1. Structure of DFNN organized in input layer with N_I neurons, Gaussian fuzzification layer with N_F neurons, N_L fully-connected hidden layers with (N_H+1) neurons in each layer and output layer with N_O neurons.

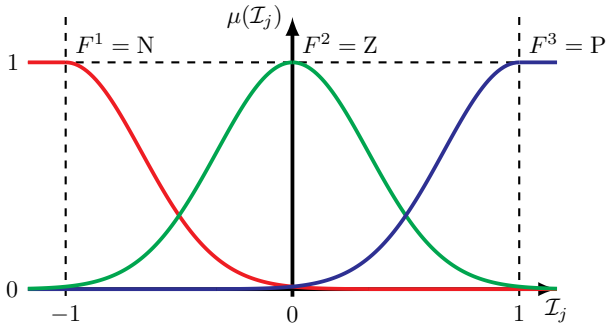


Fig. 2. "Negative" (N), "zero" (Z) and "positive" (P) FSs represented by three Gaussian MFs.

Assumption 4. *The network weights $\mathbf{W}_i$, $i=1,\ldots,N_L$, are bounded, i.e.:*

$$\|\mathbf{W}_i(k)\|_\infty \le c_{\mathbf{W},i}\quad \forall k\quad i=1,\ldots,N_L, \tag{16}$$

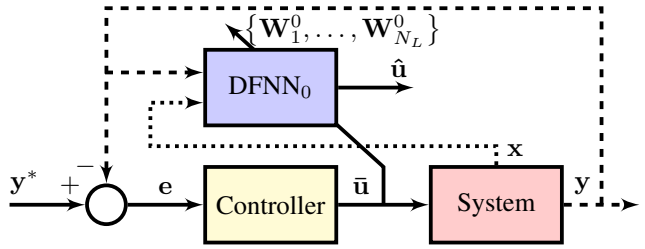
where $c_{\mathbf{W},i}$, $i=1,\ldots,N_L$, are some positive constants.

The learning is subdivided into two phases: offline pre-training and online training [22], as illustrated in Fig. 3. During the offline pre-training phase, a conventional controller performs a set of trajectories and the batch of training samples is collected. Then, DFNN-based controller, called DFNN$_0$, is pre-trained on the collected data samples, to approximate the inverse of the system's dynamics. However, DFNN$_0$ cannot adapt to the new conditions; therefore, the online training is required. During the online learning phase, DFNN controls the system and adapts the control input to improve performances. The expert knowledge encoded into the rule-base, thanks to the fuzzy mapping, provides the adaptation information to DFNN. The approximation of the inverse of the system dynamics is a typical regression problem; therefore, the cost function for both offline and online training is the mean squared error.

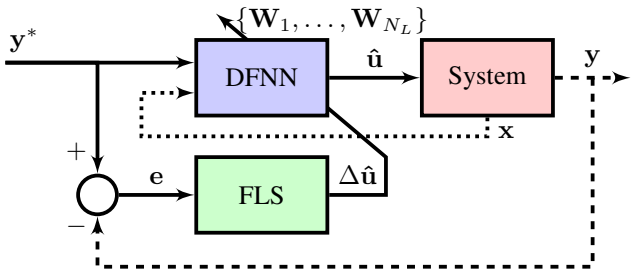
A. Offline Pre-Training

During the offline pre-training phase, a feed-forward DFNN$_0$ learns the approximate inverse dynamics of the system by adjusting the weights $\{\mathbf{W}_1^0,\ldots,\mathbf{W}_{N_L}^0\}$ in the network. In this control scheme, shown in Fig. 3a, a conventional controller, e.g., proportional-integral-derivative (PID) controller, controls the system alone. Hence, it is utilized as an ordinary feedback controller to provide labelled training samples for DFNN$_0$. Each labelled training sample $\mathcal{D}_0(k)$ consists of input $\mathcal{I}_0(k)$ and expected output $\mathcal{O}_0(k)$ pair:

$$\begin{aligned}\mathcal{D}_0(k) &=<\mathcal{I}_0(k),\mathcal{O}_0(k)>\\ &=<\{\mathbf{x}(k-\bar{r}),\mathbf{y}(k)\},\{\bar{\mathbf{u}}(k-\bar{r})\}>.\end{aligned} \tag{17}$$



(a) Offline pre-training of DFNN by conventional controller.



(b) Online training of DFNN by FLS.

Fig. 3. Block diagrams of the two control paradigms: offline pre-training and online training (solid lines represent calculated quantities, dashed lines represent measured quantities, dotted lines represent estimated quantities).

where $\bar{r} = \max_{\forall i \in [1, n_O]} \mathbf{r}_i$. The training of DFNN$_0$ involves back-propagation to minimize the loss over all training examples, and the network weights $\{\mathbf{W}_1^0, \ldots, \mathbf{W}_{N_L}^0\}$ are updated until the over-fitting appears. After the training, DFNN$_0$ can approximate the inverse dynamics of the nominal system. The pseudo-code of the offline pre-training is given in Algorithm 1.

B. Online Training

During the online training phase, DFNN controls the system and, at the same time, learns how to improve the performances. Since DFNN training requires supervised learning, another process has to provide feedback about its performances. In our method, FLS is used to monitor the system. By its nature, FLS incorporates the expert knowledge in the form of rules, and uses this knowledge to provide some useful advices [23]. The control scheme of the online training is shown in Fig. 3b.

The objective is to learn the control of the system by only looking at its performance, i.e., in our case, the tracking error:

$$\mathbf{e}(k) = \mathbf{y}^*(k) - \mathbf{y}(k), \tag{18}$$

and its time derivative:

$$\dot{\mathbf{e}}(k) = \dot{\mathbf{y}}^*(k) - \dot{\mathbf{y}}(k), \tag{19}$$

where $\dot{\mathbf{y}} = \frac{\partial \mathbf{y}}{\partial k} \in \mathbb{R}^{n_O}$ is the time derivative of the output from the system, and $\dot{\mathbf{y}}^* = \frac{\partial \mathbf{y}^*}{\partial k} \in \mathbb{R}^{n_O}$ is the time derivative of the desired output.

In our approach, FLS observes the behaviour of the system controlled by DFNN, and, depending on the situation, corrects the action of DFNN. The possible evolutions of the error are depicted in Fig. 4. For example, if the error is positive, i.e., $e_j(k) > 0$, and its time derivative is also positive, i.e., $\dot{e}_j(k) > 0$, then the system diverges (top red curve in Fig. 4). In this case, FLS will force DFNN to decrease the control signal $u_j(k)$ significantly to stabilize the system, i.e., $\Delta u_j(k) \ll 0$. In another possible case, if the error is negative, i.e., $e_j(k) < 0$, and its time derivative is zero, i.e., $\dot{e}_j(k) = 0$, then the error is steady (bottom purple line in Fig. 4). In this case, DFNN falls down in a local minimum and FLS will give a small positive perturbation, i.e., $\Delta u_j(k) > 0$. Finally, if the error is zero, i.e., $e_j(k) = 0$, and its time derivative is also zero, i.e., $\dot{e}_j(k) = 0$, then, this is the optimal case (green line in Fig. 4) and no action has to be taken, i.e., $\Delta u_j(k) = 0$.

Algorithm 1: Offline pre-training of DFNN.

Input: -
Output: Pre-trained DFNN$_0$
begin
  while $k <$ *MaxSamples* **do**
    Get $\mathbf{x}(k - \bar{r})$, $\mathbf{y}(k)$, and $\bar{\mathbf{u}}(k - \bar{r})$
    Collect $\mathcal{D}_0(k)$ in (17)
  end
  DFNN$_0 \leftarrow$ ConstructNetworkLayers()
  $\{\mathbf{W}_1^0, \ldots, \mathbf{W}_{N_L}^0\} \leftarrow$ InitializeWeights()
  Train DFNN$_0$ on $\mathcal{D}_0$
end

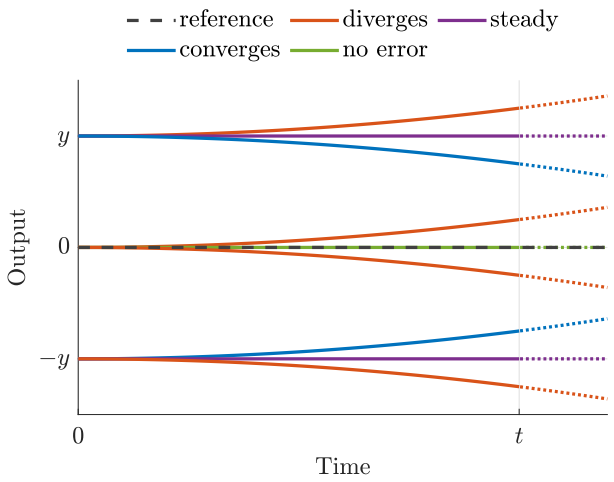


Fig. 4. Possible evolution of the controlled dynamical system. The system can diverge (red curves), converge (blue curves), it can have a steady error (purple lines), or the error can be zero (green line). In the proposed approach, FLC observes the time evolution of the output from the system controlled by DFNN and, depending on the situation, corrects the action of DFNN.

All these intuitive rules can be formally described by a Mamdani FLS. If N is the number of possible cases, for each case, one rule R_i, $i \in [1, N]$, exists in the rule-base in Table I, where the colour of the cell corresponds to the case in Fig. 4. The inputs to the FLC are selected to be the tracking error and its time derivative, i.e., $e_j(k)$ and $\dot{e}_j(k)$; while the output is the correction signal, i.e., $\Delta u_j(k)$. The input is represented by three fuzzy sets (FSs): negative, zero and positive; while the output can belong to five FSs: big decrease, small decrease, no changes, small increase and big increase.

By their nature, FLS requires operations among FSs which are computationally expensive. Since in a conventional FLS, first, the input is fuzzified, then, it goes through the inference engine and, in the end, it is defuzzified. However, an analitical relation between the input to FLS and its output, called fuzzy mapping [24], reduces significantly the computation time which allows the application of FLSs for real-time control applications. By using a similar approach in [25], a fuzzy mapping which represents FLS described in Table I can be generated for a multidimensional case.

By using the centroid defuzzification, the defuzzified output from FLS is as in [26]:

$$\varphi(\boldsymbol{\sigma}) = \frac{\sum_{i=1}^{N} f_i(\boldsymbol{\sigma}) C_i}{\sum_{i=1}^{N} f_i(\boldsymbol{\sigma})}, \tag{20}$$

TABLE I
RULE-BASE FOR THE UPDATES OF $u_j(k)$.

$e_j(k)$	$\dot{e}_j(k)$		
	Negative	**Zero**	**Positive**
Negative	R_1 : Big decrease	R_2 : Big increase	R_3 : Small increase
Zero	R_4 : Small decrease	R_5 : No changes	R_6 : Small decrease
Positive	R_7 : Small increase	R_8 : Big increase	R_9 : Big decrease

where $\boldsymbol{\sigma}$ is the vector of crisp inputs, C_i is the consequent FS of the i-th rule, and $f_i(\boldsymbol{\sigma}) = \prod_{j=1}^{M} \mu_{A_{j,i}}(\sigma_j) \in [0,1] \quad \forall i \in [1, N]$ is the firing strength of the i-th rule computed with the product t-norm, in which M is the number of inputs, and $\mu_{A_{j,i}}$ is MF describing FS A_j of the j-th crisp input. In our case, the number of inputs is two and Table I contains nine rules; thus, (20) becomes:

$$\varphi(\sigma_1, \sigma_2) = \frac{\sum_{i=1}^{9} \left(\mu_{A_{1,i}}(\sigma_1) \times \mu_{A_{2,i}}(\sigma_2)\right) C_i}{\sum_{i=1}^{9} \mu_{A_{1,i}}(\sigma_1) \times \mu_{A_{2,i}}(\sigma_2)}. \tag{21}$$

Then, the antecedent MFs are selected to be: $\mu_{A_{1,1}} = \mu_{A_{1,2}} = \mu_{A_{1,3}} = \mu_{A_{2,1}} = \mu_{A_{2,4}} = \mu_{A_{2,7}} = \mu_{A^1}$, $\mu_{A_{1,4}} = \mu_{A_{1,5}} = \mu_{A_{1,6}} = \mu_{A_{2,2}} = \mu_{A_{2,5}} = \mu_{A_{2,8}} = \mu_{A^2}$ and $\mu_{A_{1,7}} = \mu_{A_{1,8}} = \mu_{A_{1,9}} = \mu_{A_{2,3}} = \mu_{A_{2,6}} = \mu_{A_{2,9}} = \mu_{A^3} = -\mu_{A^1}$, which are defined as triangular MFs:

$$\mu_{A^k}(\sigma_j) = \begin{cases} 0 & , \sigma_j < a_{k-1} \\ \frac{\sigma_j - a_{k-1}}{a_k - a_{k-1}} & , a_{k-1} \leq \sigma_j \leq a_k \\ \frac{a_{k+1} - \sigma_j}{a_{k+1} - a_k} & , a_k < \sigma_j \leq a_{k+1} \\ 0 & , \sigma_j > a_{k+1}, \end{cases} \quad \begin{cases} k = 1, 2, 3 \\ j = 1, 2, \end{cases} \tag{22}$$

where $a_0 = -\infty$, $a_1 = -1$, $a_2 = 0$, $a_3 = 1$ and $a_4 = +\infty$, as illustrated in Fig. 5a. The consequent FSs are selected to be singleton: $C_1 = C_9 = C^1 = -1$, $C_4 = C_6 = C^2 = -0.5$, $C_5 = C^3 = 0$, $C_3 = C_7 = C^4 = 0.5$ and $C_2 = C_8 = C^5 = 1$, as illustrated in Fig. 5b. Hence, after performing some simplifications in (21), the fuzzy mapping is computed:

$$\varphi(\sigma_1, \sigma_2) = |\sigma_1| - \frac{|\sigma_2|}{2} - \frac{3\,|\sigma_1 \sigma_2|}{4} - \frac{3\sigma_1 \sigma_2}{4}. \tag{23}$$

This expression describes FLS in Table I in an analytical form. Finally, (23) in its multidimensional form can be used to update the control signal:

$$\Delta\hat{\mathbf{u}}(k) = \alpha\varphi\left(\mathbf{e}(k), \dot{\mathbf{e}}(k)\right), \tag{24}$$

where $\alpha > 0$ is a scaling factor.

Remark 1. *A large α allows to learn faster at the cost of possible divergence or oscillation of the controlled system. A smaller α may allow to learn in a safer and more conservative way but may make the learning significantly longer.*

Remark 2. *If $\mathbf{e}(k)$ and $\mathbf{e}(k-1)$ are approximately 0, i.e., $\mathbf{e}(k) \approx \mathbf{0} \wedge \mathbf{e}(k-1) \approx \mathbf{0}$; then, $\dot{\mathbf{e}}(k)$ approaches 0, i.e., $\dot{\mathbf{e}}(k) \to \mathbf{0}$. Consequently, from (24) and (23), $\Delta\hat{\mathbf{u}}$ is asymptotically equivalent to 0, i.e., $\Delta\hat{\mathbf{u}}(k) \sim \mathbf{0}$. Therefore, the weights are not updated, and the convergence condition is reached.*

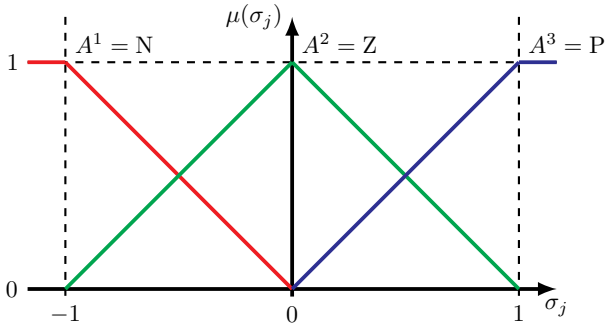
Each labelled training sample $\mathcal{D}(k)$ consists of input $\mathcal{I}(k)$ and corrected output $\mathcal{O}(k)$ pair:

$$\begin{aligned} \mathcal{D}(k) &=< \mathcal{I}(k), \mathcal{O}(k) > \\ &=< \{\mathbf{x}(k), \mathbf{y}^*(k+\bar{r})\}, \{\hat{\mathbf{u}}(k) + (\Delta\hat{\mathbf{u}}(k))\,\hat{\mathbf{u}}(k)\} > . \end{aligned} \tag{25}$$

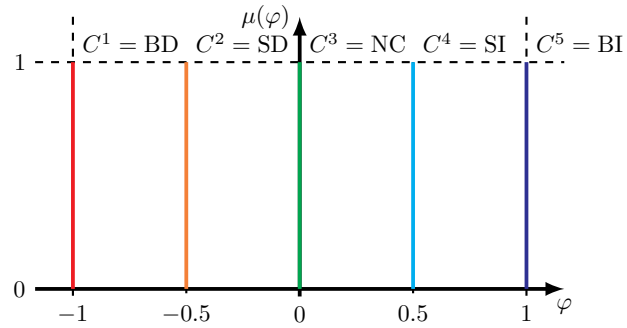
At each iteration, DFNN is adapted with this training sample, and the new output from the network is computed:

$$\hat{\mathbf{u}}(k) = \text{DFNN}(\mathcal{I}(k)). \tag{26}$$

The pseudo-code of online training is provided in Algorithm 2.



(a) "Negative" (N), "zero" (Z) and "positive" (P) FSs represented by three triangular MFs.



(b) "Big decrease" (BD), "small decrease" (SD), "no change" (NC), "small increase" (SI) and "big increase" (BI) FSs represented by five singleton MFs.

Fig. 5. Three triangular antecedence and five singleton consequent MFs for the update of the control signal.

C. Stability Analysis

For the system in (1), a necessary condition for the stability of the inverse dynamics, and hence for the effectiveness of the DFNN-based approach, is the stability of the zero dynamics of the system [27].

Theorem 1. *Consider the system in (1) and the control signal $\hat{\mathbf{u}}(k)$ in (26). The overall closed-loop system is bounded-input-bounded-state (BIBO) stable iff Assumptions 1–4 are verified.*

Proof. For the proof, please refer to Appendix A. □

Algorithm 2: Online training of DFNN.

Input: Pre-trained DFNN_0
Output: Trained DFNN
begin
  DFNN $\leftarrow$ DFNN_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 (18)
    $\dot{\mathbf{e}}(k) \leftarrow \dot{\mathbf{y}}^*(k) - \dot{\mathbf{y}}(k)$ by using (19)
    $\Delta\hat{\mathbf{u}}(k) \leftarrow \alpha\text{FLS}(\mathbf{e}(k), \dot{\mathbf{e}}(k))$ by using (24)
    $\hat{\mathbf{u}}(k) \leftarrow \text{DFNN}(\mathcal{I}(k))$ by using (26)
    Adapt DFNN with $\mathcal{D}(k)$ by using (25)
    Send $\mathbf{u}(k)$ to the system
  until *Stop*
end

IV. Simulation Study

In this section, the performances of the proposed approach in Section III are tested on four different single-input-single-output systems: nominal system, system with internal uncertainties, system with external disturbance, and system with noisy measurements. The desired trajectory is:

$$y^*(k) = \frac{1}{3}\sin(3\pi k) + \frac{1}{2}\cos(2\pi k) - 1. \tag{27}$$

1) Nominal system: The nominal nonlinear system is selected as:

$$\begin{cases} \mathbf{x}(k+1) & = \begin{bmatrix} x_2 \\ x_1 - x_1^2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k) \\ y(k) & = -0.2x_1 + x_2. \end{cases} \tag{28}$$

2) Internal uncertainties in the system: The nominal system in (28) with internal uncertainties is:

$$\begin{cases} \mathbf{x}(k+1) & = \begin{bmatrix} 0.5x_2 \\ 0.5x_1 - 0.5x_1^2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} u(k) \\ y(k) & = -0.1x_1 + 0.5x_2. \end{cases} \tag{29}$$

3) External disturbance to the system: The nominal system in (28) with external variable disturbance is:

$$\begin{cases} \mathbf{x}(k+1) & = \begin{bmatrix} x_2 \\ x_1 - x_1^2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k) + \begin{bmatrix} 0 \\ \cos(2k) \end{bmatrix} \\ y(k) & = -0.2x_1 + x_2. \end{cases} \tag{30}$$

4) Noisy measurements from the system: The nominal system in (28) with noisy measurements is:

$$\begin{cases} \mathbf{x}(k+1) & = \begin{bmatrix} x_2 \\ x_1 - x_1^3 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k) \\ y(k) & = -0.2x_1 + x_2 + \mathcal{N}(0, y(k-1)). \end{cases} \tag{31}$$

A. Control Approach

Throughout the simulations in this study, we assume the systems are black-boxes, and we use only input-output data and some basic properties such as the relative degree. It is possible to verify with (2) that the relative degree $\bar{r} = 1$ for the systems in (28)–(31). A feed-forward DFNN with hyperbolic tangent ($\tanh$) activation functions is designed. According to (17) and (25), DFNN has three inputs ($N_I = 3$) and one output ($N_O = 1$). In addition, after some heuristic analysis and experimental trials, the architecture of the network is chosen to consist of one fuzzification layer with nine neurons ($N_F = 9$), and two fully connected hidden layers ($N_L = 2$) with 16 neurons ($N_H = 16$) in each layer. The scaling factor $\alpha = 0.1$ in (24), for all cases.

To collect the training data for the offline pre-training, the nominal system is controlled by PID controller which has been tuned by trial-and-error. Thus, $2'000$ input-output training pairs $\mathcal{D}_0(k) =< \{x_1(k-1), x_2(k-1), y(k)\}, \{u(k-1)\} >$ are saved. Then, DFNN$_0$ is trained on $\mathcal{D}_0$ by using Levenberg–Marquardt algorithm. After that, the pre-trained DFNN controls the system online and, at each iteration, it is updated on $\mathcal{D}(k) =< \{x_1(k), x_2(k), y^*(k+1)\}, \{u(k) + \Delta u(k)\} >$.

B. Results

To show the efficiency of the proposed approach, the performances of the developed DFNN with online learning are compared with the performances of the exact analytical inverse of the system dynamics, PID controller, type-1 FNN-based (T1-FNN) controller with Gaussian antecedent MFs and Levenberg-Marquardt update rules from [15], and interval type-2 FNN-based (IT2-FNN) controller with elliptic antecedent MFs and sliding mode control adaptation laws from [28]. As can be seen from Figs. 6–9, the exact analytical system inverse is able to track perfectly the reference trajectory. However, in real-world it is difficult, and sometimes even impossible, to calculate the exact inverse dynamics of the system. From Fig. 6, it is possible to observe that both DFNN$_0$ and DFNN with online learning are able to track precisely the desired trajectory of the nominal system. In addition, DFNN$_0$ approximates very accurately the exact inverse of the system in (6). From Fig. 7, it is possible to observe that DFNN$_0$ is not able to track the desired trajectory on the system with modified dynamics, and its performances are worst than the ones of PID, T1-FNN and IT2-FNN controllers; while DFNN with online learning is able to learn the new system dynamics and obtain a good performance. From Fig. 8, it is possible to observe that PID, T1-FNN and DFNN$_0$ become unstable with time-varying disturbance; while DFNN with online learning is able to learn new conditions and obtain a good performance. From Fig. 9, it is possible to observe that PID controller is not able to deal with this level of noise; while all T1-FNN, IT2-FNN, DFNN$_0$ and DFNN are able to control the system.

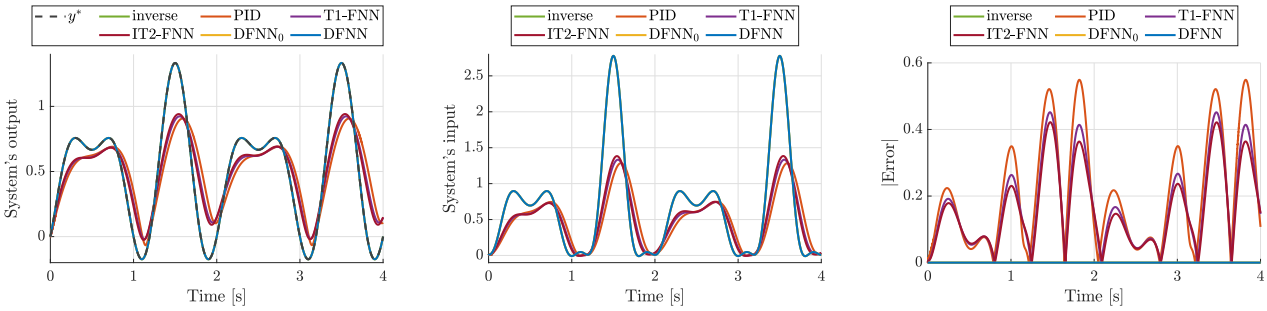
As can be seen from Table II, the exact analytical system inverse is able to track the reference trajectory with negligible error. On the other hand, the DFNN-based controller with online learning outperforms all PID, T1-FNN, IT2-FNN and DFNN$_0$ for all tested cases in terms of mean absolute error (MAE). Only in the case with noisy measurements, DFNN with online learning tries to learn also the noise which makes its performances worst than DFNN$_0$.

V. Experimental Study

To validate the capabilities of the proposed controller in Section III, the trajectory following problem of a quadcopter UAV is considered. The experimental platform used in this work is Parrot Bebop 2 quadcopter UAV as shown in Fig. 10. Robot operating system (ROS) is used to communicate with UAV. The visual-inertial odometry algorithm is used to provide the UAV's real-time position at 24Hz. This information is fed

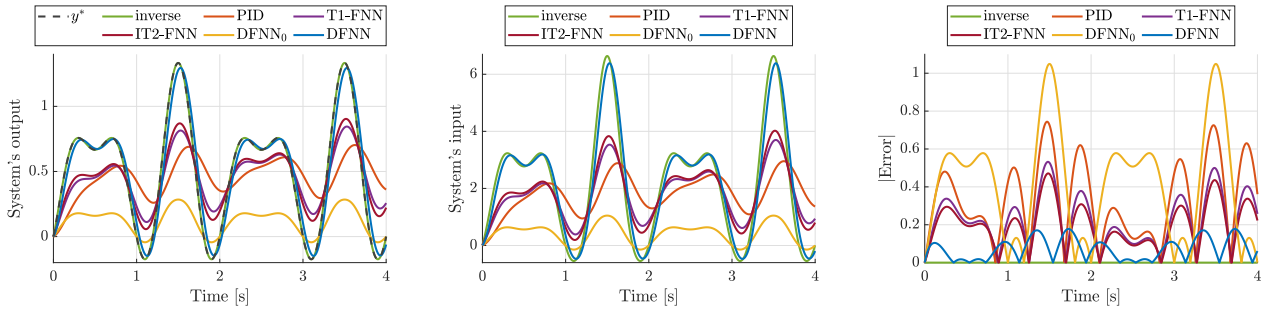
TABLE II
Comparison of different controllers in terms of MAE [m].

Controller	System in (28)	System in (29)	System in (30)	System in (31)
Inverse	~ 0	~ 0	~ 0	~ 0
PID	0.265	0.386	∞	∞
T1-FNN	0.216	0.261	∞	0.225
IT2-FNN	0.196	0.225	0.223	0.204
DFNN$_0$	7.53×10^{-6}	0.505	∞	0.097
DFNN	1.01×10^{-6}	0.090	0.038	0.100



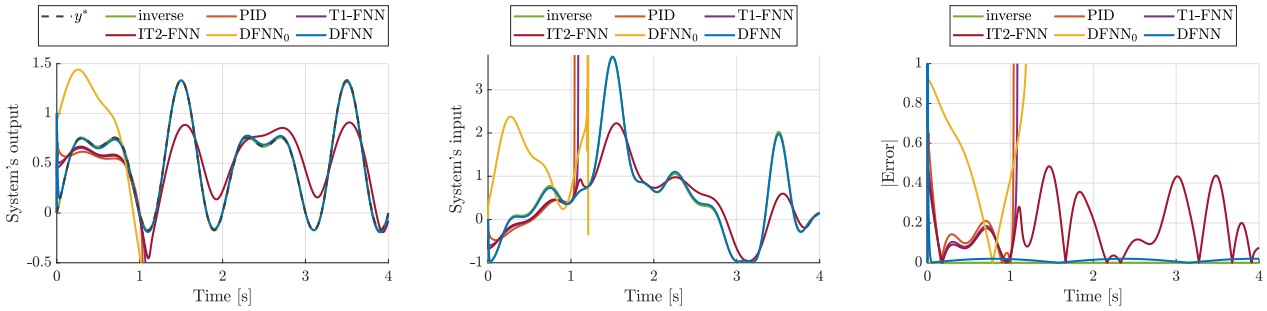
(a) Tracking performance. (b) Control signal. (c) Tracking error.

Fig. 6. Simulation results for different controllers on the nominal system in (28). It is possible to observe that both DFNN$_0$ and DFNN with online learning are able to track precisely the desired trajectory of the nominal system. In addition, DFNN$_0$ approximates accurately the exact inverse of the system in (6).



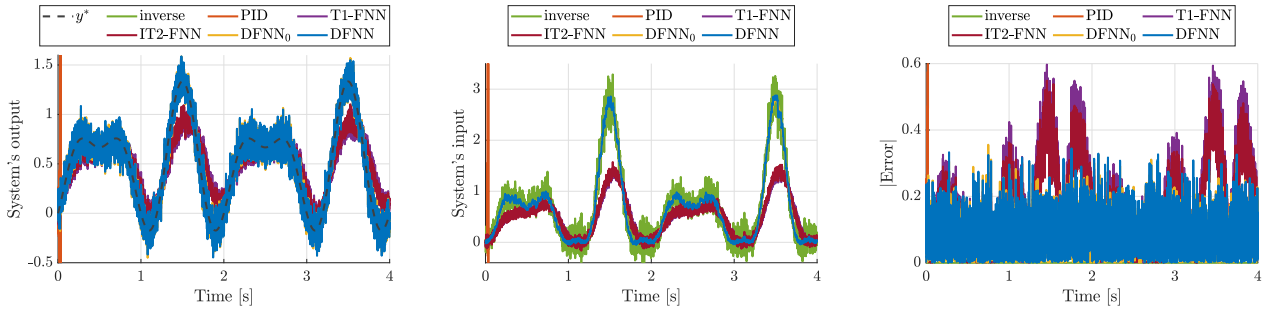
(a) Tracking performance. (b) Control signal. (c) Tracking error.

Fig. 7. Simulation results for different controllers on the nominal system with internal uncertainties in (29). DFNN$_0$ is not able to track the desired trajectory on the system with modified dynamics. While DFNN with online learning is able to learn the new system dynamics and obtain a good performances.



(a) Simulation performance. (b) Control signal. (c) Tracking error.

Fig. 8. Simulation results for different controllers on the nominal system with external time-varying disturbance in (30). All PID, T1-FNN and DFNN$_0$ become unstable with time-varying disturbance. While DFNN with online learning is able to learn the new conditions and obtain good performances.



(a) Tracking performance. (b) Control signal. (c) Tracking error.

Fig. 9. Simulation results for different controllers on the nominal system with noisy measurements in (31). It is possible to observe that PID controller is not able to deal with this level of noise. While all T1-FNN, IT2-FNN DFNN$_0$ and DFNN are able to control the system.

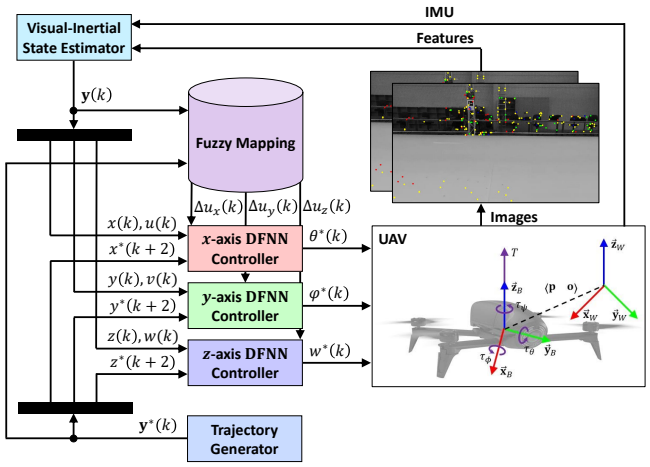


Fig. 10. Illustration of the proposed online learning-based control scheme. During the online learning phase, DFNN controls the UAV and, by using the expert knowledge encoded into the fuzzy mapping, improves further the performance in real-time.

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.

A. Dynamical Model of Unmanned Aerial Vehicle

The world-fixed frame is $\mathcal{F}_W = \{\vec{\mathbf{x}}_W, \vec{\mathbf{y}}_W, \vec{\mathbf{z}}_W\}$ and the body frame is $\mathcal{F}_B = \{\vec{\mathbf{x}}_B, \vec{\mathbf{y}}_B, \vec{\mathbf{z}}_B\}$, illustrated in Fig. 10. 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 mass in $\mathcal{F}_W$, and its attitude $\mathbf{o} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$ is given by three Euler angles. The time derivative of the position gives the linear velocity $\mathbf{v} = \begin{bmatrix} u & v & w \end{bmatrix}^T$ of UAV expressed in $\mathcal{F}_W$. Equivalently, the time derivative of the attitude gives angular velocity $\boldsymbol{\omega} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$ in $\mathcal{F}_W$ and angular rates $\boldsymbol{\omega}_B = \begin{bmatrix} p & q & r \end{bmatrix}^T$ in $\mathcal{F}_B$. The UAV is controlled by adjusting the total thrust T along $\vec{\mathbf{z}}_B$, and the moments τ_ϕ, τ_θ and τ_ψ on UAV's center of mass around $\vec{\mathbf{x}}_B$, $\vec{\mathbf{y}}_B$ and $\vec{\mathbf{z}}_B$, respectively.

The dynamical model of UAV is provided in [29], and it can be rewritten in form of (1), where $\mathbf{x} = \begin{bmatrix} x & y & z & \phi & \theta & \psi & u & v & w & p & q & r \end{bmatrix}^T$, $\mathbf{u} = \begin{bmatrix} \tau_\phi & \tau_\theta & T & \tau_\psi \end{bmatrix}^T$, $\mathbf{y} = \mathbf{x}$, $f(\mathbf{x})$ and $g(\mathbf{x})$ are defined in (32) and (33), respectively [30]. If the attitude controller as in [31] is included in the dynamical model, then the virtual control inputs are $\mathbf{u} = \begin{bmatrix} \phi & \theta & w & r \end{bmatrix}^T$.

Remark 3. *The dynamical system described by (32) and (33) is nonlinear, coupled, underactuated and open-loop unstable.*

B. Monocular Visual-Inertial Localization

Monocular keyframe-based visual simultaneous localization and mapping (SLAM) has become a key technology for localization of different types of robots, especially for UAVs. In the literature, the most representative monocular keyframe-based visual-inertial SLAM approach is feature-based parallel tracking and mapping [32]. It has been demonstrated to be successful in different real-time applications [33].

C. Control Approach

The dynamical system of UAV is subdivided into three simpler subsystems to reduce the complexity and accelerate the learning process. Three feed-forward DFNNs are used to learn the control mapping for each controlled axis: x, y and z, as depicted in Fig. 10. From the dynamical model of UAV, it is possible to calculate that the relative degree $\bar{r} = 2$. According to (17) and (25), each DFNN has three inputs ($N_I = 3$) and one output ($N_O = 1$). In addition, after some heuristic analysis and experimental trials, the architecture of the network is chosen to consist of one fuzzification layer with nine neurons ($N_F = 9$), and two fully connected hidden layers ($N_L = 2$) with 64 neurons ($N_H = 64$) in each layer and with hyperbolic tangent ($\tanh$) activation functions.

The inputs to DFNN for the x-axis are the state components relative to the x-axis, $\{x(k), u(k), x^*(k+2)\}$, and the output is the desired pitch angle, $\{\theta^*(k)\}$. Similarly, the inputs to DFNN for the y-axis are the state components relative to the y-axis, $\{y(k), v(k), y^*(k+2)\}$, and the output is the desired roll angle, $\{\phi^*(k)\}$. Finally, the inputs to DFNN for the z-axis are the errors and their time derivatives on the z-axis, $\{z(k), w(k), z^*(k+2)\}$, and the output is the desired vertical velocity, $\{w^*(k)\}$.

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. The scaling factor $\alpha = 0.1$ in (24).

Remark 4. *Both DFNN controllers with and without online learning consist of three independent and parallel subnetworks for x, y and z axes to speed up the learning.*

$$f(\mathbf{x}) = \begin{bmatrix} u & v & w & p + s_\phi t_\theta q + c_\phi t_\theta r & c_\phi q - s_\phi r & \frac{s_\phi}{c_\theta} q + \frac{c_\phi}{c_\theta} r & 0 & 0 & g & \frac{I_y - I_z}{I_x} qr & \frac{I_z - I_x}{I_y} pr & \frac{I_x - I_y}{I_z} pq \end{bmatrix}^T. \tag{32}$$

$$g(\mathbf{x}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{m}\left(c_\phi c_\psi s_\theta + s_\phi s_\psi\right) & -\frac{1}{m}\left(c_\phi s_\psi s_\theta - c_\psi s_\phi\right) & -\frac{1}{m} c_\phi c_\theta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix}^T. \tag{33}$$

To prepare the training samples of the flight data, the system was controlled by a conventional controller alone, while the current state, desired state, and 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 circular and eight-shaped trajectories on xy-, xz- and yz-planes with the reference speed of $1\mathrm{m/s}$.

D. Study Cases

Four circular trajectories with different working conditions have been tested: slow, fast, near-ground and with payload. The study cases are designed in a way to exploit different components of UAV dynamics. Furthermore, the visual-inertial algorithm for the state estimation produces noisy output [33]. In order to show the efficiency and efficacy of the DFNN-based controller with online learning, it is compared with a well-tuned PID controller (used for the collection of training samples), FNN-based controller with Gaussian antecedent MFs and Levenberg-Marquardt update rules from [15] and FNN-based controller with elliptic antecedent MFs and sliding mode control adaptation laws from [28], and DFNN_0 controller without online training.

Remark 5. *For real-time experiments, the inverse dynamics of the system is not used to control the system, since, in real-world it is difficult, and sometimes even impossible, to estimate the exact inverse dynamics of the system.*

The first study case is the tracking of the slow circular trajectory with a radius of $2\mathrm{m}$ on the xy-plane at a velocity of $1\mathrm{m/s}$ which has also been used during the pre-training phase. This case study is a reference example where UAV operates in its nominal conditions. The results of the trajectory tracking for this study case are shown in Fig. 11.

The second study case is the tracking of the fast circular trajectory with a radius of $2\mathrm{m}$ on the xy-plane at a velocity of $2\mathrm{m/s}$. In this study case, the fast responses of the controllers and the robustness of the visual-inertial state estimator to the motion blur are verified. The results of the trajectory tracking for this study case are shown in Fig. 12.

The third study case is the tracking of the circular trajectory, while flying at a height of $0.2\mathrm{m}$. In this study case, the ground effect generates an external disturbance on UAV. The results of the trajectory tracking for this study case are shown in Fig. 13.

The fourth study case is the tracking of the circular trajectory, while flying with a payload (Odroid-C2 on-board computer on top and office scissors attached to one of the arms) of $209\mathrm{g}$. In this study case, the dynamical model of UAV (mass and moments of inertia) is altered by the payload. The results of the trajectory tracking for this study case are shown in Fig. 14.

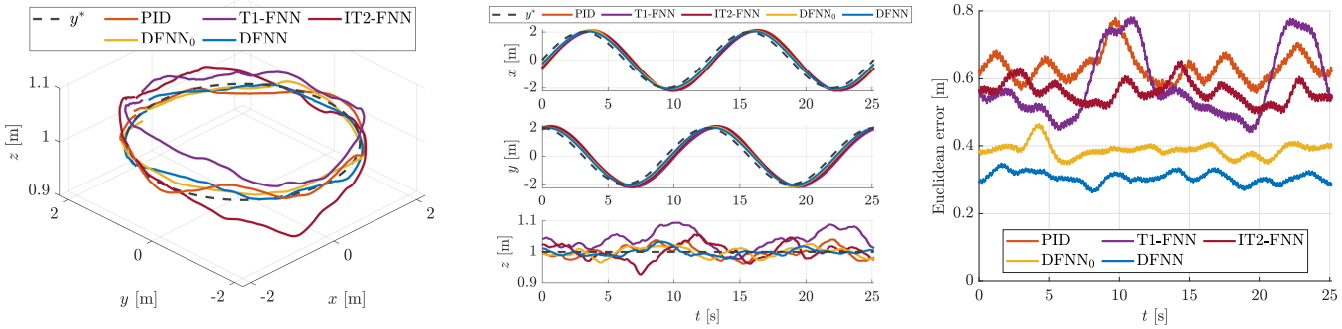
E. Discussion

A sample of experimental results for five controllers (PID, T1-FNN from [15], IT2-FNN from [28], DFNN_0 and DFNN) on four different circular trajectories (slow, fast, near-ground and with payload) are illustrated in Figs. 11–14, respectively. It is possible to observe from Figs. 11a–14a, that DFNN-based controller with online training is able to track more closely the 3D reference trajectory. As visualized from Figs. 11b–14b, DFNN-base controller has faster responses, since it is able to estimate the evolution of the system dynamics, and compute the desired control signal. It is possible to observe from Figs. 11c–14c, that DFNN-based controller with online training is able to learn the system dynamics and decrease the tracking error over time on all tested trajectories. The real-time experimental video is available at tiny.cc/DFNN.

For the statistical analysis of control performances, the experiments are repeated five times for each controller-case combination for a total of 100 experiments under quasi-same conditions. To compare the trajectory tracking performances, a box-plot is presented in Fig. 15. It is possible to observe that on average the DFNN-based controller with online learning outperforms other controllers on the tested trajectories. It has to be emphasised that DFNN evolves online from the pre-trained DFNN_0 during the learning process. Moreover, as expected, DFNN_0 has poor performances in the cases which have not been used for its training. It is also interesting to observe that the performances of PID controller do not get worse in case of near-ground and with payload trajectories, because derivative and integral components can compensate for these disturbances. Furthermore, the FNN-based controllers (T1-FNN and IT2-FNN) have similar performances for the slow, near-ground and with payload trajectories because their fast learning capabilities can compensate the disturbances coming from the ground effects and increased mass. The maximum absolute error is lower for the online DFNN-based controller, even for the cases unseen during the pre-training. Finally, DFNN-based controller with online learning has the lowest variance of the error.

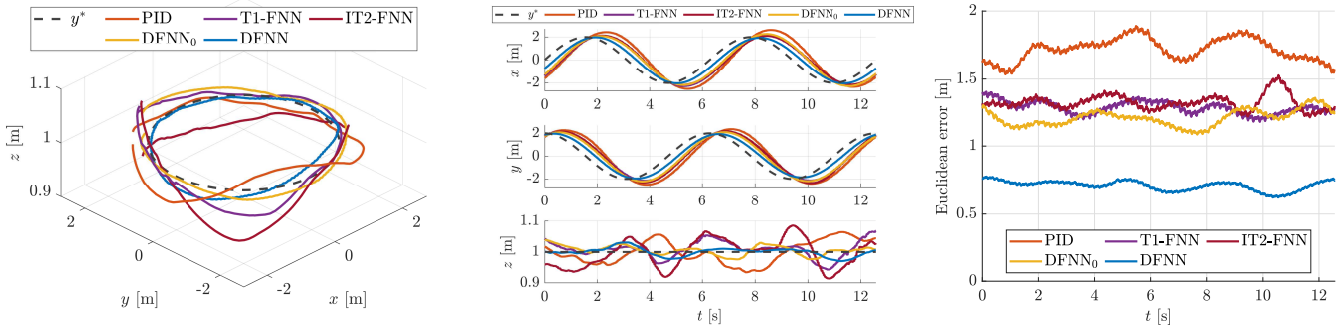
As can be seen from Table III, the DFNN-based controller with online learning outperforms all PID, T1-FNN, IT2-FNN and DFNN_0 for all tested cases in terms of mean absolute error (MAE). Averaged results from numerous experiments depict that the overall improvement of 51%, 59%, 53% and 51% in terms of MAE is achieved as compared to a well-tuned PID controller for slow, fast, near-ground and with payload cases, respectively. While this ratio goes up 21%, 42%, 40% and 46%, when compared to the DFNN_0 controller for the same cases.

Nevertheless, the online DFNN-based controller can learn promptly the system dynamics, the computing time is still the main drawback of this controller because of the online back-propagation. The computing time is polynomially proportional to the product of the number of hidden layers and the number of neurons in each hidden layer, i.e., $O(N_L \cdot N_H^4)$. Therefore, deeper is the network, more complex functions it can learn, but more computational power it requires. The average experimental computation time for DFNN with online back-propagation is around $9.4\mathrm{ms}$, while for PID, T1-FNN, IT2-FNN and DFNN_0 without online learning this time is only $8\mu\mathrm{s}$, $11\mu\mathrm{s}$, $13\mu\mathrm{s}$ and $32\mu\mathrm{s}$, respectively. However, $9.4\mathrm{ms}$ is still an acceptable time for real-time applications, which allows the controller to run at $100\mathrm{Hz}$.



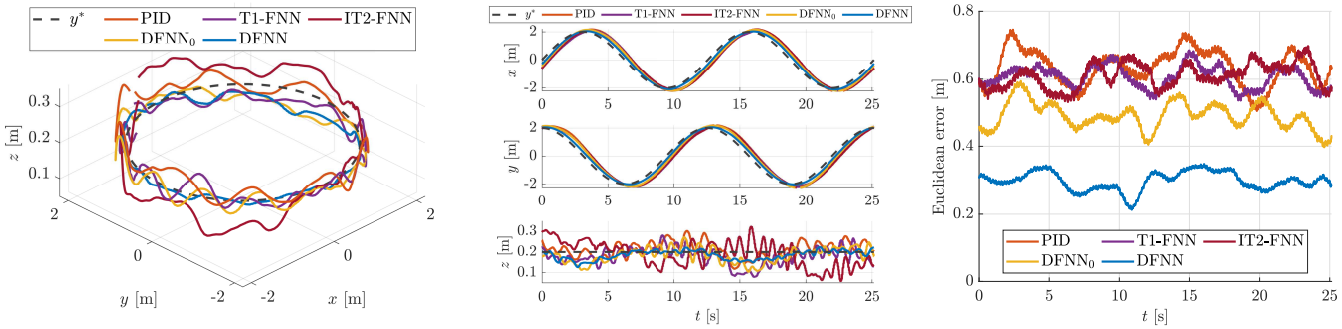
(a) 3D view for the trajectory tracking. (b) Trajectory on x, y and z axes. (c) Euclidean error of the tracking.

Fig. 11. Experimental results for different controllers on the slow circular trajectory at velocity of $1\mathrm{m/s}$.



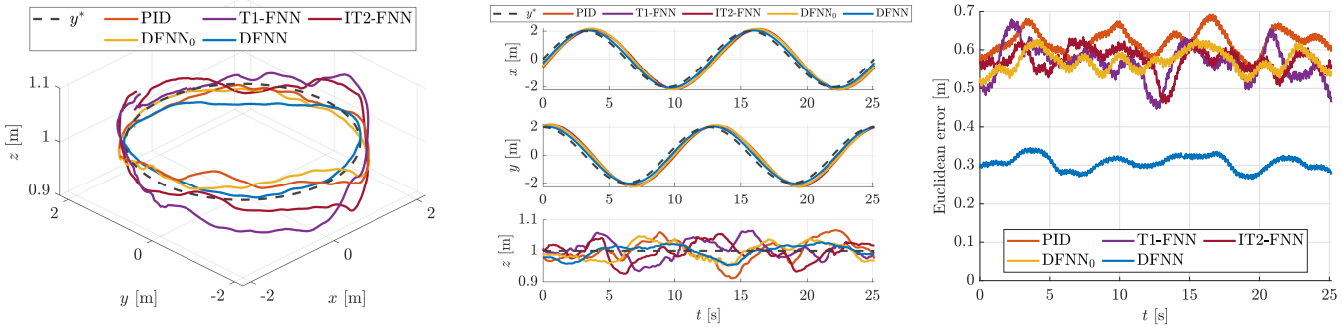
(a) 3D view for the trajectory tracking. (b) Trajectory on x, y and z axes. (c) Euclidean error of the tracking.

Fig. 12. Experimental results for different controllers on the fast circular trajectory at velocity of $2\mathrm{m/s}$.



(a) 3D view for the trajectory tracking. (b) Trajectory on x, y and z axes. (c) Euclidean error of the tracking.

Fig. 13. Experimental results for different controllers on the near-ground circular trajectory at height of $0.2\mathrm{m}$.



(a) 3D view for the trajectory tracking. (b) Trajectory on x, y and z axes. (c) Euclidean error of the tracking.

Fig. 14. Experimental results for different controllers on the circular trajectory with payload of $209\mathrm{g}$.

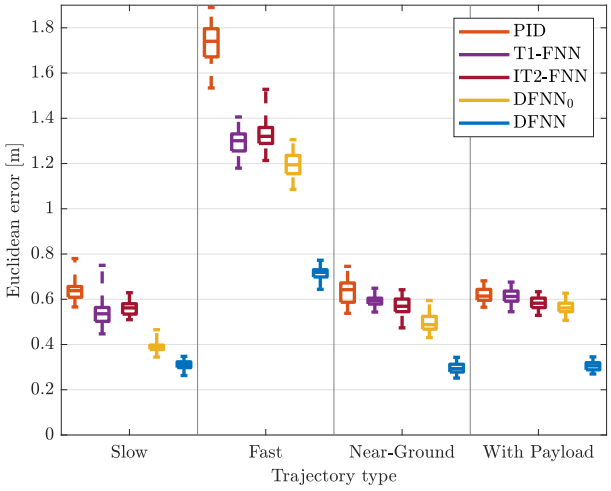


Fig. 15. Box-plot of the tracking performances of five controllers in four scenarios. For each controllers-scenarios case, the experiments are repeated five times under quasi-same conditions. It is possible to observe that DFNN-based controller with online learning has the lowest maximum absolute error, even for the cases unseen during the pre-training.

VI. Conclusions and Future Work

In this work, we presented a novel approach for the control of dynamical systems that improves system's control performance online by combining deep learning and fuzzy logic. The learning is subdivided into two phases: offline and online training. During the offline training phase, a conventional controller performs a set of trajectories and a batch of training samples is collected. Then, a DFNN-based controller, DFNN_0, is pre-trained on the collected data samples. However, DFNN_0 cannot adapt to new operating conditions different from the pre-training cases; therefore, online training is required. During the online training phase, DFNN-based controller takes the control of the system and adapts to improve the tracking performance. The expert knowledge encoded into the rule-base, thanks to the derived fuzzy mapping, provides the adaptation information to DFNN allowing the online learning. Once DFNNs are trained, the experimental results show that the proposed approach improves the performance by more than 50% when compared to a conventional controller. We believe that the results of this study will open doors to a wider use of DFNN-based controllers in real-world control applications.

Appendix A
Stability Proof

Proof. A dynamical system is BIBO stable, if for any bounded input corresponds a bounded output. In our case, the input is $\mathbf{y}^*$; while the output is $\mathbf{y}$.

TABLE III
Comparison of different controllers in terms of MAE [m].

Trajectory	PID	T1-FNN	IT2-FNN	DFNN_0	DFNN
Slow	0.640	0.593	0.568	0.387	0.307
Fast	1.710	1.182	1.265	1.204	0.704
Near-Ground	0.638	0.609	0.601	0.497	0.299
With Payload	0.620	0.555	0.546	0.570	0.306

i) If the controlled system is the nominal system on which DFNN_0 was trained, and if the output from DFNN_0, $\hat{\mathbf{u}}(k)$, accurately approximates the exact inverse of the system, $\mathbf{u}(k)$, i.e., $\hat{\mathbf{u}}(k) \approx \mathbf{u}(k)$; then, the inverse model update, $\Delta\hat{\mathbf{u}}(k) \sim \mathbf{0}$. Thus, the control input to the system is $\mathbf{u}(k)$. From (5), it can be shown that the system's output, $\mathbf{y}$, can be bounded by:

$$\|\mathbf{y}(k)\|_\infty \leq c_1\|\mathbf{x}(k)\|_\infty + c_2\|\mathbf{u}(k)\|_\infty + c_3 \quad \forall k, \quad (34)$$

where c_1, c_2 and c_3 are some positive constants. By using Assumption 3, it can be shown that the state of the dynamical system, $\mathbf{x}$, can be bounded by:

$$\|\mathbf{x}(k)\|_\infty \leq c_4\|\mathbf{u}(k)\|_\infty + c_5\|\mathbf{x}(0)\|_\infty + c_6 \quad \forall k, \quad (35)$$

where c_4, c_5 and c_6 are some positive constants. From (6), it can be shown that the input to the dynamical system, $\mathbf{u}$, can be bounded by:

$$\|\mathbf{u}(k)\|_\infty \leq c_7\|\mathbf{x}(k)\|_\infty + c_8\|\mathbf{y}^*(k)\|_\infty + c_9 \quad \forall k, \quad (36)$$

where c_7, c_8 and c_9 are some positive constants. By using (35) and (36) in (34), the overall closed-loop system in Fig. 3a is BIBO stable.

ii) If the controlled system is different from the nominal system on which DFNN_0 was trained, or if the output from DFNN_0, $\hat{\mathbf{u}}(k)$, does not approximate accurately the exact inverse of the system, $\mathbf{u}(k)$, i.e., $\hat{\mathbf{u}}(k) \not\approx \mathbf{u}(k)$; then, the inverse model update, $\Delta\hat{\mathbf{u}}(k) \neq \mathbf{0}$. Therefore, the control input to the system is $\hat{\mathbf{u}}(k) + \Delta\hat{\mathbf{u}}(k)$. From (5), it can be shown that the system's output, $\mathbf{y}$, can be bounded by:

$$\begin{aligned}\|\mathbf{y}(k)\|_\infty &\leq c_1\|\mathbf{x}(k)\|_\infty + c_2\|\hat{\mathbf{u}}(k) + \Delta\hat{\mathbf{u}}(k)\|_\infty + c_3 \\ &\leq c_1\|\mathbf{x}(k)\|_\infty + c_2\|\hat{\mathbf{u}}(k)\|_\infty + c_2\|\Delta\hat{\mathbf{u}}(k)\|_\infty + c_3 \quad \forall k.\end{aligned} \quad (37)$$

By using Assumption 3, it can be shown that the state of the dynamical system, $\mathbf{x}$, can be bounded by:

$$\begin{aligned}\|\mathbf{x}(k)\|_\infty &\leq c_4\|\hat{\mathbf{u}}(k) + \Delta\hat{\mathbf{u}}(k)\|_\infty + c_5\|\mathbf{x}(0)\|_\infty + c_6 \\ &\leq c_4\|\hat{\mathbf{u}}(k)\|_\infty + c_4\|\Delta\hat{\mathbf{u}}(k)\|_\infty + c_5\|\mathbf{x}(0)\|_\infty + c_6 \quad \forall k.\end{aligned} \quad (38)$$

From (6), it can be shown that the output from the DFNN module, $\hat{\mathbf{u}}$, can be bounded by:

$$\|\hat{\mathbf{u}}(k)\|_\infty \leq \sum_{i=1}^{N_L} c_{\mathbf{W},i}\|\mathbf{W}_i(k)\|_\infty + c_{10} \quad \forall k, \quad (39)$$

where c_{10} is some positive constant. It has to be noted that the inputs to DFNN and FLS modules are bounded by the Gaussian and triangular MFs in (15) and (22), respectively. Simultaneously, the output from the FLS module is bounded by the singleton MFs in Fig. 5b, i.e.:

$$\|\Delta\hat{\mathbf{u}}(k)\|_\infty \leq \alpha < \infty \quad \forall k. \quad (40)$$

By using (38), (39) and (40) in (37), the overall closed-loop system in Fig. 3b is BIBO stable.

Since both control architectures with and without online learning are BIBO stable, the overall closed-loop system is BIBO stable. □

ACKNOWLEDGMENT

This research was supported by the Aarhus University, Department of Engineering (28173).

REFERENCES

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

[2] X. Lin, Y. Rivenson, N. T. Yardimci, M. Veli, Y. Luo, M. Jarrahi, and A. Ozcan, "All-optical machine learning using diffractive deep neural networks," *Science*, vol. 361, no. 6406, pp. 1004–1008, 2018.

[3] L. Jiao and F. Liu, "Wishart Deep Stacking Network for Fast POLSAR Image Classification," *IEEE Transactions on Image Processing*, vol. 25, no. 7, pp. 3273–3286, July 2016.

[4] P. Zhou, H. Jiang, L. Dai, Y. Hu, and Q. Liu, "State-Clustering Based Multiple Deep Neural Networks Modeling Approach for Speech Recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 4, pp. 631–642, April 2015.

[5] J. Zhang and C. Zong, "Deep Neural Networks in Machine Translation: An Overview," *IEEE Intelligent Systems*, vol. 30, no. 5, pp. 16–25, Sept 2015.

[6] J. Mendel and D. Wu, *Perceptual Computing: Aiding People in Making Subjective Judgments*, ser. IEEE Press Series on Computational Intelligence. Wiley, 2010.

[7] L. A. Zadeh, "Is there a need for fuzzy logic?" *Information Sciences*, vol. 178, no. 13, pp. 2751–2779, 2008.

[8] A. Celikyilmaz and I. B. Türksen, *Modeling Uncertainty with Improved Fuzzy Functions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 149–215.

[9] A. M. Yaakob, A. Serguieva, and A. Gegov, "FN-TOPSIS: Fuzzy Networks for Ranking Traded Equities," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 2, pp. 315–332, April 2017.

[10] J. van den Berg, U. Kaymak, and R. J. Almeida, "Conditional Density Estimation Using Probabilistic Fuzzy Systems," *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 5, pp. 869–882, Oct 2013.

[11] T. Kumbasar and H. Hagras, "A Self-Tuning zSlices-Based General Type-2 Fuzzy PI Controller," *IEEE Transactions on Fuzzy Systems*, vol. 23, no. 4, pp. 991–1013, Aug 2015.

[12] T. Zhou, F. Chung, and S. Wang, "Deep TSK Fuzzy Classifier With Stacked Generalization and Triplely Concise Interpretability Guarantee for Large Data," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 5, pp. 1207–1221, Oct 2017.

[13] M. Pratama, J. Lu, E. Lughofer, G. Zhang, and M. J. Er, "An Incremental Learning of Concept Drifts Using Evolving Type-2 Recurrent Fuzzy Neural Networks," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 5, pp. 1175–1192, Oct 2017.

[14] S. Zhou, Q. Chen, and X. Wang, "Fuzzy deep belief networks for semi-supervised sentiment classification," *Neurocomputing*, vol. 131, pp. 312–322, 2014.

[15] A. Sarabakha, N. Imanberdiyev, E. Kayacan, M. A. Khanesar, and H. Hagras, "Novel Levenberg–Marquardt Based Learning Algorithm for Unmanned Aerial Vehicles," *Information Sciences*, vol. 417, pp. 361–380, 2017.

[16] Y. Deng, Z. Ren, Y. Kong, F. Bao, and Q. Dai, "A Hierarchical Fused Fuzzy Deep Neural Network for Data Classification," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 4, pp. 1006–1012, Aug 2017.

[17] A. Isidori, *Elementary Theory of Nonlinear Feedback for Multi-Input Multi-Output Systems*. London: Springer London, 1995, pp. 219–291.

[18] A. Isidori, "The zero dynamics of a nonlinear system: From the origin to the latest progresses of a long successful story," *European Journal of Control*, vol. 19, no. 5, pp. 369–378, 2013, the Path of Control.

[19] 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.

[20] S. Zhou, A. Sarabakha, E. Kayacan, M. K. Helwa, and A. P. Schoellig, "Knowledge Transfer Between Robots with Similar Dynamics for High-Accuracy Impromptu Trajectory Tracking," in *2019 European Control Conference (ECC)*, June 2019, pp. 1–8.

[21] J. P. Hespanha, D. Liberzon, D. Angeli, and E. D. Sontag, "Nonlinear norm-observability notions and stability of switched systems," *IEEE Transactions on Automatic Control*, vol. 50, no. 2, pp. 154–168, Feb 2005.

[22] A. Sarabakha and E. Kayacan, "Online Deep Learning for Improved Trajectory Tracking of Unmanned Aerial Vehicles Using Expert Knowledge," in *2019 IEEE International Conference on Robotics and Automation (ICRA)*, May 2019.

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

[24] 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.

[25] A. Sarabakha, C. Fu, and E. Kayacan, "Intuit before tuning: Type-1 and type-2 fuzzy logic controllers," *Applied Soft Computing*, vol. 81, p. 105495, 2019.

[26] J. M. Mendel, *Type-1 Fuzzy Systems*. Cham: Springer International Publishing, 2017, ch. 3, pp. 101–159.

[27] F. J. Doyle, III and M. A. Henson, "Nonlinear Systems Theory," in *Nonlinear Process Control*, M. A. Henson and D. E. Seborg, Eds. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997, pp. 111–147.

[28] E. Kayacan, A. Sarabakha, S. Coupland, R. John, and M. A. Khanesar, "Type-2 fuzzy elliptic membership functions for modeling uncertainty," *Engineering Applications of Artificial Intelligence*, vol. 70, pp. 170–183, 2018.

[29] 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.

[30] 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.

[31] 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.

[32] S. Weiss, M. W. Achtelik, S. Lynen, M. Chli, and R. Siegwart, "Real-time onboard visual-inertial state estimation and self-calibration of MAVs in unknown environments," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 957–964.

[33] C. Fu, A. Sarabakha, E. Kayacan, C. Wagner, R. John, and J. M. Garibaldi, "Input Uncertainty Sensitivity Enhanced Nonsingleton Fuzzy Logic Controllers for Long-Term Navigation of Quadrotor UAVs," *IEEE/ASME Transactions on Mechatronics*, vol. 23, no. 2, pp. 725–734, April 2018.

Andriy Sarabakha was born in Lviv, Ukraine. He received a B.Sc. degree in computer engineering in 2012 from "Sapienza" - University of Rome, Rome, Italy, as well as a M.Sc. degree in artificial intelligence and robotics in 2015 from the same university. From January 2016, he is a Ph.D. student in Nanyang Technological University, Singapore, at the School of Mechanical and Aerospace Engineering. His research areas are unmanned aerial vehicles, artificial intelligence and fuzzy logic control.

Erdal Kayacan received a Ph.D. degree in electrical and electronic engineering at Bogazici University, Istanbul, Turkey. After finishing his post-doctoral research at KU Leuven at the division of mechatronics, biostatistics and sensors (MeBioS) in 2014, he worked in Nanyang Technological University at the School of Mechanical and Aerospace Engineering as an assistant professor for four years. Currently, he is pursuing his research in Aarhus University at the Department of Engineering as an associate professor. He is the Director of Artificial Intelligence in Robotics Laboratory (Air Lab). Dr. Kayacan is co-writer of a course book "Fuzzy Neural Networks for Real Time Control Applications, 1st Edition Concepts, Modeling and Algorithms for Fast Learning", Butterworth-Heinemann, Print Book ISBN:9780128026878. (17 Sept 2015). From 1st Jan 2017, he is an Associate Editor of IEEE Transactions on Fuzzy Systems.