Hardware Implementation of Low-complexity Deep Learning-based Model Predictive Controller

Nirlipta Ranjan Mohanty[1], Saket Adhau[2], Deepak Ingole[3], and Dayaram Sonawane[1]

Abstract— **Model predictive control (MPC) is an advanced control strategy that predicts the future behavior of the plant while considering the system dynamics and constraints. This optimization-based control algorithm needs huge amount of computational resources as it solves the optimization problem at each sampling time. This computational load demands powerful hardware and light-weight algorithms for implementing MPC on an embedded systems with limited computational resources. Deep neural network (DNN) is a attractive alternative for MPC as it provide the close approximation for various linear/nonlinear functions. In this paper, a feed forward neural network (FFNN) and recurrent neural network (RNN)-based linear MPC is developed. These neural network algorithms which are trained offline can efficiently approximate MPC control law which can be easily executed on low-level embedded hardware. The closed-loop performance is verified on a hardware-in-the-loop (HIL) co-simulation on ARM microcontroller. The performance of proposed DNN-MPC is demonstrated with a case study of a 2 degree-of-freedom (DOF) helicopter. Hardware result shows that the DNN-MPC is faster and consumes less memory as compared to MPC while retaining most of the performance indices.**

Index Terms— **Model predictive control, deep neural network, approximate control law, linear systems, HIL co-simulation.**

I. Introduction

Model predictive control (MPC) is an advanced control technique having excellent control performance for various complex systems. The important reasons for its success includes systematically handling multi-input multi-output systems, handling physical constraints on input and output variables, potential to consider multiple process objectives, and ability to control processes with nonlinear time-varying dynamics [1].

MPC calculations are based on current measurements and predictions of the future values of the controlled variables using a mathematical model of the system. It solves numerical optimization problem at each sampling time that minimizes a given objective function and computes a sequence of optimal control actions. The receding horizon control (RHC) strategy, which is a distinguishing feature of MPC, allows applying the first computed control action, thus utilizing the most recent measurements to optimize the control performance [2]. MPC has been successfully implemented in various industrial process applications due to its versatility, robustness, and safety guarantees [3].

Despite having great success of MPC, there are still challenges for implementing MPC in real-time systems. Online implementation of MPC involves a considerable amount of calculations as it repeatedly solves an optimal control problem within the sampling time. The computational burden increases with an increase in the number of constraints, control and state variables, and prediction horizon. For real-time MPC implementation, different approaches are proposed in the literature. One approach is the development of fast solvers includes fast MPC [4], alternating directions method of multipliers (ADMM) [5], and fast gradient method [6] approaches to solving the optimization problem online and are successfully used in implementing MPC. These methods show fast performance when problem dimensions are small, hence limits application to smaller systems. Authors in [7] have proposed a novel technique to use posit number system for efficient embedded implementation of online MPC that reduces the computational burden and memory utilization of MPC.

Another approach is to use explicit MPC (EMPC) [8] that solves optimization problem offline and store the solution in a lookup table (LUT), thus increasing the computational speed as online computations is limited to search algorithm. The major drawback of this approach is that the dimension of the LUT increases exponentially with the length of the horizon and optimization variables hence restricted its application to embedded systems with limited storage capabilities and computational power. Another approach for online optimization is the use of learning-based MPC (LBMPC). On this track, DNN based MPC is proposed that learns the MPC policy and approximates the optimal control law to reduce the computational load [9]. The continued growth of computational power and the availability of large labelled data are two important factors for the recent success of DNN.

The success of MPC mostly depends on the system model. However, in practice, there exists model mismatch with the plant, which leads to the low performance and offset in the reference tracking. The well-known approach to achieve offset-free tracking is to use a disturbance observer augmented with the process model [10]. Nevertheless, this augmentation increases the computational cost of the optimization and limits its applicability to small systems.

In this paper, we propose an approximate offset-free linear MPC using a DNN approach to reduce the computational burden of classical offset-free linear MPC. The idea is to design an DNN-MPC based on data obtained by open-

[1] College of Engineering Pune, Shivajinagar 411005, India. {nrm18.instru, dns.instru}@coep.ac.in
[2] Norwegian University of Science and Technology, Trondheim, Norway. saket.adhau@ntnu.no
[3] KU Leuven, Department of Mechanical Engineering, Leuven, Belgium. deepak.ingole@kuleuven.be

loop offset-free MPC where the pair of state and inputs are stored. The DNN-MPC is designed using two neural network architectures, namely, feedforward neural network (FFNN) and recurrent neural network (RNN). The developed DNN-MPC is implemented on a STM32 Nucleo-144 development board with STM32F429ZI microcontroller. The proposed DNN-MPC is tested in the hardware-in-the-loop (HIL) co-simulation on a case study of a 2 degree-of-freedom (DOF) helicopter. The HIL co-simulation results of DNN-MPC are presented and subsequently compared with the classical offset-free MPC. Presented results show that the developed DNN-MPC is simple, faster, and takes less memory as compared to classical MPC. The performance of DNN-MPC is close to the classical offset-free MPC and successfully handle the output disturbances. The main contributions of this paper are as follows:

- design of DNN based controller to approximate MPC,
- hardware implementation of DNN-MPC with two architectures,
- HIL co-simulation of DNN-MPC and comparison of offset-free MPC and DNN-MPC.

II. Model Predictive Control

This section presents a classical offset-free linear MPC formulation using disturbance modeling approach.

A. MPC Formulation

We consider a discrete-time linear time-invariant (LTI) system described as follows:

$$x(t+T_s) = Ax(t) + Bu(t), \tag{1a}$$
$$y(t) = Cx(t) + Du(t), \tag{1b}$$

where, $x(t) \in \mathbb{R}^{n_x}$, $u(t) \in \mathbb{R}^{n_u}$ and $y(t) \in \mathbb{R}^{n_y}$, are state vector, control input vector, and output vector of the system, respectively. $A \in \mathbb{R}^{n_x \times n_x}, B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$, and $D \in \mathbb{R}^{n_y \times n_u}$ are the system matrices describing the plant dynamics with (A, B) is controllable and (C, A) is observable. Here T_s is the sample time of the system.

B. Disturbance Modeling

To achieve offset-free performance and disturbance rejection, the plant model (1) is augmented with disturbance vector $d(t) \in \mathbb{R}^{n_p}$ as shown below:

$$\underbrace{\begin{bmatrix} x(t+T_s) \\ d(t+T_s) \end{bmatrix}}_{x_e(t+T_s)} = \underbrace{\begin{bmatrix} A & B_d \\ \mathbb{O} & \mathbb{I} \end{bmatrix}}_{A_e} \underbrace{\begin{bmatrix} x(t) \\ d(t) \end{bmatrix}}_{x_e(t)} + \underbrace{\begin{bmatrix} B \\ \mathbb{O} \end{bmatrix}}_{B_e} u(t), \tag{2a}$$

$$y_e(t) = \underbrace{\begin{bmatrix} C & C_d \end{bmatrix}}_{C_e} \underbrace{\begin{bmatrix} x(t) \\ d(t) \end{bmatrix}}_{x_e(t)} + D_e u(t), \tag{2b}$$

where $B_d \in \mathbb{R}^{n_x \times n_p}$, $C_d \in \mathbb{R}^{n_y \times n_p}$ are the disturbance model matrices.

C. Observer Design

Extended state x_e is estimated from the plant measurement by designing a Luenberger observer for augmented system (2) as follows [11]:

$$\hat{x}_e(t+T_s) = A_e\hat{x}_e(t) + B_e u(t) + L_e(y(t) - \hat{y}_e(t)), \tag{3a}$$
$$\hat{y}_e(t) = C_e\hat{x}_e(t) + D_e u(t), \tag{3b}$$

where $L_e = \begin{bmatrix} L_x & L_d \end{bmatrix}$ is the estimator gain matrix for state and disturbance with dimension $(n_x \times n_y)$ and $(n_p \times n_y)$, respectively.

D. MPC Problem Formulation

Using system model (2) and disturbance observer (3), the prediction for differential state and output in (4) can be written as:

$$\min_{U_N} \sum_{k=0}^{N-1} (y_k - y_{\mathrm{r},k})^T Q (y_k - y_{\mathrm{r},k}) + \Delta u_k^T R \Delta u_k, \tag{4a}$$

s.t.

$$x_{k+T_s} = Ax_k + Bu_k + B_d d_k, \quad k = 0, \ldots, N-1, \tag{4b}$$
$$d_{k+T_s} = d_k, \quad k = 0, \ldots, N-1, \tag{4c}$$
$$y_k = Cx_k + Du_k + C_d d_k, \quad k = 0, \ldots, N-1, \tag{4d}$$
$$\Delta u_k = u_k - u_{k-1}, \quad k = 0, \ldots, N-1, \tag{4e}$$
$$u_{\min} \leq u_k \leq u_{\max}, \quad k = 0, \ldots, N-1, \tag{4f}$$
$$u_{-1} = u(t - T_s), \tag{4g}$$
$$x_0 = x(t), \tag{4h}$$

where $Q \in \mathbb{R}^{n_x \times n_x}$ and $R \in \mathbb{R}^{n_u \times n_u}$ are the weighting matrices, with conditions $Q \succeq 0$ to be positive semi-definite, and $R \succ 0$ to be positive definite. We denote N as the prediction horizon, y_r is the reference trajectory of the output and initial conditions are x_0 and u_{-1}. Based on the concept (RHC) [11] the solution of (4) generates sequence of optimal control actions $(U_N^\star = u_0^\star, \ldots, u_{N-1}^\star)$. However, only the first term of the control sequence $(u_0^\star)$ is feed to the plant and a new sequence is calculated using new measurements at the next sampling time. The optimization process is repeated to predict future states x_{k+1} at time $k+1$.

E. Online/Implicit MPC

It is well known that the constrained finite time optimal control problem (4) can be transformed in a standard quadratic programming problem as follows:

$$\min_U \frac{1}{2} U^T H U + f^T U, \tag{5a}$$
$$\text{s.t. } GU \leq w, \tag{5b}$$

where $H \in \mathbb{R}^{n_u N \times n_u N}$ is the hessian matrix, $f \in \mathbb{R}^{n_x \times n_u N}$, $G \in \mathbb{R}^{q \times n_u N}$, $w \in \mathbb{R}^q$, and q is number of inequalities. The QP problem (5) can be solved using well known methods such as active set method and interior point methods [12].

III. Deep Neural Network

Artificial neural network (ANN) is a machine learning approach that mimics the human brain through a set of algorithms. ANN consists of an input and output layer with one or more hidden layer within. ANN having one hidden layer are called shallow neural networks, and more than one layer is termed a deep neural networks. DNN have better approximation capabilities of complex function compared to shallow neural networks [13].

The neurons of neural networks have weights and bias associated with them, and their values are updated during the training process based on the error at the output using a backpropagation algorithm. Activation functions used in ANN are nonlinear mathematical equations that determine the output of the learned model, its accuracy, and computational efficiency. Popular activation functions used in ANN-based controller is sigmoid, hypertangent, rectified linear unit (ReLu). The ANN with an appropriate number of hidden layers and nodes can match the performance of MPC. Generally, the selection of hidden layer and nodes are made by trial and error method [14]. In this work, we have investigated the FFNN and RNN type regression model for designing the neural network controller.

A. Architecture of FFNN

The feedforward neural network was the first and simplest type of ANN devised. In this network, the information moves in only one direction forward from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or feedback loops in the network [9]. Due to the absence of feedback, FFNN provides only a static system mapping. FFNN is defined as a sequence of layers of neurons which determines a function $\mathcal{N}\colon \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ of the form:

$$\mathcal{N} = f_L + 1 \circ g_L \circ f_L \circ \cdots g_1 \circ f_1(x), \tag{6}$$

where $x \in \mathbb{R}^{n_x}$, $y \in \mathbb{R}^{n_y}$ are input and output of the network, L is the number of hidden layers with M number of neurons in each layer. $\mathcal{N}$ is described as DNN for $L \geq 2$ and shallow network for $L = 1$. The interconnections among nodes are shown in Fig. 1. The output of the network is defined as follows:

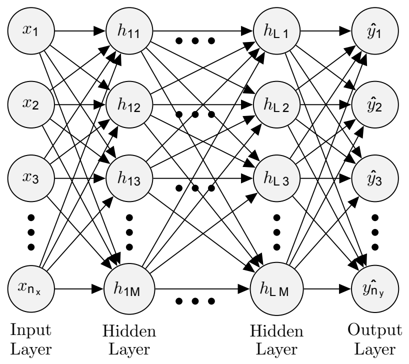

Fig. 1. Architecture of feed forward neural network with L hidden layers.

$$\hat{y}_t = g\left(w_0 + \sum_{i=1}^{n_x} x_i w_i\right), \tag{7}$$

where g is the activation function, w_i are weights for corresponding inputs and w_0 is the bias of the neuron. The objective is to find the optimal values of w_i and w_0 by optimizing some cost function. In this paper hyperbolic tangent are considered as activation function as follows:

$$g(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \tag{8}$$

Equation (7) can be rewrite as follows:

$$\hat{y}_t = g(w_0 + X^T W), \tag{9}$$

where $X = \begin{bmatrix} x_1 \\ \vdots \\ x_{n_x} \end{bmatrix}$ and $W = \begin{bmatrix} w_1 \\ \vdots \\ w_{n_w} \end{bmatrix}$.

B. Architecture of RNN

A recurrent neural network is a class of ANNs where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behaviour. It has both feedforward and feedback connections between the neurons. Previous states generated from past inputs are feedback into the network, and this gives memory to it. While in FFNN, only current input states are considered, in RNN, both current and previous states are focused. It makes RNN capture the dynamic behaviour of a system [15]. RNN structure is generated by adding a loop in the FFNN network, which allows information to process over time. At time step t RNN takes input x_t, computes output of the network $\hat{y}_t$, updates internal state h_t, and passes the updated internal state information to the next time step $t+1$ of the network. The structure of RNN is shown in Fig. 2. Internal state h_t is defined as follows:

$$h_t = g(h_{t-1}, x_t), \tag{10}$$

where g is the activation function, h_{t-1} is previous state from previous time step, and x_t is current input the network is receiving. Equation (10) can be rewrite as follows:

$$h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t), \tag{11}$$

where $\tanh$ is the activation function for hidden nodes, $W_{xh} \in \mathbb{R}^{n_h \times n_x}$ and $W_{hh} \in \mathbb{R}^{n_h \times n_h}$ are the input-to-hidden weight matrix and hidden-to-hidden weight matrix, respectively. The output of the network is defined as follows:

$$\hat{y}_t = W_{hy}^T h_t, \tag{12}$$

where $W_{hy} \in \mathbb{R}^{n_y \times n_h}$ is the hidden-to-output weight matrix.

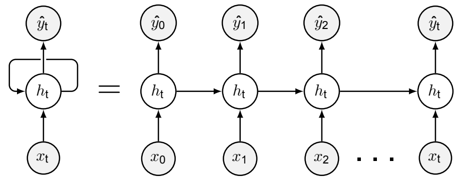

Fig. 2. A recurrent neural network and its unfolded structure.

IV. Case study: 2 Degree of freedom helicopter

To show the performance of developed DNN-MPC, we consider a case study of helicopter control problem. The state-space representation of 2 DOF helicopter model can be expressed as follows:

$$x(t+T_s) = Ax(t) + Bu(t), \tag{13a}$$
$$y(t) = Cx(t) + Du(t), \tag{13b}$$

where state vector $x(t) = [\theta, \psi, \dot{\theta}, \dot{\psi}]^T$ with θ, ψ being the angular positions in pitch axis and yaw axis, respectively, input vector $u(t) = [V_{m,p}, V_{m,y}]^T$ with $V_{m,p}, V_{m,y}$ being the pitch and yaw motor voltage respectively and output vector is $y(t) = [\theta, \psi]^T$. The corresponding system matrices are given as follows:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\dfrac{B_p}{J_{T_p}} & 0 \\ 0 & 0 & 0 & -\dfrac{B_y}{J_{T_y}} \end{bmatrix}, B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \dfrac{K_{pp}}{J_{T_p}} & \dfrac{K_{py}}{J_{T_p}} \\ \dfrac{K_{yp}}{J_{T_y}} & \dfrac{K_{yy}}{J_{T_y}} \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \text{and } D = \begin{bmatrix} 0 \end{bmatrix},$$

where $J_{T_p} = J_{eq,p} + m_{\text{heli}} l^2{}_{cm}$ and $J_{T_y} = J_{eq,y} + m_{\text{heli}} l^2{}_{cm}$. The values of the parameters used in this helicopter model are adopted from [16]. The constraints on input are $-24\text{V} \leq V_{m,p} \leq 24\text{V}$ and $-15\text{V} \leq V_{m,y} \leq 15\text{V}$.

V. Hardware Implementation of DNN-MPC

In this section, we present the training, hardware setup, and implementation design flow of DNN-MPC.

A. Design Flowchart of DNN-MPC

Fig. 3 shows the detailed design flowchart of DNN-MPC. There are three main steps in the development of the DNN-MPC. First, the design of classical offset-free MPC, second, the design of DNN-MPC and third is to implement DNN-MPC on hardware.

- MPC: in this step an offset-free MPC is designed for reference tracking with the constraints on input and states. MPC has simulated in open-loop with random initial conditions (states) to generate learning and target data for DNN training. The pairs of states and inputs are stored for further process.
- DNN: in this step the stored data is divided randomly for training, testing, and validation purpose. The number of hidden layers, hidden neurons, activation function, training algorithm, and loss function is selected as per application requirements. Once DNN is trained it is tested and validated. For unsatisfactory result DNN is retrained by re-initializing weights and biases or changing number of hidden layers and hidden neurons.
- HIL co-simulation: the developed DNN-MPC controller is implemented on a STM32 microcontroller to validate its performance through HIL co-simulations.

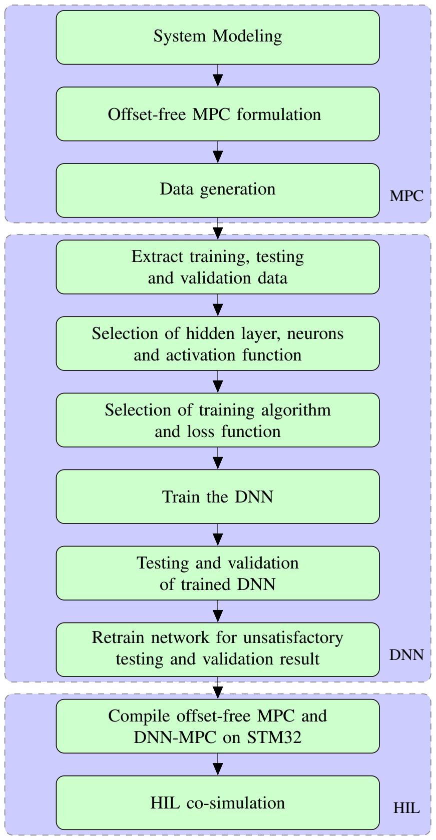
System Modeling
Offset-free MPC formulation
Data generation
MPC
Extract training, testing and validation data
Selection of hidden layer, neurons and activation function
Selection of training algorithm and loss function
Train the DNN
Testing and validation of trained DNN
Retrain network for unsatisfactory testing and validation result
DNN
Compile offset-free MPC and DNN-MPC on STM32
HIL co-simulation
HIL

Fig. 3. Design flowchart of DNN-MPC in MATLAB/Simulink and STM32 MCU.

B. Training of DNN-MPC

To generate data for DNN training, MPC is formulated using a model of the helicopter and constraints of the states and the control inputs. MPC runs in open-loop and for each sample time initial conditions for states and references are chosen randomly from operating state-space of the system and are uniformly distributed. MPC solves the quadratic optimization problem for the given conditions using `mpcqpsolver` solver using interior point method [17] and generates optimal control actions. The initial conditions and the optimal actions are stored for DNN training. If the optimization problem is infeasible, the data points are discarded. The training data sets have information about the policy implemented by the MPC, while solving the optimization problem. Hence the number of generated data pair and their quality has a major influence on the approximation behavior of DNN controller.

The learning data for the FFNN is an array of length six that consists of randomly generated system states, references for pitch and yaw tracking. Teaching data is an array of length two that consists of the first optimal solution of the MPC controller, i.e. pitch and yaw motor voltages. For RNN implementation two more indexes added to the learning array that contains the randomly generated initial condition for control actions. The training data set for neural network training is normalized between $[-1, 1]$. Out of the total data, 70% of the generated data is randomly selected for training the neural network and the remaining 30% for testing and validation purpose. Both FFNN and RNN consists of two hidden layers with ten neurons each. All the hidden nodes have `tanh` activation function with a linear function for the output layer. DNN is trained by using MATLAB deep learning toolbox. `feedforwardnet` and `layrecnet` are used to generate FFNN and RNN model respectively. Training the neural network use optimization process to find the optimal values of the weights and biases of each neuron by minimizing a given loss function. In this work, the loss function is chosen as a mean squared error (MSE) which is the difference between target output and network output. MSE is expressed mathematically as follows:

$$\mathrm{MSE} = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{n}} (\mathrm{y_i} - \mathrm{\hat{y}_i})^2, \tag{14}$$

where n is the number of data points, $\mathrm{y_i}$ and $\mathrm{\hat{y}_i}$ are target value and network output respectively. The training is done using Levenberg Marquardt algorithm [18]. Weights, and biases are updated using the back propagation of error algorithm. Around 100000 data points are generated for training. Time required for DNN to train is 8 minutes 20 seconds and 25 minutes 40 seconds for FFNN and RNN respectively. After the training, the neural network is tested and validated with test data and validation data sets. For unsatisfactory result, the neural network is retrained by re-initializing the weights and biases or altering the hidden layers and hidden neurons.

C. Hardware Setup and Configuration

The designed DNN-MPC is implemented on a STM Nucleo-144 development board with STM32F746ZGT6 MCU. It is an ARM 32-bit Cortex-M7 microcontroller having 1024 Kb of program memory, 320 Kb of SRAM and delivers clock speed up to 180 MHz. The algorithm is developed and run in MATLAB/Simulink environment. Communication between PC and Nucleo board is carryout through UART serial communication with baud rate of 115200.

A MATLAB/Simulink scheme is developed which has a system model to represent the real plant and controller in feedback configuration. The subsytem for the controller is developed and build to generate C/C++ code and deployed on to the hardware. During simulation the controller is loaded into the STM32 microcontroller to execute the calculation in real-time and generate control actions that is given to plant model running in a Simulink. The hardware sampling time, memory footprint, controller output, and the plant output is stored for performance comparison and validation of DNN-MPC. Fig. 5 shows the HIL co-simulation set-up for DNN based MPC.

VI. Hardware Results

A. Reference Tracking Performance

In this work, prediction horizon (N), output penalty (Q), and input penalty (R) were set to 10, $0.001\times\mathbb{I}_{(n_u \times n_u)}$, and $100\times\mathbb{I}_{(n_y \times n_y)}$, respectively. Helicopter sampling time is chosen as 0.1 s. We demonstrated the performance of DNN-MPC with random disturbances applied to pitch angle.

Performance of DNN-MPC is tested for the reference tracking of pitch and yaw angles for a given trajectory for simulation time of 100 s. During HIL co-simulation, a constant disturbance of magnitude 0.1 is applied to pitch angle between 30 to 70 s of simulation. Fig. 6 shows the response of classical offset-free MPC and DNN-MPCs (FFNN and RNN). It can be seen from Fig. 6 (a) that the offset-free MPC tracks pitch angle reference trajectory very well and mitigate disturbance faster (see zoomed subplot at the top right). It is interesting to see that DNN-MPC also tracks the reference trajectory well but shows a little steady-state error and also mitigates disturbances but takes more time as compared to offset-free MPC. Fig. 6 (b) shows the response of yaw angle for all three controllers. It clearly shows the effect of pitch angle on yaw angle response. Fig. 6 (c-d) shows the pitch and yaw motor voltages obtained by three controllers. It can be seen that the DNN-MPC took less voltage to track reference trajectories. Which means that DNN-MPC shows satisfactory performance with less voltage.

B. Performance comparison

The performance of the controller in the presence of unmeasured disturbance is presented in the Table I. Memory footprints, hardware sampling time, MSE, integral square error (ISE), integral time square error (ITSE) and integral absolute error (IAE) are considered as performance indices. The results show that both the DNN controllers reduce memory footprints remarkably, and there is a reduction in computational cost for FFNN. MSE and other key performance indices are not much affected by the implementation of DNN controller.

TABLE I
COMPARISON OF KEY PERFORMANCE INDICES OF DNN-MPC AND OFFSET-FREE MPC.

KPI	**Offset-free MPC**		**FFNN-MPC**		**RNN-MPC**	
Memory (KB)	155		116		116	
Execution time (s)	0.01		0.008		0.013	
	Pitch	**Yaw**	**Pitch**	**Yaw**	**Pitch**	**Yaw**
MSE	0.7569	$5.36E-4$	3.4548	0.0044	3.4993	0.0038
ISE	$7.07E5$	$5.35E2$	$3.4E6$	$4.39E3$	$3.45E6$	$3.78E3$
IAE	$8.60E4$	$2.58E3$	$3.49E5$	$2.53E4$	$3.46E5$	$2.95E4$
ITSE	$7.07E8$	$5.36E5$	$3.40E9$	$4.39E6$	$3.45E9$	$3.78E6$
ITAE	$8.60E7$	$2.58E6$	$3.40E9$	$2.53E7$	$3.46E8$	$2.95E7$

VII. Conclusions

Model predictive control (MPC) is an advanced control strategy that predicts the future behaviour of the plant while considering the system dynamics and constraints. This optimization-based control algorithm needs a huge amount of computational resources as it solves the optimization problem at each sampling time. This computational load demands powerful hardware and lightweight algorithms for implementing MPC on an embedded system with limited

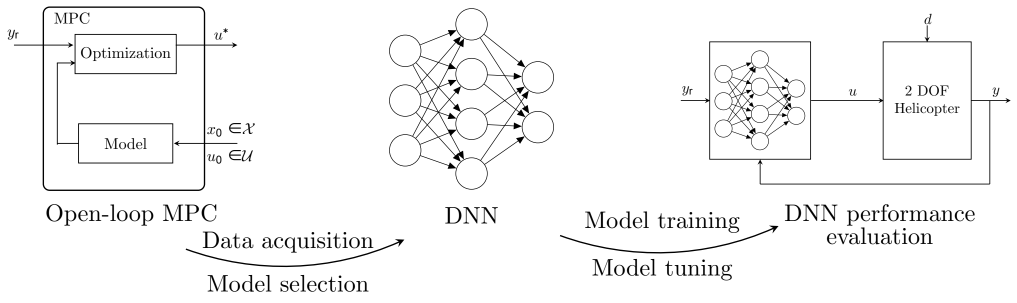

Fig. 4. Schematic of the DNN-MPC. From left to right: MPC runs in open-loop with random initial data. DNN trained with the labeled dataset. Trained DNN model is implemented in closed-loop control to evaluate performance.

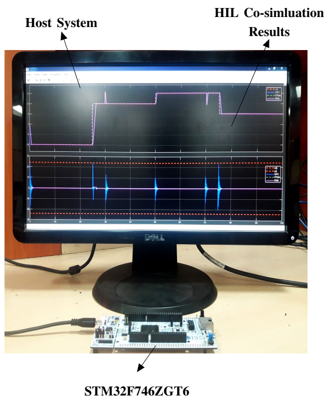

Fig. 5. Hardware set-up used in the HIL co-simulation of DNN-MPC to control 2DOF helicopter.

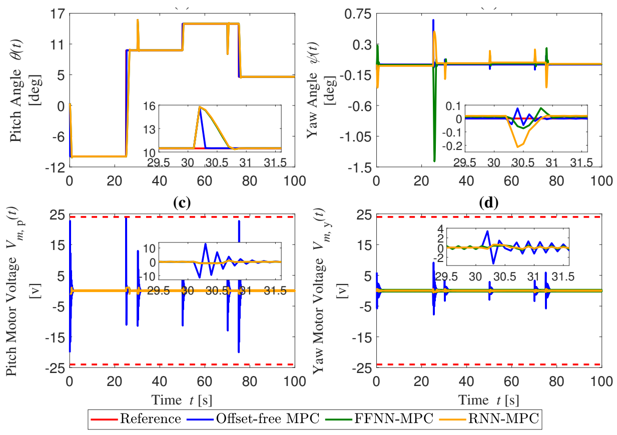

Fig. 6. Reference tracking performance of the DNN-MPC and offset-free MPC implemented on hardware for disturbance applied on pitch axis.

computational resources. A deep neural network (DNN) is an attractive alternative for MPC as it provides a close approximation for various linear/nonlinear functions. In this paper, a feed-forward neural network (FFNN) and recurrent neural network (RNN)-based linear MPC is developed. These neural network algorithms, which are trained offline, can efficiently approximate MPC control law which can be easily executed on low-level embedded hardware. The closed-loop performance is verified on a hardware-in-the-loop (HIL) co-simulation on ARM microcontroller. The performance of the proposed DNN-MPC is demonstrated with a case study of a 2 degree-of-freedom (DOF) helicopter. Hardware result shows that the DNN-MPC is faster and consumes less memory as compared to MPC while retaining most of the performance indices.

References

[1] J. M. Maciejowski, *Predictive control: with constraints*. Pearson education, 2002.
[2] J. A. Rossiter, *Model-based predictive control: a practical approach*. CRC press, 2003.
[3] J. H. Lee, "Model predictive control: Review of the three decades of development," *International Journal of Control, Automation and Systems*, vol. 9, no. 3, p. 415, 2011.
[4] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on control systems technology*, vol. 18, no. 2, pp. 267–278, 2009.
[5] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.
[6] S. Richter, C. N. Jones, and M. Morari, "Computational complexity certification for real-time mpc with input constraints based on the fast gradient method," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1391–1403, 2011.
[7] C. Jugade, D. Ingole, D. Sonawane, M. Kvasnica, and J. Gustafson, "A framework for embedded model predictive control using posits," in *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE, 2020, pp. 2509–2514.
[8] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.
[9] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020.
[10] U. Maeder and M. Morari, "Offset-free reference tracking with model predictive control," *Automatica*, vol. 46, no. 9, pp. 1469–1476, 2010.
[11] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.
[12] J. Nocedal and S. Wright, *Numerical Optimization*, 2nd ed. Springer-Verlag, USA, 2006.
[13] I. Safran and O. Shamir, "Depth-width tradeoffs in approximating natural functions with neural networks," in *International Conference on Machine Learning*, 2017, pp. 2979–2987.
[14] E. Asadi, M. G. da Silva, C. H. Antunes, L. Dias, and L. Glicksman, "Multi-objective optimization for building retrofit: A model using genetic algorithm and artificial neural network and an application," *Energy and Buildings*, vol. 81, pp. 444–456, 2014.
[15] X. Bao, Z. Sun, and N. Sharma, "A recurrent neural network based mpc for a hybrid neuroprosthesis system," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 4715–4720.
[16] Q. Quanser, "2- dof helicopter user and control manual," *Markham, Ontario*, 2006.
[17] "The Mathworks, Inc., Natick, Massachusetts," Available at `http://www.mathworks.com`, 2020.
[18] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*. Springer, 1978, pp. 105–116.