DeepControl: Energy-Efficient Control of a Quadrotor using a Deep Neural Network

Pratyush Varshney[1], Gajendra Nagar[2] and Indranil Saha[3]

***Abstract*— Synthesis of a feedback controller for nonlinear dynamical systems like a quadrotor requires to deal with the trade-off between performance and online computation time of the controller. Model predictive controllers (MPC) provide excellent control performance, but at the cost of a very high online computation. In this paper, we present our experience in approximating the behavior of an MPC controller for a quadrotor with a feed-forward neural network. To facilitate the collection of training data, we create a faithful model of the quadrotor and use Gazebo simulator to collect sufficient training data. The deep neural network (DNN) controller learned from the training data has been tested on various trajectories to compare its performance with a model-predictive controller. Our experimental results show that our DNN controller can provide almost similar trajectory tracking performance at a lower control computation cost, which helps in increasing the flight time of the quadrotor. The hardware requirements of our DNN controller is also significantly less than that for the MPC controller, thus the use of DNN based controller also helps in reducing the overall price of a quadrotor.**

I. Introduction

Autonomous dynamical systems like quadrotors rely on the efficacy of the feedback controllers for their correct operation in uncertain environments. Synthesizing feedback controllers for nonlinear dynamical systems poses tremendous challenges to the control engineers as they need to deal with the trade-off between the performance of the controller and the amount of online computation required for their efficient operation. For example, Proportional Integral Derivative (PID) controller [1] has a simple structure leading to very low online computation overhead. But the performance of PID controllers for complex nonlinear systems is often not satisfactory. On the other hand, a Model Predictive Controller (MPC) [2] can provide excellent performance at the cost of a very high online computation overhead even for a system with linear dynamics. The optimization problems solved in the MPC become further complicated when the system is nonlinear [3]. Although there exist optimization techniques like Sequential Convex Optimization [4] that can solve the nonlinear optimization problems, they suffer from the lack of scalability for complex systems with a large number of state variables.

To use an MPC as an on-board controller for a quadrotor, we need to mount a high-performance computer on the quadrotor. However, high energy requirement of the MPC control computation on the on-board processor reduces the flight time. Adding extra battery may help in increasing the flight time, but reduces the payload capacity. Employing the *explicit MPC* [5], [6] strategy could help in reducing the online computation. However, as observed in [7], the computation time to synthesize an explicit controller grows exponentially both in the number of time steps, and the size of state and input vectors, thus making explicit MPC not suitable for a high-dimensional nonlinear dynamical system like quadrotor.

In this paper, we explore the possibility of synthesizing a feedback controller for a quadrotor in the form of a neural network. We employ supervised learning [8] where we generate training data capturing the state-control mapping from the execution of a model predictive controller. However, generation of training data by flying a quadrotor is tedious as the battery of the quadrotor needs to be charged for several times in the process of generating the training data. To alleviate this problem, we devise a mechanism to create a faithful model of the quadrotor which can be used for simulation on the Gazebo Simulator [9] to generate the required training data.

We train a two-layer Deep Neural Network (DNN) to approximate the behavior of an MPC. Through a series of experiments, we evaluate the efficacy of the DNN based feedback controller. Our experimental results demonstrate that the trajectory tracking performance of the DNN based feedback controller is close to that of the MPC controller. However, the control cost of the DNN based controller is consistently better than the MPC controller as the DNN represents a smooth function. We also measure the flight time for the quadrotor under both the MPC controller and the DNN based controller. Though our test vehicle is quite heavy and the power consumption to run the motors is significantly more than the power required for computation, our DNN based controller helped in increasing the flight time upto 12.5%. Finally, one major advantage of DNN controller is that its hardware requirement is low. The hardware required to run the DNN based controller is only 20% of that of the hardware required to run the MPC. This helps us reduce the overall cost of a quadrotor significantly.

Related Work. Various control mechanisms have been introduced for quadrotor, for example, sliding-mode control [10], feedback linearization [11], H_∞ robust control [12], adaptive control [13], and minimum snap trajectory based control [14]. All these control mechanisms assume the availability of a faithful mathematical model of the quadrotor. To address this limitation, Bansal et al [15] has recently

*This work was supported by SERB Early Career Research Award ECR/2016/000474

[1]Pratyush is with Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India pratyushvarshney at cse.iitk.ac.in

[2]Gajendra is with Department of Aerospace Engineering, Indian Institute of Technology, Kanpur, India gajena at iitk.ac.in

[3]Indranil is with Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India isaha at cse.iitk.ac.in

proposed a mechanism to model the dynamics of a quadrotor using a neural network and showed how this neural network based model can be used for control computation. Li et al. [16] have used a DNN to generate a feasible reference trajectory from a user given desired trajectory for a quadrotor.

There have been some recent work on synthesizing neural network based feedback controller for dynamical systems. Recent work has shown that Reinforcement Learning [17] can be successfully used to synthesize a controller for a nonlinear dynamical system like quadrotor. The authors have used Actor-Critic Algorithm [18] to synthesize the controller. Unfortunately, the time required for such policy search is enormous. Zhang et al. [19] proposed a methodology to train a neural network through policy search methodology based on the data generated from an MPC controller under full state observations. However, the states available from the onboard sensors only have been used as the inputs to the neural network based controller so that it can control the quadrotor based on only the on-board sensors. As the neural network gets trained only based on the data from the on-board sensors, such a technique is not guaranteed to be successful for all possible trajectory tracking. Recently there has been work on approximating MPC controller using Neural Networks [20]. The authors show that using Dykstra's projection they are able to reduce the Policy search space. However, their approach is restricted to linear systems.

To the best of our knowledge, we, for the first time, approximate the behavior of an MPC controller for a quadrotor with a deep feed-forward neural network based feedback controller and carry out its detailed performance analysis.

Paper Organization. The rest of the paper is organized as follows. In Section II we introduce the background and formalize the problem statement. In Section III we provide the detailed methodology that approximates the behavior of an MPC closely. Tn Section IV, we describe our experimental setup and explain the experiment results with detailed analysis about the performance of the controller. Finally, we conclude the paper in Section V.

II. Background and Problem Statement

A. Quadrotor Dynamics

The quadrotor is a nonlinear dynamical system that can be represented with discrete-time state space equation of the form:

$$s_{t+1} = f(s_t, u_t)$$

where s_t is the state of quadrotor and u_t is the control applied at time t. The control applied is generated by a feedback controller based on the state. A feedback controller can be defined as a function (μ) that takes the desired state s_t^d and current state s_t as inputs, to give the desired control u_t, which when applied to the system minimizes the error between desired state and the current state.

$$u_t = \mu(s_t, s_t^d)$$

The state-space of the quadrotor is a 12 dimensional vector [21] defined as: $s_t = \begin{bmatrix} p & v & \zeta & \omega \end{bmatrix}^T$, where p is the position of the quadrotor: $\{p_x, p_y, p_z\}$ in x, y, and z dimension, v is the velocity :$\{v_x, v_y, v_z\}$, ζ are the three attitude Euler angles [22]: {roll (ϕ), pitch (θ), yaw (ψ) }, and ω is the body rates: $\{\omega_\phi, \omega_\theta, \omega_\psi\}$. The dynamics of the quadrotor are such that roll and pitch movements are decoupled from the yaw movement.

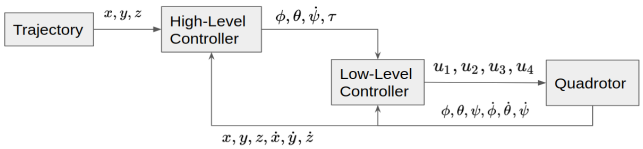

Fig. 1: Controller architecture of autopilot. The high-level controller takes trajectory as input and outputs roll (ϕ), pitch (θ), yaw rate ($\dot{\psi}$) and thrust. Low-level controller runs in the cascaded loop and outputs the motor speed required to maintain the desired orientation.

.

B. Flight Stack Architecture

The controller of the quadrotor is generally a two level cascaded controller. The high-level controller takes in the desired trajectory (P^d) defined as: $P^d = \{p_1^d, p_2^d, ...\}$ and generates the corresponding desired high level controls: roll (ϕ^d), pitch (θ^d), yaw rate($\dot{\psi}^d$) and thrust (τ^d), to reach the target position. Here p_t^d denotes the desired position of the quadrotor at time t. The low-level controller takes the desired attitude and thrust as input and outputs the motor speed, that maintains the desired attitude and position. The high-level controller generally runs at a much slower rate than the low-level controller.

C. High Level Controller

The high-level controller can be implemented by many types of the control algorithms from the classical control theory. There is a wide spectrum of controllers that can be synthesized for a quadrotor. On the one side of the spectrum, there is the PID controller that has low computation cost but also provides poor performance. On the other hand there is MPC based on the optimal control theory. The MPC provides high performance,but has a high computational cost.

1) Proportional-Integral-Derivative Controller: The PID controller is one of the simplest linear controller. It consists of three constants k_P, k_I, k_D. For the current state $s(t)$ and desired state $s_d(t)$, the control $u(t)$ is generated by PID as:

$$u(t) = k_P * e(t) + k_D * \frac{de(t)}{dt} + \int k_I * e(t)dt$$

where $e(t) = s(t) - s_d(t)$. Generally the three constants are found by observing the behaviour of the system and fine-tuning them accordingly. The run-time computation cost of the PID controller is very low compared to optimal controllers. This comes with a drawback in performance. As PID controller does not take into account the dynamics of the system explicitly, the performance of the controller somewhat depends on the fine-tuning mechanism of the three constants, which may be extremely time consuming.

2) Model Predictive Controller (MPC): The MPC is an optimization based controller that generates the controls by solving a constrained optimization problem for minimizing trajectory cost over a finite horizon. The optimization problem solved by MPC is defined as:

$$\begin{aligned} \min_{u_t} \quad & J = \sum_{t=0}^{H_T} \left(s_t^T Q s_t + u_t^T R u_t\right) \\ \text{subject to} \quad & s_{t+1} = f(s_t, u_t) \\ & u_t = \mu(s_t, s_t^d) \\ & u_t \in \mathcal{U} \\ & s_t \in \mathcal{S} \\ & u_{max} \leq u_t \leq u_{min} \end{aligned}$$

where $\mathcal{U}$ is the control space and $\mathcal{S}$ is the state space of the dynamical system. The symbols u_{min} and u_{max} denote the upper and lower bounds on the controls respectively.

The MPC generates the controls for horizon H_T from the current state s_t. The system then applies the first control and feedbacks its observation s_{t+1} (we can assume the state is fully observable) to the MPC. In the next step, again an optimization problem is solved, starting from the state s_{t+1}. This is repeated for the complete trajectory. As MPC requires to solve the optimization problem at every step, the cost of the computation is very high as compared to PID.

If the dynamics of the system $s_{t+1} = f(s_t, u_t)$ is non-linear, then the problem becomes a nonlinear optimization problem. This type of MPC is termed as Non-Linear Model Predictive Controller (NMPC).

The high-level flight controller when implemented as MPC provides a good performance but requires additional hardware. This hardware is generally a high end computation processor that consumes additional power. Thereby the flight time of the quadrotor is reduced. When the high-level flight controller is implemented as PID controller, the hardware required to execute the controller consumes low power compared to the MPC. The performance of this controller is not as good compared to MPC. The MPC can deal with the external disturbances more effectively than a PID controller.

To address the trade-off between performance and computation cost, we propose a Deep Neural Network (DNN) based controller that approximates the MPC controller using supervised learning. We observe that the performance of the controller is comparative to MPC but computation cost is comparative to the PID controller. As the hardware processor required to run a DNN controller is significantly less expensive than the hardware processor required to run an on-board optimizer, a quadrotor using a DNN based controller is also becomes economical. Below, we we briefly introduce the DNN architecture.

D. Neural Networks

Neural Networks [23] have emerged as one of the most efficient function approximators. The advancements in the Deep Learning [24] have shown that Neural Networks are effective in areas like speech recognition [25], image recognition [26] etc. We use Neural Network to approximate an offline optimizer for MPC. The Neural Network consist of one or more layers. Each layer consists of neurons that are fully connected to all the neurons in the next layer, except the final layer neurons. The connection between neurons are weighted. The aim is to learn the weights in such a fashion that the Network is able to output the controls generated by MPC for a given state as input. For a layer l with X as the input vector, the output of the layer l, y_l, is defined as:

$$y_l = \phi(W_l^T X + B_l)$$

where W_l is the weight matrix between layer (l) and $(l+1)$ and B_l is the bias. The activation function ϕ performs the non-linear transformation. Examples of activation functions are *tanh*, *sigmoid* [27], Rectified Linear Unit (ReLu) [28] etc. The parameters W and B are learned by minimizing the loss between the predicted value by the neural network $\hat{f}(x)$ and the desired output y. One of the frequently used loss function for regression is the *huber loss* [29] defined as:

$$L(\hat{f}(x), y) = \begin{cases} \frac{1}{2}(y - \hat{f}(x))^2 & \text{for} \quad |y - \hat{f}(x)| \leq \delta \\ \delta|y - \hat{f}(x)| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

where, δ is the threshold from where the huber loss changes from quadratic to linear. The gradients from the loss are used to learn the weights W by gradient descent methods [30].

$$W \leftarrow W - \alpha \frac{\partial}{\partial W}(L(\hat{f}(x), y))$$

Here α is the learning rate that controls the amount of gradient to be adjusted with respect to the loss gradients.

III. Methodology

In our method we learn offline optimizer of MPC using a Deep Neural Network (DNN). The neural networks are very good at generalizing. They also have high non-linearity. We utilize the generalization and non-linear nature of the neural network to learn an offline optimizer.

Our method involves mainly four steps:

1) Create a simulation model of the quadrotor using the learned system dynamics.
2) Learn the controller using supervised learning from simulation data.

A. Simulation Model of Quadrotor

We first create a model of the quadrotor in the simulator whose performance is as close as possible to the real world quadrotor as there are multiple benefits of learning the system model. For example, to perform supervised learning we need to collect data by flying the quadrotor in real world. This can prove to be a tedious task as the battery life of the quadrotor with an on-board processor running on it is quite low.

To overcome this limitation we create a simulator model of the quadrotor. We model it near to the real world by using the transfer function [31] H(s). The transfer function gives an approximation about the effect of the control on the low level dynamics of the quadrotor. For our simulator model, we use the NMPC as the high level controller. For low-level controller we use the traditional PID controller.

The dynamics of the simulator model and real world model can be compared by the transfer functions: H(s). We tune the constants (k_P, k_I, k_D) of the PID controller of the simulated

model of quadrotor such that on applying the high-level controls, we get the similar transfer function as of the real-world quadrotor. This ensures that our modelled quadrotor behaves in a similar fashion like the real world quadrotor. We can tune the PID parameters by observing the output of the transfer function. If the values of the time constant is more in the simulator than the real world quadrotor, we either decrease the k_D or increase the k_P. Similarly if the values of the time constant is less in the simulator than the real world quadrotor, we either increase the k_D or decrease the k_P.

We implement a NMPC controller [32] on the quadrotor. The NMPC is a high-level controller that outputs three Euler angles: roll (ϕ), pitch (θ), yaw rate($\dot{\psi}$) and Thrust (τ) as output. These controls from the high-level NMPC are passed to a low-level PID controller that maintains the desired angles and thrust.

We run the NMPC controller on different trajectories that cover motion of the quadrotor in all the three *x,y,z* axes. We decouple the yaw (ψ) of the quadrotor as it is independent of the roll (ϕ), pitch (θ) and thrust (τ).

The collected data is used to learn the transfer function of the quadrotor defined by:

$$H(s) = \frac{Y(s)}{X(s)} \tag{1}$$

where Y(s) and X(s) are the output and input functions respectively. For quadrotor the Y(s) is: the current orientation $\{\phi, \theta\}$. The input X(s) is the commanded roll and pitch $\{\phi_{cmd}, \theta_{cmd}\}$.

The low level controller of the quadrotor is responsible for maintaining the desired orientation provided by the high-level controller. The low-level controller generates the rates as output and takes orientation as input.

$$\dot{\phi} = \frac{1}{\tau_\phi}(k_\phi \phi_{cmd} - \phi) \qquad \dot{\theta} = \frac{1}{\tau_\theta}(k_\theta \theta_{cmd} - \theta)$$

where θ_{cmd} and ϕ_{cmd} are the commanded roll and pitch angles respectively. k_ϕ and k_θ are the roll and pitch gains, τ_θ and τ_ϕ are the roll and pitch time constants.

We calculate the gains k_ϕ and k_θ as

$$k_\phi = \frac{\phi}{\phi_{cmd}} \qquad k_\theta = \frac{\theta}{\theta_{cmd}}$$

The time constants are calculated using the `pole` function of the `matlab`.

$$\tau_\phi = \frac{-1}{pole(k_\phi)} \qquad \tau_\theta = \frac{-1}{pole(k_\theta)}$$

B. Learn NMPC Controller using Supervised Learning

The generalization power of Deep Neural Networks can be used to approximate an offline optimizer used in NMPC. We show that a controller learned using supervised learning from the data collected from NMPC is able to perform close to the NMPC.

In the supervised learning setup, the neural networks learn to approximate a function mapping such that: $f(x) : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ is the set of inputs $\{x_1, x_2, ..., x_N\}$ and $\mathcal{Y}$ is the set of corresponding output $\{y_1, y_2, ..., y_N\}$. The neural network learns a function $\hat{f}(x; w)$ paramterized by weights w, that is close to $f(x)$.

Supervised learning can be used to approximate the NMPC controller for the quadrotor. The function $f(x)$ for the quadrotor is mapping between the state to controls: $f(x) : \mathcal{S} \rightarrow \mathcal{U}$. The state $s \in \mathcal{S}$ is $s = \begin{bmatrix} p & v & \zeta & \omega \end{bmatrix}^T$

The controls generated by the NMPC, $u \in \mathcal{U}$ is the commanded roll (ϕ_{cmd}), pitch (θ_{cmd}), yaw rate ($\dot{\psi}_{cmd}$) and thrust τ. $u = \{\phi_{cmd}, \theta_{cmd}, \dot{\psi}_{cmd}, \tau\}$

Supervised learning requires labelled data from which the neural network can learn the function to approximate. We create labelled data by flying the quadrotor in simulation on several setpoints, sampled uniformly from the size of the flying arena. The state s and the controls generated by the NMPC u are recorded for the setpoints. We learn the parameter weights w of the deep neural network (DNN) controller by gradient descent.

$$u_w = \hat{f}(s; w)$$

We implement the learned DNN on an embedded low cost, low power consuming processor: `RaspberryPi 3`. We use open source `PX4` [33] as the auto-pilot. The controller in the `PX4` is a cascaded feedback PID controller at both high-level and low-level. The high level position controller is a PD controller that generates roll (ϕ), pitch (θ), yaw (ψ) and thrust (τ). The low-level controller is a PID controller that takes input from the high-level controller and outputs the control moments $\{u_1, u_2, u_3, u_4\}$. These moments are passed through a mixer that converts the control moments to motor speed.

We replace the high-level PD controller from the `PX4` with our learned DNN controller. As the weights of the DNN is a matrix, the computation cost of the controller is quite low as compared to the NMPC.

IV. Experiment

A. Experimental Setup

In our experiments, we use Flame Wheel (f450) quadrotor with 'X' configuration, as shown in Fig. 2b. The distance between the diagonal motors of the quadrotor is 450 cm. The quadrotor has 4 Electronic Speed Controllers (ESCs) rated $20\mathrm{Ampere}$ each attached with motor of 980 Kv ratings. For the low level controller we use a *pixhawk* board with `PX4` as the firmware. The battery used to fly this quadrotor is a 3 Cell Lithium Polymer of 5300 $\mathrm{mA\,h}$ rating.

We implement the MPC controller on the flame-wheel using an on-board Odroid XU4 [34]. The Odroid has A7 cortex Octa-core processor with 2GB of RAM. The Odroid has a wifi module that communicates with the Ground Control Station (GCS). The communication between Odroid and *pixhawk* is done using *mavlink* [35] protocol.

The complete experimental setup is shown in Fig. 3. For indoor localization, we use VICON systems [36] that provides the odometry at 100 Hz to the GCS. The GCS and

(a) Flying Arena (b) Flame-wheel f450 Quadrotor

Fig. 2: (a) The dimension of the flying arena is $4 \times 2 \times 1$ m. The quadrotor is in hover state using the DNN controller. (b) The f450 quadrotor with raspberry 3.0 mounted on it.

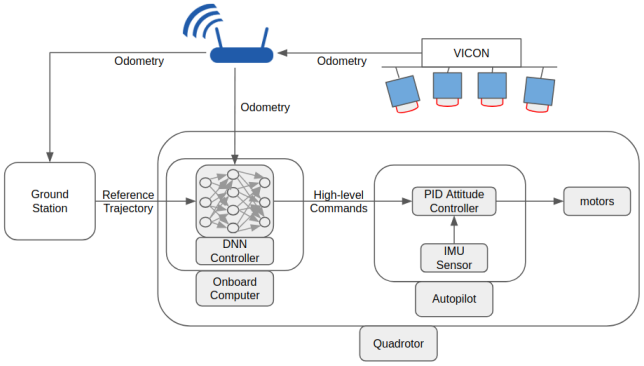

Fig. 3: Block diagram of experimental setup for DeepControl. The Ground Station sends the reference trajectory to the high level DNN Controller running on-board the quadrotor. The low-level attitude controller maintains the desired orientation. The VICON system provides the localization feedback for the controller.

Odroid both run ROS [37] nodes for communication. The odometry from the VICON system is processed and sent to the ROS node running on the Odroid by the GCS. The GCS also publishes the target position to the Odroid. The MPC package running on the on-board Odroid takes as input: Odometry and target position from GCS. The output of the MPC: $\{\phi_{cmd}, \theta_{cmd}, \dot{\psi}_{cmd}, \tau_{cmd}\}$ is sent to the `PX4` running on the *pixhawk* board. The low level PID controller on the *pixhawk* generates the rotor speed commands that maintain the desired attitude.

B. System Identification and Simulation Modelling

For the purpose of system identification, we manually flew the quadrotor in our lab arena and recorded the state of quadrotor (s_t) and control input from the RC transmitter (u_t) at time t, using ROS.

We recorded the data at 100Hz. The manual flight of the quadrotor was performed for 1 minute in which we performed maneuvers such that the effect of commanded roll and pitch can be identified. From the recorded data we learn the transfer function H(s) of the quadrotor. For our quadrotor the values were - mass: 1.45, roll time constant: 0.185622, pitch time constant: 0.163042, roll gain: 1.037192, pitch gain: 0.966195.

The above constants are able to define the low-level dynamics of the real-world quadrotor. We create a simulation model of the f450 in Gazebo [38]. The structure of the f450 was obtained from Open-UAV [39]. The dynamics of the quadrotor do not exactly match with our real-world quadrotor as the motors, ESC etc cannot be modelled very precisely. Therefore we tune the low-level PID controller gains of the simulated model such that the transfer function of the simulated model is close to our real-world f450. The tuning of the PID gains can be done in few iterations. In our case, the coefficient were, for roll: $k_P = 160$, $k_D = 30$, for pitch: $k_P = 181$, $k_D = 30$. We observe that for the above numbers, the transfer function of the simulated f450 and real world f450 are nearly equal as shown in Fig. 4 and Fig. 5.

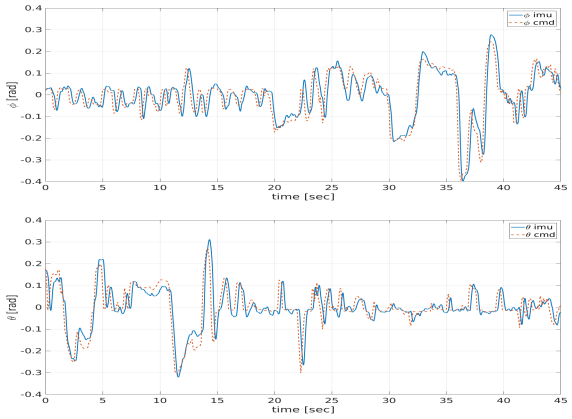

Fig. 4: Orientation tracking of the Real Flame-Wheel 450 quadrotor. The blue line represents the orientation of the quadrotor and red line represents the commanded angles.

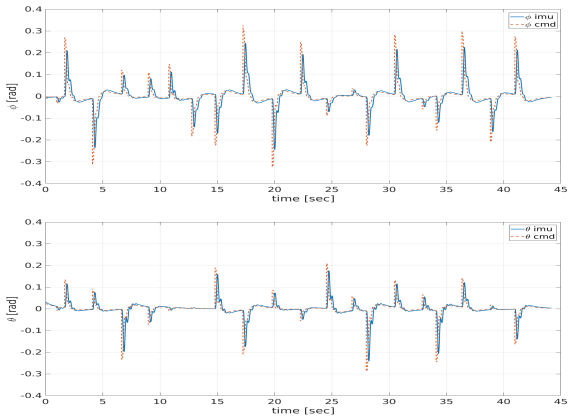

Fig. 5: Orientation tracking of the simulator model of Flame-Wheel 450 quadrotor. The blue line represents the orientation of the quadrotor and red line represents the commanded angles.

C. Approximating MPC using DNN

The supervised learning requires labelled data. We approximate the MPC by running it on the simulated quadrotor model. This has several benefits as it saves the time required to collect the training data by flying the real quadrotor. The flight time of the quadrotor with 3 cell battery was observed

to be around 15 min with Odroid running on it, which is quite low for data collection. Instead of changing/recharging batteries for multiple number of times, the data can be easily collected from the simulator model.

We collected data in simulator by sampling setpoints uniformly. We sampled the points in simulation as per the size of the flying arena. The size of the flying arena in our experimental setup is $4 \times 2 \times 1$ m, as shown in Fig. 2a. The states (s_t) and controls (u_t) at time t were recorded for simulator flight. We collected around 100000 data points for training.

The collected data from the simulator was used to learn the approximate MPC using DNN. The neural network consists of two fully connected hidden layer with 64 neurons in each layer. As the yaw is decoupled from the roll, pitch and thrust motion of the quadrotor, we train our network to predict the three controls: roll, pitch and thrust. We used Rectifier Linear Unit (ReLu) as the activation function in the hidden states. For the output layer we use *tanh* as the activation function. We minimize the huber loss between the predicted value and the ground truth value by using Adam Optimizer [40].

D. Comparison of MPC and DNN controllers

For a quantitative comparison of MPC and DNN controller we fly the quadrotor in different trajectories that can generalize the motions of the quadrotor. We select three types of trajectories for evaluation.

- Square: The square trajectory has both roll and pitch movements that are independent of each other. This helps in evaluating the roll and pitch outputs of the DNN and MPC independently. We define the trajectory as 1 $\times$ 1 meter square.
- ∞ Shape: This trajectory has both the roll and pitch motions simultaneously. This trajectory evaluates the effect of roll and pitch controls, when both are dependent on each other.
- Step Input: This trajectory evaluates the response time of the controller when step inputs are provided. We provide the setpoint at 30 cm apart. We create another instance of aggressive step input trajectories where setpoints are 1 m apart.

All the above three trajectories were generated by using Polynomial trajectory optimizer [41].

1) Trajectory Tracking: The trajectory tracking performance of the DNN controller is similar to the MPC. The tracking performance of the DNN for independent roll and pitch movements over the square trajectory is shown in Fig. 7. The DNN controller is able to respond to the aggressive step inputs and shows performance similar to the MPC as shown in Fig. 6. The DNN controller is able to understand the coupled dynamics of the roll and pitch. This can be seen by the ∞ shaped trajectory tracking shown in Fig. 8. The video of the trajectory tracking is provided in the supplementray material.

2) Position Error: We evaluate both the controllers based on the position error with respect to the reference position.

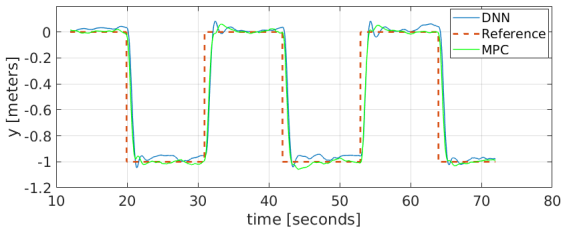

Fig. 6: Comparison of performance between MPC and DNN controller for step response for real-world f450.

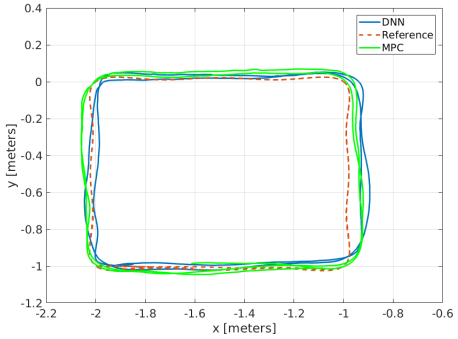

Fig. 7: Comparison of performance between MPC and DNN controllers for trajectory of 1 $\times$ 1 meter square for real-world f450.

We define the state cost as:

$$J_{pos} = \sum_{t=0}^{T} ||p(t) - p_{ref}(t)||^2$$

where $p(t)$ is the position at time t, $p_{ref}(t)$ the position reference. We consider the position error only as we generate the reference trajectory in terms of positions only. The overall performance of the DNN controller is comparable to the MPC as shown in Table I, in-spite of the fact that the controller is learned from simulated data and tested on a real-world quadrotor. The position error for the ∞ shaped trajectory by both the controller is shown in Fig. 9. The position cost in the Table I is calculated over a flight time of 1 minute.

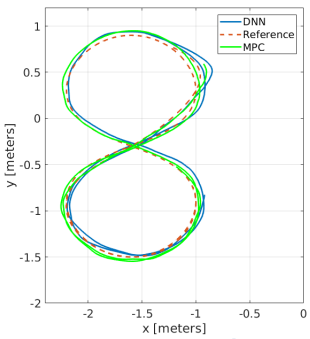

Fig. 8: Comparison of performance between MPC and DNN controllers for ∞-shaped trajectory for real-world f450.

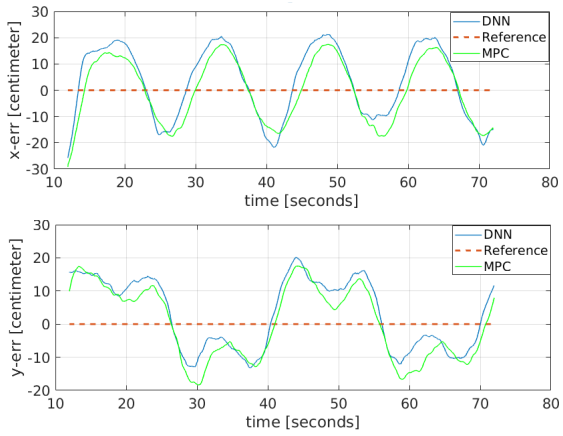

Fig. 9: Trajectory error for MPC and DNN controller for ∞ shaped trajectory shown in fig.[8]

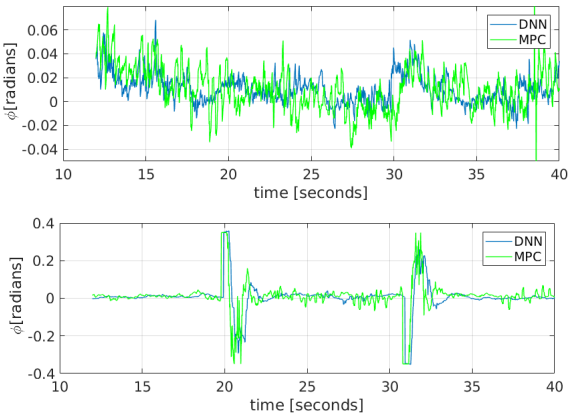

Fig. 10: Comparison of Controls generated by MPC and DNN for ∞ shaped trajectory and aggressive trajectory (step input). It can be observed that the controls generated by the DNN are smoother than the MPC. It seems that DNN approximates a mean of the the controls generated by MPC for similar states.

3) Control Cost: The control cost has a direct effect on the motors efficiency. High control cost indicates that the motors are stressed by the controller as it generates abrupt changes in the input. We define control cost for a trajectory as:

$$J_{ctrl} = \sum_{t=0}^{T} ||u(t)||^2$$

where $u(t)$ is the control generated at time t.

The control cost of the the DNN is observed to be lower than the MPC as shown in Table I. This is because of the smooth controls generated by DNN as shown in Fig. 10 for ∞ shape trajectory and step inputs. The DNN gives smooth output as compared to MPC due to the fact that during supervised learning, the DNN tires to minimize the squared error (Huber loss is used in training). This results in a smooth curve fitting by DNN over the set of the controls generated by MPC.

Trajectory	MPC		DNN	
	State Cost	Control Cost	State Cost	Control Cost
Square	6.60	0.29	7.56	0.24
∞ shape	16.79	0.71	18.22	0.62
Step Input	28.26	5.43	31.90	4.59

TABLE I: Comparison of Trajectory cost between MPC and DNN Controller for real-world f450.

Trajectory	Flight Time[secs]		% Gain
	MPC	DNN	
Hover	960	1044	8.75 %
Step Input	922	1020	10.6 %
Aggressive Step Input	902	1015	12.5 %

TABLE II: Comparison of Flight time between MPC and DNN Controller for real-world f450

4) Flight Time: We observe that the flight time of the quadrotor has increased as shown in Table II, when we use the DNN controller, as the controls generated by the DNN are smooth and the optimization problem that is being solved by the MPC is replaced with simple matrix operations. The gain in the flight time is more for aggressive trajectories due to the smoother controls generated by the DNN. The hardware required to run the DNN controller is nearly 20% the cost of the hardware required for MPC.

E. Processor Consumption

We compare the computation cost of the MPC and DNN controller on a single core ARM Cortex A7 processor. The MPC solves an online optimization problem by using Acado solver [42] to generate controls. The DNN controller performs matrix arithmetic to calculate the control. The comparison is shown in Table III with both the controllers implemented in C++. For DNN, we use Eigen library [43] to perform the matrix arithmetic.

It is also worth noting that the processor hardware required to run the DNN controller is nearly 20% of the cost of the processor hardware required for MPC. The `Raspberrypi 3` used for DNN controller has ARMv7 Processor on it with 4 cores.

V. Conclusion

In this paper, we demonstrate the possibility of using a Deep Neural Network to approximate a Model Predictive Controller for nonlinear dynamical systems like quadrotor. We observe that by modelling the low level dynamics of the quadrotor in simulation, we are able to learn the DNN controller for a real-world quadrotor. The performance analysis indicates that the DNN controller increases flight time as it can run on a less powerful processor, thus consumes less CPU power, but yet it is able to track the trajectory in an efficient manner. The results indicate that the Neural Networks are exceptional at generalizing the nonlinear dynamics and have potential to become an alternative to classical controllers.

In future, we wish to analyze the effect of the number of layers in the Neural Network on the performance of the feedback control. There is a further scope of improvement in the performance of the DNN by applying Reinforcement

Trajectory	MPC	DNN
Hover	25 %	3 %
∞ shaped	29 %	4%
Step Input	33 %	4%

TABLE III: Comparison of CPU usage on a single core ARM Cortex A7 processor.

Learning based techniques [44] on the learned controller. Finally, we would like to apply the recently developed Lyapnov based analysis for robustness of controller to perform safe learning [45].

REFERENCES

[1] A. Visioli. *Practical PID Control*. Advances in Industrial Control. Springer London, 2006.

[2] Carlos E. Garcia, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice a survey. *Automatica*, 25(3):335 – 348, 1989.

[3] Frank Allgöwer, Rolf Findeisen, and Zoltan K. Nagy. Nonlinear model predictive control: From theory to application. *J. Chin. Inst. Chem. Engrs.*, 35(3):299–315, 2004.

[4] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pages 1–10. Citeseer, 2013.

[5] P. Tondel, T.A. Johansen, and A. Bemporad. An algorithm for multi-parametric quadratic programming and explicit mpc solutions. In *CDC*, pages 1199–1204, 2001.

[6] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[7] Patrick Bouffard. On-board model predictive control of a quadrotor helicopter: Design, implementation, and experiments. Technical report, University of California Berkeley, 2012.

[8] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.

[9] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016.

[10] Rong Xu and Ümit Özgüner. Sliding mode control of a quadrotor helicopter. In *45th IEEE Conference on Decision and Control (CDC)*, page 49574962, 2006.

[11] Ilolger Voos. Nonlinear control of a quadrotor micro-uav using feedback-linearization. In *IEEE International Conference on Mechatronics*, page 16, 2009.

[12] Guilherme V Raffo, Manuel G Ortega, and Francisco R Rubio. An integral predictive/nonlinear h control structure for a quadrotor helicopter. *Automatica*, 46(1):29–39, 2010.

[13] C. Nicol, C.J. B. Macnab, and A. Ramirez-Serrano. Robust adaptive control of a quadrotor helicopter. *Mechatronics*, 21(6):927–938, 2011.

[14] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 2520–2525, 2011.

[15] Somil Bansal, Anayo K. Akametalu, Frank J. Jiang, Forrest Laine, and Claire J. Tomlin. Learning quadrotor dynamics using neural network for flight control. In *55th IEEE Conference on Decision and Control (CDC)*, pages 4653–4660, 2016.

[16] Qiyang Li, Jingxing Qian, Zining Zhu, Xuchan Bao, Mohamed K. Helwa, and Angela P. Schoellig. Deep neural networks for improved, impromptu trajectory tracking of quadrotors. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pages 5183–5189, 2017.

[17] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[18] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[19] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, pages 528–535, 2016.

[20] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas, and M. Morari. Approximating explicit model predictive control using constrained neural networks. In *2018 Annual American Control Conference (ACC)*, pages 1520–1527, June 2018.

[21] Randal Beard. Quadrotor dynamics and control rev 0.1. 2008.

[22] James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, 58(15-16):1–35, 2006.

[23] Simon Haykin. *Neural networks*, volume 2. Prentice hall New York, 1994.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[25] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[27] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.

[28] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[29] Peter J Huber et al. Robust estimation of a location parameter. *The annals of mathematical statistics*, 35(1):73–101, 1964.

[30] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[31] Karl Johan Aström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

[32] Mina Kamel, Thomas Stastny, Kostas Alexis, and Roland Siegwart. Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system. In Anis Koubaa, editor, *Robot Operating System (ROS) The Complete Reference, Volume 2*. Springer.

[33] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6235–6240, 2015.

[34] ODROID XU4. Website. [Online] `https://www.hardkernel.com`.

[35] MAVlink (2014) MAVlink micro air vehicle protocol. Website. [Online] `http://mavlink.org`.

[36] VICON motion capture system. Website. [Online] `http://www.vicon.com/`.

[37] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[38] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, 2004.

[39] Matt Schmittle, Anna Lukina, Lukas Vacek, Jnaneshwar Das, Christopher P. van Buskirk, Stephen Rees, Janos Sztipanovits, Radu Grosu, and Vijay Kumar. Openuav: A uav testbed for the cps and robotics community. *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 130–139, 2018.

[40] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016.

[42] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. ACADO toolkit-an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, may 2010.

[43] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[44] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[45] Spencer M Richards, Felix Berkenkamp, and Andreas Krause. The lyapunov neural network: Adaptive stability certification for safe learning of dynamic systems. *arXiv preprint arXiv:1808.00924*, 2018.