Deep Neuromorphic Controller with Dynamic Topology for Aerial Robots

Basaran Bahadir Kocer[1], Mohamad Abdul Hady[2], Harikumar Kandath[3], Mahardhika Pratama[4*], and Mirko Kovac[1,5]

***Abstract*— Current aerial robots are increasingly adaptive; they can morph to enable operation in changing conditions to complete diverse missions. Each mission may require the robot to conduct a different task. A conventional learning approach can handle these variations when the system is trained for similar tasks in a representative environment. However, it may result in overfitting to the new data stream or the failure to adapt, leading to degradation or a potential crash. These problems can be mitigated with an excessive amount of data and embedded model, but the computational power and the memory of the aerial robots are limited. In order to address the variations in the model, environment as well as the tasks within onboard computation limitations, we propose a deep neuromorphic controller approach with variable topologies to handle each different condition and the data stream with a feasible computation and memory allocation. The proposed approach is based on a deep neuromorphic (multi and variable layered neural network) controller with dynamic depth and progressive layer adaptation for each new data stream. This adaptive structure is combined with a switching function to form a sliding mode controller. The network parameter update rule guarantees the stability of the closed loop system by the convergence of the error dynamics to the sliding surface. Being the first implementation on an aerial robot in this context, the results illustrate the adaptation capability, stability, computational efficiency as well as the real-time validation.**

I. Introduction

A safe and efficient controller predominantly requires specifications for dedicated aerial robots, a labor-intensive tuning process, precise knowledge for the system and operation environment, and the isolated task definitions [1]. However, such designs are not practical since variations are inevitable, and the computation and memory sources are limited. In order to address this problem, there is a need to possess a computationally efficient learning approach that is neither crafted for a particular task nor needs laborious tuning for each new model and environment.

There are numerous approaches to ensure a safe flight. The common preference is to leverage a mathematical model based on physical principles. However, the system parameters and task requirements might change significantly during the flight [2], [3]. Furthermore, the environmental conditions may not be anticipated, which might degrade the flight performance [4]. A flexible learning scheme for an aerial robot might require physical and/or cybernetical adaptation [5], [6]. In both cases, the system needs to adopt environmental, physical, or task-based changes. For the physical adaptation, there are initial studies proposed with neuromorphic hardware [7]. This study aims to explore the cybernetical adaptation, which includes system inputs, outputs, and parameters, and it might require to estimate associations between them using a limited number of observations. This is a more challenging problem if the association needs to be estimated for one pass loops where there is no labeled or unlabeled batch of data. A similar approach can be seen in nature where the learning structure can be simple and parsimonious to adapt a diverse set of tasks [8], [9]. However, it is still a troublesome work to provide a learning scheme which can adopt new tasks for robots [10], [11]. Transfer learning can be a good candidate for this problem, but most of the available frameworks consider fixed capacity models [12].

Evolution in neural network weights and structure is an active and on-going research [13]. It is a natural process in animals where the adaptation occurs depending on the task, environmental conditions, and the inner structure of the brain [14]. Since the success of the learning heavily depends on the proper selection of architecture and the connection weights, it is critical to find the correct representation of the system [15]. It is a resource and time consuming process to collect the data and tune the connection weights even for a fixed topology of multi-layer neural networks. One of the initial studies explored the use of a spiking neural network (SNN) on a neuromorphic chip [16]. A performance of multilayered neural networks with adaptive weights is exhibited against the wind in [17]. The architectural configuration is fixed and a single task based on wind disturbance rejection is addressed. An evolving SNN is simulated in [18] and tested for a landing problem in [19]. To the best of our knowledge, there is still a gap for deep neuromorphic multilayered neural networks to be applied on aerial robots for various tasks. In our proposed approach and contribution, the following points are highlighted:

- We propose a deep neuromorphic controller with parameters derived based on a sliding surface and com-

(Basaran Bahadir Kocer and Mohamad Abdul Hady contributed equally to this work. This research was mainly carried out when B. B. Kocer, M. A. Hady and H. Kandath were with SCSE, NTU, Singapore.)[1]The authors are with the Aerial Robotics Laboratory, Imperial College London, London SW7 2AZ, UK. [2]Mohamad Abdul Hady is with Electrical Engineering Department, Sepuluh Nopember Institute of Technology, Surabaya-60111, Indonesia. [3]Harikumar Kandath is with Robotics Research Center, International Institute of Information Technology, Hyderabad-500032, India. [4]Mahardhika Pratama is with the School of Computer Science and Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore. [5]Mirko Kovac is with Materials and Technology Center of Robotics at the Swiss Federal Laboratories for Materials Science and Technology, Switzerland.* Corresponding Author.

bined with a dynamic neural network which has self-adjustment of network width (number of nodes) and depth expansion (number of layers) capabilities.
- A novel network parameter adaptation law derived from a sliding mode control design principle that satisfies the conditions for Lyapunov stability.
- Real-time performance evaluation of the online self reconstructed neural network controller, developed with Pytorch library, and implemented onboard for the aerial robot.
- A discussion on real time applicability for aerial robots performing different tasks is provided for such a neuromorphic controller.

In section II, we present the considered problem which is followed by the proposed framework in the section III. In the results and discussion (section IV), the numerical investigations and experiments are discussed with the statistical evaluations. Finally, we demonstrate a video[1] that shows that the learning with deep neuromorphic architecture is feasible for aerial robots.

II. Problem Formulation

For an aerial robot operation, various tasks could arrive in sequence. When this is coupled with a model or environmental changes, a fixed capacity model may fail. In order to handle all these variations in a reasonable computational burden, a neuromorphic approach can be adapted. However, the following aspects need to be addressed: (i) avoiding to grow a neural network model out of the computation limit required for the aerial robot working frequency; (ii) deriving the mechanism for the neural network layer expansion; (iii) preventing the catastrophic forgetting problem where the initial learning segment could be lost with the new data stream; (iv) providing stability for the aerial robot.

Current aerial robots have integrated with the on-board controller to stabilize the system at a lower level yet high-frequency loop [20]. However, e.g., in the case of the inspection, material deposition, or repair operation with an interaction task under wind disturbances, the system needs to fly autonomously while handling different tasks and the variations. Therefore, in this work, it is our aim to design a deep neuromorphic controller which is able to work in different operating conditions to adapt to each new task without any manual parameter tuning.

To formalize the bottlenecks of the aerial robot operations, some modeling and control aspects are described further. As a case study, a control system is constructed that can capture different tasks and variations. It is assumed that the lower-level control achieves tracking of reference commands in all the three axes. On top of them, the deep neuromorphic controller is developed to improve further the system performance.

The position of the UAV is denoted by a vector $P(t) = [x(t), y(t), z(t)]^{\mathrm{T}}$. In this work, the translational dynamics along x-axis is considered to explain the control design. A similar approach is applicable to the other axes and hence not discussed here separately. When disturbances or uncertainties are present, the nonlinear system dynamics along x-axis can be expressed as given below.

$$\dot{x}(t) = v_x(t) \tag{1}$$

$$\ddot{x}(t) = \dot{v}_x(t) = u_x(t) + f(v_x(t), d_x(t)) \tag{2}$$

where $v_x(t)$ is the velocity and $u_x(t)$ is the control input and $d_x(t)$ is the unknown disturbance. The $f(v_x, d_x)$ is an unknown function representing the acceleration due to drag and the other unknown disturbances $d_x(t)$ and uncertainties. The following are the assumptions made in this paper.

Assumption I: The upper bound on the magnitude of the function $f(v_x(t), d_x(t))$ is unknown.

Assumption II: The control input $u_x(t)$ of the translational dynamics is tracked by a suitable controller employed in the inner loop dynamics.

The upper bound on the unmodelled forces, especially those generated due to the interaction of objects in the close proximity of the rotating propeller and other aerodynamic disturbances, are hard to quantify. The inner loop rotational dynamics are faster when compared to the outer loop translational dynamics. So the desired velocity input or acceleration input denoted by $u_x(t)$ is tracked by the inner loop controller. The inner loop controller takes the form of a PID controller and is implemented using the conventional technique of successive loop closure applied to multirotor UAVs [21].

The case study for our learning approach is to design a deep neuromorphic controller for the translational dynamics given by (1) and (2) that can adapt to different tasks and conditions without re-tuning for each scenario. The scenarios considered here are i) to track a desired circular trajectory in the presence of unknown disturbance inputs like wind gusts and ii) tracking the desired altitude in the presence of interaction effects generated due to the proximity of the ceiling and ground to the UAV. A deep neuromorphic controller is proposed, where the number of layers, nodes, and the parameters of the neural network are having self-adjustment ability ensuring stable closed loop dynamics. The notation "(t)", explicitly denoting the variables as a function of time, is dropped in the subsequent sections of this paper for the sake of simplicity.

III. Online Reconstruction and Learning of Deep Neural Network based controller

The proposed controller has a control input u_x comprising of a switching function (u_{s}) and the output from a deep neuromorphic controller (DNC) (u_{nn}). The DNC is a deep neural network consisting of an input layer, l-number of expandable hidden layers and m-number of adjustable nodes, and a linear single output layer. The $\tanh(.)$ activation function is selected for each node of the hidden layers that keep the output between -1 and 1. The topology of the evolution of the morphing deep neural network is presented in Fig. 1. The DNC takes four inputs consisting of the reference and the error horizon in three steps. The output of the network is the control signal for the dedicated axis.

[1] https://www.youtube.com/watch?v=YaDJpo7EP-o

The DNC has the following three features: i) adaptation of network parameters; ii) expansion of network layer; and iii) growing or pruning of nodes in the layer. Each of these features are explained in the below sections.

A. Adaptation of Network Parameters and Stability

The network parameters of the DNC are updated to ensure that the proposed controller given in (3) takes the form of a sliding mode controller (SMC).

$$u_x = u_{\mathrm{s}} + u_{\mathrm{nn}} \tag{3}$$

Following the traditional SMC, let the control input be expressed as the sum of two inputs, as given below.

$$u_x = u_{\mathrm{s}} + u_{\mathrm{eq}} \tag{4}$$

where u_{s} is a switching function of magnitude k as given below.

$$u_{\mathrm{s}} = k\,\mathrm{sgn}(\varsigma) \tag{5}$$

where ς is the sliding surface expressed by

$$\varsigma = \dot{e} + \gamma e \tag{6}$$

where $e = x_r - x$ is the tracking error with x_r the reference signal and the constant $\gamma > 0$. The input u_{eq} is the equivalent control during sliding phase where $\varsigma = 0$ and $\dot{\varsigma} = 0$ [22]. Using (6), it follows that

$$\dot{\varsigma} = \ddot{e} + \gamma\dot{e} = \ddot{x}_r - \ddot{x} + \gamma\dot{e} = 0 \tag{7}$$

From (2) and noting that $u_{\mathrm{s}} = 0$ during sliding phase, (7) can be rewritten as

$$\ddot{x}_r - f(v_x, d_x) - u_{\mathrm{eq}} + \gamma\dot{e} = 0 \tag{8}$$

from which the expression for equivalent control

$$u_{\mathrm{eq}} = \ddot{x}_r - f(v_x, d_x) + \gamma\dot{e} \tag{9}$$

For the convergence of the reaching phase to sliding phase, a Lyapunov function (L) is defined as given below.

$$L = \frac{1}{2}\varsigma^2 \tag{10}$$

$$\dot{L} = \varsigma\dot{\varsigma} = \varsigma(\ddot{x}_r - f(v_x, d_x) - u_s - u_{\mathrm{eq}} + \gamma\dot{e}) \tag{11}$$

Using (9), we obtain

$$\dot{L} = \varsigma(-u_{\mathrm{s}}) \tag{12}$$

By choosing $k > 0$, it can be verified that $\dot{L} < 0$, $\forall\varsigma \neq 0$. However, to apply this control we need to know $f(v_x, d_x)$ as seen from (9). Alternatively, if we know the upper bound on the magnitude of (v_x, d_x) denoted by f_{xm}, i.e. $|f(v_x, d_x)| < f_{xm}$, then it can be verified that choosing $u_{\mathrm{eq}} = \ddot{x}_r + \gamma\dot{e}$ and $u_{\mathrm{s}} = f_{xm}\,\mathrm{sgn}(\varsigma)$ yields $\dot{L} < 0$, $\forall\varsigma \neq 0$. According to *Assumption 1*, f_{xm} is unknown. So in the proposed method, the neural network approximates the equivalent control (u_{eq}) given in (9) using a gradient based network adaptation technique as described below.

In the proposed method $u_x = u_{\mathrm{s}} + u_{\mathrm{nn}}$, where u_{s} is given by (5) and if $u_{\mathrm{nn}} = u_{\mathrm{eq}}$ then the condition $\varsigma\dot{\varsigma} < 0$ is achieved. The procedure for updating the weights of the neural network is based on minimizing the difference between u_{nn} and u_{eq}, expressed as a quadratic function J as given below.

$$J = \frac{1}{2}(u_{\mathrm{eq}} - u_{\mathrm{nn}})^2 \tag{13}$$

Let the neural network output be expressed as

$$u_{\mathrm{nn}} = \Gamma(W, U) \tag{14}$$

where U is the input to the network, W is the vector of adjustable parameters (weights), and $\Gamma(.)$ is a nonlinear function. Following the gradient descent method for updating the network parameters, we obtain the following update rule.

$$\dot{W} = -\alpha\nabla J_W \tag{15}$$

where $\alpha > 0$ is the learning rate. Taking gradient of (13) with respect to W gives

$$\nabla J_W = -(u_{\mathrm{eq}} - u_{\mathrm{nn}})\frac{\partial u_{\mathrm{nn}}}{\partial W} \tag{16}$$

when $u_x = u_{\mathrm{s}} + u_{\mathrm{nn}}$, then

$$\dot{\varsigma} = u_{\mathrm{eq}} - u_{\mathrm{s}} - u_{\mathrm{nn}} \tag{17}$$

From the above equation, we obtain

$$u_{\mathrm{eq}} - u_{\mathrm{nn}} = \dot{\varsigma} + k\,\mathrm{sgn}(\varsigma) \tag{18}$$

Using equations (14) to (18), the parameter update rule given in (15) becomes

$$\dot{W} = \alpha(\dot{\varsigma} + k\,\mathrm{sgn}(\varsigma))\frac{\partial\Gamma(W, U)}{\partial W} \tag{19}$$

Let the input to the network U belong to a compact set denoted by Ω_u. Then, using the universal approximation property of the neural network [23], it follows that the error in approximation is bounded after a finite time $t > t_l$ for some $W = W^*$, i.e.

$$\max_{U\in\Omega_u} |u_{\mathrm{eq}} - \Gamma(W^*, U)| < \epsilon \tag{20}$$

for $t > t_l$ where $\epsilon > 0$ is a finite bound on the network approximation error. For $t > t_l$,

$$\varsigma\dot{\varsigma} = \varsigma(u_{\mathrm{eq}} - u_{\mathrm{nn}} - u_{\mathrm{s}}) \tag{21}$$

Using (5) and (20), the above equation can be written as

$$\varsigma\dot{\varsigma} < |\varsigma|\epsilon - \varsigma k\,\mathrm{sgn}(\varsigma) \tag{22}$$

which implies $\varsigma\dot{\varsigma} < 0$ for $k > \epsilon$. So the convergence of the system to the sliding surface depends upon the network approximation error. From (19), $\dot{W} = 0$ when (ς, $\dot{\varsigma}$) = (0,0) or when $\dot{\varsigma} = -k\,\mathrm{sgn}(\varsigma)$. The condition $\dot{\varsigma} = -k\,\mathrm{sgn}(\varsigma)$ implies the stability condition $\varsigma\dot{\varsigma} < 0$.

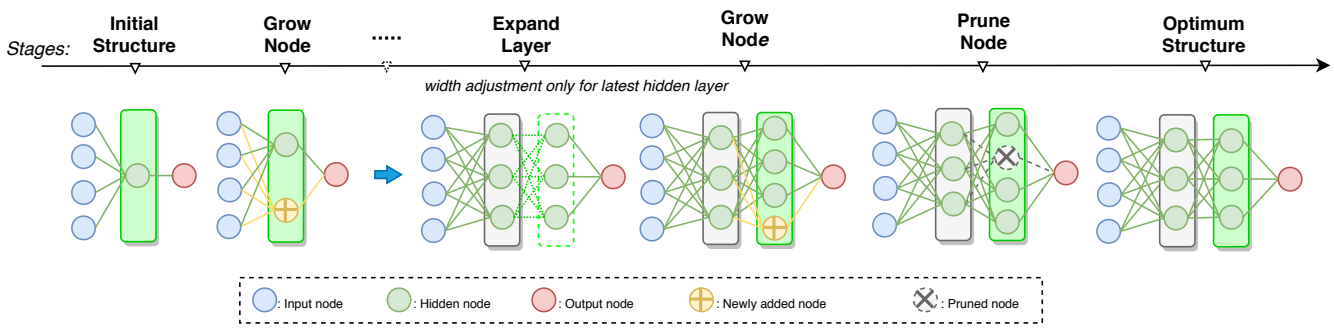


Fig. 1: The structure of the proposed approach: the initial configuration consists of one hidden layer and one node. In this representation, the input of the network is reference and error history in 3 steps. When the neural network model does not capturing the new features of the online data stream, a new configuration can be expected. For growing the node, the condition (28) needs to be satisfied. Similarly, for expansion of the layer, it is required to match the condition (33). To avoid overfitting, a pruning is activated when the condition (30) is satisfied.

B. Online Deep NN Topology Evolution Mechanism

DNC features an elastic network structure where its network structure is self-organized via the hidden node/layer growing and pruning mechanism on the fly while engaging in the control action. This step is performed via the bias-variance decomposition technique where high bias exhibits the under-fitting condition addressed by the addition of new nodes while high variance signifies the over-fitting situation overcome by the removal of inactive nodes. The network bias (β_N) is derived from (18) estimating the deviation of perfect control action. On the other hand, the variance (V_N) is calculated by applying the recursive mean and variance formula with a forgetting factor (λ_f) to the network bias. Both network bias and variance are mathematically expressed as follows.

$$\beta_N = \dot{\varsigma} + u_{\mathrm{s}} \tag{23}$$

$$\mu_\beta^N = (1 - \frac{1}{\omega_N})\mu_\beta^{N-1} + (\frac{1}{\omega_N})\beta_N \tag{24}$$

$$\omega_N = \lambda_f \omega_{N-1} + 1 \tag{25}$$

$$V_N = \frac{1}{\omega_s^N} \sum_{i=1}^{N_w} \lambda_f^{(N-i)} [\beta_N - \mu_\beta^N]^2 \tag{26}$$

$$\omega_s^N = \frac{2\lambda_f(1 - \lambda_f^{(N-1)})}{(1-\lambda_f)(1+\lambda_f)} \tag{27}$$

where both of which can be executed in the one-pass fashion with a sliding window (N_w) for the variance, thereby being compatible for hardware implementation, μ_β^N is the mean value for the network bias and N is total number of time samples.

1) Layer Width Adjustment Mechanism: The node growing strategy is based on the high bias condition identified by the modified statistical process control (SPC) approach. The SPC method is commonly found in the anomaly detection tasks and is applied here to identify the high bias condition. It is mathematically written as follows.

$$\mu_\beta^N + \sigma_\beta^N \geq \mu_\beta^{\min} + \pi \sigma_\beta^{\min} \tag{28}$$

where σ_β^N is the standard deviation of β^N and μ_β^{min}, σ_β^{min} are the lower bounds of μ_β^N and σ_β^N respectively. The underlying difference with the conventional SPC method lies in the presence of the parameter π leading to the dynamic confidence level of the statistical test. π is formulated as follows:

$$\pi = 1.3 \exp\left(-(\beta_N)^2\right) + 0.7 \tag{29}$$

A new node is inserted into the network if (28) is satisfied. The use of π allows flexible approach where the hidden node growing strategy occurs frequently in the high bias condition lowering the confidence level to 68.2% due to $\pi \approx 1$. On the other hand, it becomes strict in the low bias condition resulted from the confidence level around 95.2% as a result of $\pi \approx 2$. Furthermore, μ_β^{min}, σ_β^{min} are reset if (28) is met. The hidden node pruning module is triggered if high variance condition is come across. As with the growing condition, the modified SPC technique is utilized for the pruning mechanism and written as follows

$$\mu_{\mathrm{V}}^N + \sigma_{\mathrm{V}}^N \geq \mu_{\mathrm{V}}^{\min} + 2\chi\sigma_{\mathrm{V}}^{\min} \tag{30}$$

where the parameter χ is akin to π except the fact that the β_N is replaced by V_N. The hidden node pruning strategy is activated once (30) is satisfied. The node to be pruned is the $i-th$ node with the lowest node contribution (NC_i) calculated from the L_2 norm of its consecutive weights (W_i).

$$NC_i = ||W_i||_2 \tag{31}$$

The node having the lowest statistical contribution is obtained from (31). Hence, the j-th node is pruned if the condition $\arg\min_{i\in[1,2,3,..,n]}(NC_i) = NC_j$ is satisfied.

2) Network Depth Expansion Mechanism: The layer expansion strategy is governed by the concept drift detection method identifying the significant change in the data stream, an indication of a heavy shift in dynamics of the UAV, environmental changes like disturbances, or a new task. This implies that the existing model capacity has to be expanded to accommodate the new concept. The addition of a new layer is called for because it is capable of enhancing the model capacity significantly as compared to the addition of new nodes [24]. This method is performed by first finding the gradient of mean bias over preceding evaluation window E denoted as ∇_β^E, to detect whenever the mean bias (μ_β^E) starts

to increase during saturated performance. It is formulated as follows.

$$\nabla_{\beta}^{E} = \frac{\mu_{\beta}^{E} - \mu_{\beta}^{E-1}}{\Delta t} \tag{32}$$

where Δt is the time duration of an evaluation window. A statistical test is derived with the null hypothesis to confirm that the gradient of mean bias is increasing ($\nabla_{\beta}^{E} > 0$) and occurring under saturated performance, denoted as

$$|\nabla_{\beta}^{E} - \nabla_{\beta}^{min}| < \frac{R}{\mu_{V}^{E}} \sqrt{\frac{1}{2E} \ln \frac{1}{\delta}} \tag{33}$$

where $R = \nabla_{\beta}^{max} - \nabla_{\beta}^{min}$ is the range of gradient bias and δ is the confidence level for layer adaptation (is set 0.05 to achieve 95%). When the condition (33) is satisfied and $\nabla_{\beta}^{E} > 0$, a new fully-connected hidden layer is added by copying the latest hidden layer and stored as the closest hidden layer to the output layer. The parameters of the new layer are initialized to unity.

IV. Results and Discussion

A. Numerical Investigations

Two different tasks are defined in the Gazebo environment: 1) hover and 2) tracking of circular trajectories. In the first phase, our approach was tested for hover. After keeping the system in the hover phase for 200 s, the resulting configuration is stored. Afterward, a new trajectory is defined, consisting of a circle in two different velocities. In this case, the stored configuration is fixed for nodes, layers, and weights. At the same time, learning from scratch for neuromorphology, as well as the stored weights, are tested for the same condition. To avoid the chattering effects typically associated with the switching function, it is replaced by a saturation function during the implementation, as given below.

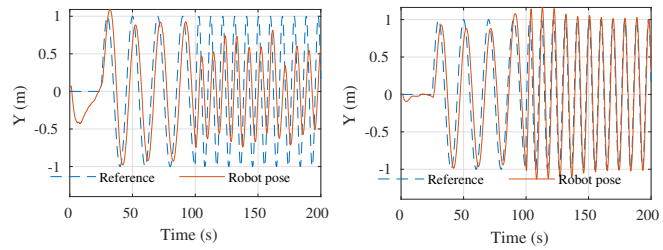
$$u_{\mathrm{s}} = k\,\mathrm{sat}(\frac{\varsigma}{\varsigma_m}) \tag{34}$$

where $\varsigma_m > 0$ and $\mathrm{sat}(\frac{\varsigma}{\varsigma_m}) = \frac{\varsigma}{\varsigma_m}$ for $|\varsigma| \leq \varsigma_m$. For $|\varsigma| > \varsigma_m$, $\mathrm{sat}(\frac{\varsigma}{\varsigma_m}) = \mathrm{sgn}(\varsigma)$. The parameters used for the evaluations are given in Table I.

TABLE I: List of Parameters

u_s, (X, Y, Z)	$k_x = k_y = 0.45$, $k_z = 0.85$, $\varsigma_m = 0.4$
Learning rate (X, Y, Z)	$\alpha_x = 0.0125$, $\alpha_y = 0.0125$, $\alpha_z = 0.25$
Forgetting factor	$\lambda_f = 0.98$
Sliding Window	$N_w = 100$
Evaluation Window	$E = 300$
Layer expansion	$\delta = 0.05$

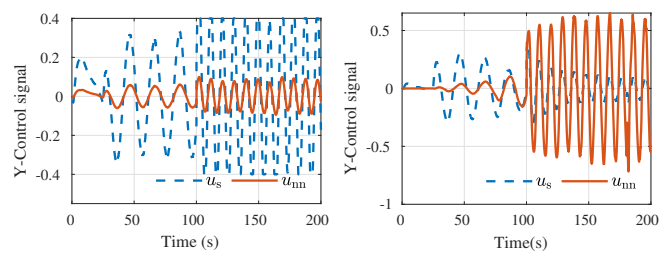
To abide by the page limit, only the Y-axis results are presented to show the effectiveness of adaptive configuration. In the first half of the trajectories, the system tracks a slow trajectory followed by a faster one in the second half, as seen in Fig. 2. Since the fixed configuration is learned in the hover phase, it can track the slower references. However, the fixed-capacity model could not capture a comparably faster trajectory. On the contrary, the deep neuromorphic approach could track both the phases.



(a) Fixed capacity model for Y-axis. (b) Fully adaptive model for Y-axis.

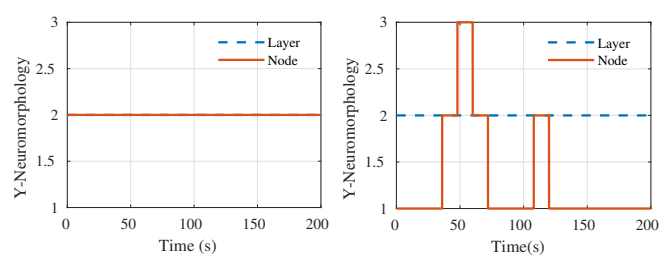
Fig. 2: Tracking of trajectories with deep neuromorphic controller.

In order to achieve this tracking performance, the proposed approach learned the control signal, as can be seen in Fig. 3. In Fig. 3a, the fixed capacity model is not able to drive the error dynamics to the sliding surface. In the fully learning case (Fig. 3b), the neuromorphic controller takes over the control effort and decreases the tracking error in time. This indicates that the error dynamics is converging to the sliding surface as the magnitude of u_{s} is reducing progressively from Fig. 3b. The network topology of the controller for this condition is given in Fig. 4. In order to adapt to the changes in the task, the neuromorphic approach utilizes node growing and pruning, together with the weight adaptation. There is an increase in the number of nodes from 1 to 2 after 30 s and 100 s when the reference input changes. Since the change in the task is not in the level of concept drift, the layer adaptation is not observed for this case.



(a) Control signals of the fixed capacity model. (b) Control signals of the fully adaptive model.

Fig. 3: Learning of control signals with deep neuromorphic controller.



(a) Configuration of the fixed capacity model. (b) Layer and node adaptation for fully adaptive model.

Fig. 4: Topology of the network.

B. Experimental Results

Considering various tasks, such as tracking, wind gust rejection, and surface interaction, a set of experiments are defined. The wind gust rejection is simulated through an

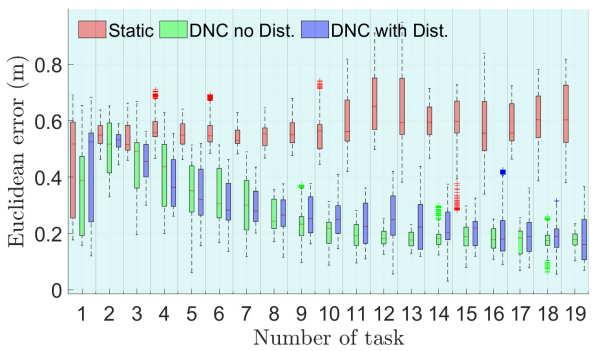


Fig. 5: Circular trajectory tracking: statistical evaluations.

industrial fan and surface interaction is represented through a transparent ceiling. For the real time feasibility, a batch of data is collected, and the computation times are recorded. When the system works with less number of layers (≈3), the computations are in milliseconds range, and goes up to seconds, when the number of layers are increased significantly (>8). The proposed approach is embedded to quad-core Intel NUC for the experimental evaluations with the onboard/hardware settings presented in [25]. Since the tracking results are very similar to the numerical investigations, the experimental results are given with statistical evaluations in Fig. 5. For this phase, the wind gust condition is also considered. Similarly, the fixed-capacity model could not handle the variations and lose the flexibility of capturing the new data stream to learn the driving mechanism of various tracking cases. Furthermore, layer adaptation is observed in real time experiments, which were different than the simulations in Gazebo since there are more unexpected changes and uncertainties in real time. Two different cases are given in Fig. 6. In these experiments, the learning scheme is trying to handle the variations for the wind gust disturbance while tracking the defined circular trajectory. As compared to the disturbances that might be represented in a static manner, the neural network model initiates the layer expansion to handle the wind gust disturbance.

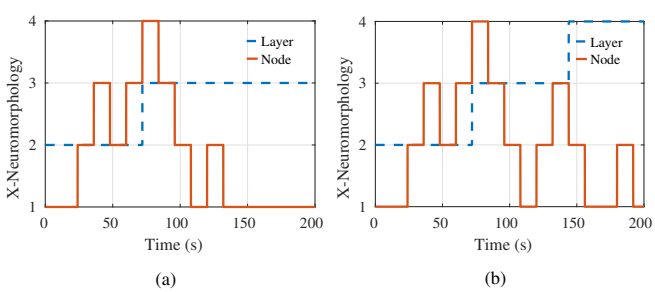


Fig. 6: DNC topology: These data are with the circular tracking results while resisting against the wind gust disturbance in two different experiments.

C. Discussion

During the implementation phase, it is observed that if the learning rate is high, the model adapts to the data stream quickly, but it leads to over-fitting conditions (high variance) when the task is changed. When the learning rate is low, the adaptation becomes slower, and that leads to slow convergence of the error dynamics to the sliding surface. To overcome the aforementioned issue, a weight decay regularization method (adding a damping term "$-\rho W$" to (19) with $\rho > 0$) is employed. Also, our approach is an online continuous learning that does not implement any stopping criteria for learning. Therefore, the implementation of regularization method is crucial for our approach. Based on our observations from simulations, the weight decay factor (ρ) is set to 0.125×10^{-3}. If the tasks are not expecting too many sudden changes, the learning rate $\alpha \in \{0.0125, 0.25\}$ is satisfactory for real-time experiments. In general, the layer addition is observed during a sudden disturbance like a wind gust. The first instant is quite challenging since the node weights need to be initialized. In this implementation, we tested available initialization approaches and ended up with unitary weight since it was the most efficient one. Another effective parameter is the coefficient k (bound on u_s), and various values are tested, and they affect the learning. The higher values might cause the system to learn a slightly aggressive controller while it could be sluggish with very low values. The system can learn as the tested parameters change between 0.45 and 0.85.

V. Conclusion

We demonstrated a deep neuromorphic controller leveraging dynamic topology for both neural network layers and nodes. With the help of the stabilizing conditions, the proposed neuromorphic approach could learn different tasks, variations, and the uncertainties in the system dynamics and environmental conditions. This is a promising approach, particularly for the repetitive tasks under varying environmental conditions or tasks that change over a period of time. The realistic numerical investigations and the initial experiments showed that our approach could learn the changes in the tasks without destabilizing the system. Moreover, the computational aspects are investigated, and it is found that the current onboard approaches can leverage the proposed approach since it does not come with a high computational burden when experimented over different cases. In our future work, we intend to collect the data to a cloud system and share the knowledge for each new robot [26].

Acknowledgment

The research work is supported by the Nanyang Technological University internal grant for the development of a large VTOL research platform. We thank Mr. Peter Zheng for proofreading, corrections and the visual supports. We acknowledge the funding of EPSRC (award no. EP/R009953/1, EP/L016230/1 and EP/R026173/1), NERC (award no. NE/R012229/1) and the EU H2020 AeroTwin project (grant ID 810321). Mirko Kovac is supported by the Royal Society Wolfson fellowship (RSWF/R1/18003).

REFERENCES

[1] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[2] W. Zhang, M. Brunner, L. Ott, M. Kamel, R. Siegwart, J. Nieto, J. Roberts, and P. Pounds, "Learning dynamics for improving control of overactuated flying systems," *IEEE Robotics and Automation Letters*, 2020.

[3] P. Zheng, X. Tan, B. B. Kocer, E. Yang, and M. Kovac, "Tiltdrone: A fully-actuated tilting quadrotor platform," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6845–6852, 2020.

[4] F. Xiao, P. Zheng, J. d. Tria, B. B. Kocer, and M. Kovac, "Optic flow-based reactive collision prevention for mavs using the fictitious obstacle hypothesis," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3144–3151, 2021.

[5] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, pp. 460–466, 2015.

[6] D. Isele, J. M. Luna, E. Eaton, V. Gabriel, J. Irwin, B. Kallaher, and M. E. Taylor, "Lifelong learning for disturbance rejection on mobile robots," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 3993–3998.

[7] H. Blum, A. Dietmüller, M. Milde, J. Conradt, G. Indiveri, and Y. Sandamirskaya, "A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor," *Robotics Science and Systems, RSS 2017*, 2017.

[8] U. Hasson, S. A. Nastase, and A. Goldstein, "Direct fit to nature: An evolutionary perspective on biological and artificial neural networks," *Neuron*, vol. 105, no. 3, pp. 416–434, 2020.

[9] M. Kovač, "The bioinspiration design paradigm: A perspective for soft robotics," *Soft Robotics*, vol. 1, no. 1, pp. 28–37, 2014.

[10] S. Thrun, "Is learning the n-th thing any easier than learning the first?" in *Advances in neural information processing systems*, 1996, pp. 640–646.

[11] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll, "A survey of robotics control based on learning-inspired spiking neural networks," *Frontiers in neurorobotics*, vol. 12, p. 35, 2018.

[12] Y.-X. Wang, D. Ramanan, and M. Hebert, "Growing a brain: Fine-tuning by increasing model capacity," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2471–2480.

[21] A. M. Singh, D. J. Lee, D. P. Hong, and K. T. Chong, "Successive loop closure based controller design for an autonomous quadrotor vehicle," in *Applied Mechanics and Materials*, vol. 483. Trans Tech Publ, 2014, pp. 361–367.

[13] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[14] G. Tang and K. P. Michmizos, "Gridbot: An autonomous robot controlled by a spiking neural network mimicking the brain's navigational system," in *Proceedings of the International Conference on Neuromorphic Systems*, 2018, pp. 1–8.

[15] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[16] T. S. Clawson, S. Ferrari, S. B. Fuller, and R. J. Wood, "Spiking neural network (snn) control of a flapping insect-scale robot," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 3381–3388.

[17] M. Bisheban and T. Lee, "Geometric adaptive control with neural networks for a quadrotor in wind fields," *IEEE Transactions on Control Systems Technology*, 2020.

[18] D. Howard and A. Elfes, "Evolving spiking networks for turbulence-tolerant quadrotor control," in *Artificial Life Conference Proceedings 14*. MIT Press, 2014, pp. 431–438.

[19] J. J. Hagenaars, F. Paredes-Vallés, S. M. Bohté, and G. C. De Croon, "Evolved neuromorphic control for high speed divergence-based landings of mavs," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6239–6246, 2020.

[20] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2992–2997.

[22] J.-J. E. Slotine, W. Li, *et al.*, *Applied nonlinear control*. Prentice hall Englewood Cliffs, NJ, 1991, vol. 199, no. 1.

[23] J. A. Farrell and M. M. Polycarpou, *Adaptive approximation based control: unifying neural, fuzzy and traditional adaptive approximation approaches*. John Wiley & Sons, 2006, vol. 48.

[24] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in neural information processing systems*, 2014, pp. 2924–2932.

[25] B. B. Kocer, M. E. Tiryaki, M. Pratama, T. Tjahjowidodo, and G. G. L. Seet, "Aerial robot control in close proximity to ceiling: A force estimation-based nonlinear mpc," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 2813–2819.

[26] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4555–4562, 2019.