Neural Enhanced Control for Quadrotor Linear Behavior Fitting

Esteban Carvalho, Pierre Susbielle, Ahmad Hably, Jilles Dibangoye, Nicolas Marchand

Neural Enhanced Control for Quadrotor Linear Behavior Fitting

Estéban Carvalho[1,2], Pierre Susbielle[1], Ahmad Hably[1], Jilles S. Dibangoye[2] and Nicolas Marchand[1]

***Abstract*— Designing an efficient autopilot for quadrotor can be a very long and tedious process. This comes from the complex nonlinear dynamics that rule the flying robot behavior as battery discharge, blade flapping, gyroscopic effect, frictions, etc. In this paper we propose to use a traditional cascaded control architecture enhanced with Deep Neural Network (DNN). The idea is to easily setup a control algorithm using linear cascaded laws and then correct unmodelled dynamics and approximations made during the linear control design with the DNN. The tuning process is reduced to choice of proportional and derivative gains of each control loop. The approach is tested in the ROS/Gazebo simulation environment and experimentally in a motion capture room. Results confirm that the methodology significantly improves the performance of linear approaches on nonlinear quadrotor system.**

Supplementary Material

Video: youtu.be/c70nlsMVi9M
Code: github.com/gipsa-lab-uav/uav_control_ai

I. Introduction

In the last decade, unmanned aerial vehicles (UAVs) have become predominant worldwide. From leisure film making to public transport, surveillance, monitoring or even geographic mapping, their area of application grows continuously. The flight navigation is usually done manually, which requires a minimum of training to pilot UAVs. A key component to make UAVs available for all is an automatic flight control reactive to unexpected situations and environments.

A functional autopilot consists of two elements: a trajectory generation [1, 2] and a controller [3, 4]. On the first hand, a fast trajectory generation algorithm is crucial to maintain good flight performances. It should be able to face unpredicted obstacles and respond in real time. As a result, [1, 2] propose a polynomial path generator that aims rapid and feasible trajectory generation, with lowcost computation and complexity. On the other hand, the controller tracks the desired trajectory. Research of [3] introduce an accurate quadrotor model that fits the system dynamic and thus improves flight performances. However, it induces at the same time a nonlinear controller which leads complex to tune algorithm as well as computation consuming systems. Although a linear model is easier to handle, a popular approach consists in linearizing a nonlinear model to avoid complexity [5].

*This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) and ROBOTEX 2.0 (Grants ROBOTEX ANR-10-EQPX-44-01 and TIRREX ANR-21-ESRE-0015) funded by the French program Investissements d'Avenir.

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-Lab, F-38000, Grenoble, France; email: firstname.lastname@gipsa-lab.fr

[2] Univ. Lyon, INSA Lyon, CITI, F-69621, Villeurbanne, France; email: firstname.lastname@inria.fr

With the advances in artificial intelligence, modelling and control are currently evolving. As presented in survey [6], combining machine learning techniques with traditional control approaches is improving systems' performances. Data-driven methods achieve suitable modelling of errors, uncertainties and perturbations. For instance, [7] highlights the benefits of Convolutional Neural Networks (CNN) in a fast driving car control. Works [8]–[10] focus on UAVs hybrid approaches. In [10], Gaussian processes are introduced to model disturbances which are then combined to a predictive model control. This greatly reduces tracking errors at high speeds. Authors of [8] propose a Deep Neural Network (DNN) to generate a reference trajectory. It shows a gain in tracking with smaller errors while keeping classical PID controller. Finally, a predictive control for racing drone is presented in [9]. The vehicle flies through moving gates. Gates' position is predicted by a CNN and sent to the control. With this algorithm, the team managed to beat the second team in half the time.

One may use neural networks to model system dynamics, as shown in [11], and then apply the learned model to adapt a control law. [12, 13] have shown the relevance of using learning on modelling complex aerodynamics phenomena usually neglected, such as vibrations, for helicopter flight control. Based on those advances, [14] came up with a shallow neural network (NN) that model UAVs dynamics. A more recent study [15] applies a DNN to model ground effects and proposes a DNN-based controller that enhances the landing behavior, achieving impressive results.

The problem of getting an efficient controller that fits some given expected behavior can also be addressed by using reinforcement learning (RL) methods. RL algorithms for drone control is developing a lot, see [16, 17]. However, these contributions fall short in that they need a huge amount of data. In addition, the data collected must be relevant to learn a controller and thus requires a large exploration process, usually done randomly. Hence for high order systems like UAVs, it requires a lot of data and time. Secondly, those unsupervised method do not provide stability guaranty for the system which is very problematic. To overcome the data collection and time problem, as well as the stability issue, we propose an hybrid control approach based on a linear control and a neural network that describes nonlinear error dynamics. Our approach allows to easily setup a controller that will fit expected behavior. In order to reduce the design complexity and to preserve stability, a linear cascaded control is used. A Deep Neural Network is added for fast performance enhancement using collected data. It will learn the error between the linear model and the real system dynamics, both

controlled with the linear law. Fitting an a priori given linear behaviour is especially of interest for obstacle avoidance. Indeed, it is far easier to predict the error between a reference trajectory and the real one when the system is linear. It also eases a fast trajectory generation. The DNN will not only model a complex and a nonlinear dynamic error, but also includes parametric errors. The error model identified by the DNN is then injected in the real time command so that the quadrotor behavior fit to the linear expected dynamics. Proposed methodology is summarized in fig.1.

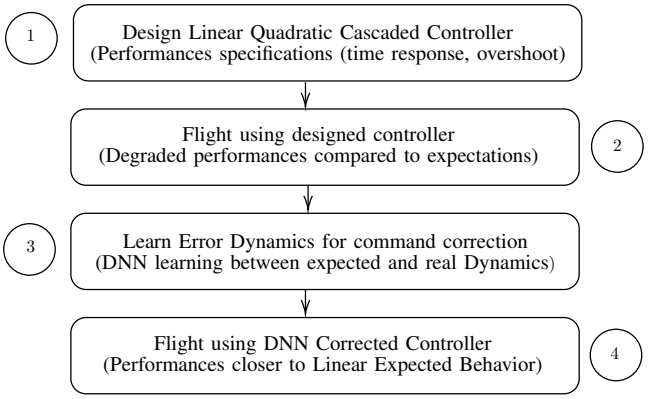

Fig. 1: Proposed methodology to fit expected linear behavior on the quadrotor nonlinear dynamics using DNN correction.

Main contributions. In contrast to prior research, the contributions of this paper are threefold:

- Learning the error between the real system dynamics and a linear dynamical model.
- We consider a linear control applied to a complex system, avoiding nonlinear controls, and correcting the resulting error to refit the linear expected quadrotor dynamics.
- We show good results in simulation and in real time experimental flights and propose an open source ROS package.

The paper is organized as follows. Section II defines the quadrotor dynamics and its linearized model used for control design. Section III presents a neural architecture to model the error dynamics. Proposed linear cascaded controller and DNN correction law are given in Section IV. We expose simulation and experimental results in Section V, including neural networks training and controller's tests. Finally, conclusion and future work perspectives are given in Section VI.

II. Quadrotor dynamics

Quadrotors are aerial vehicles built with two counter-rotating rotors located on a rigid cross-shaped body. Translations on z axis are achieved by decreasing or accelerating the four rotor speeds at constant thrust for each of them. Translations on the x or y axis are obtained by adjusting the pitch or roll by applying more thrust to the two adjacent rotors and less to the two opposite ones. Finally, yaw is achieved by applying more thrust to the rotors rotating in the desired direction and less to the other two. In this section, the quadrotor's Newton-Euler dynamic and the linear model used for control purposes in Section IV are presented.

A. Quadrotor Dynamics

As a starting point, $\{I\}$ defines the inertial frame described by its units vectors $\{\vec{i_1}, \vec{i_2}, \vec{i_3}\}$ and $\{B\}$ represents the body fixed frame described by its units vectors $\{\vec{b_1}, \vec{b_2}, \vec{b_3}\}$. $R \in SO(3)$ defines the orientation of the body frame $\{B\}$ in $\{I\}$ using Z-Y-X Euler formalism(see Fig. 2):

$$\begin{aligned} \boldsymbol{R} &= \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\phi) \\ &= \begin{bmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \end{aligned} \tag{1}$$

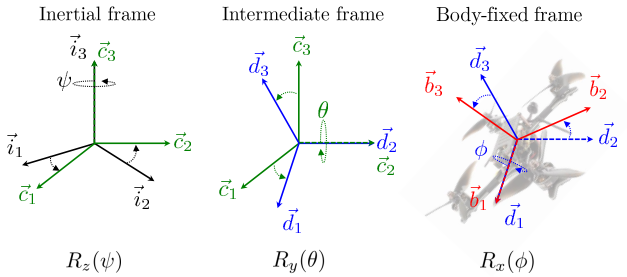

Fig. 2: Euler angles references. Inertial frame (left), intermediate frame (middle) and body fixed frame (right).

where $c. := \cos(\cdot)$ and $s. := \sin(\cdot)$.

Let $\xi := (x, y, z)^T \in \{I\}$ be the position of the drone center of mass, $v := (v_x, v_y, v_z)^T \in \{I\}$ its linear velocity expressed in $\{I\}$, $\Omega := (p, q, r)^T$ the angular velocity expressed in $\{B\}$ and $\zeta := (\phi, \theta, \psi)$ denotes Euler angles. The total mass of the vehicle is given by m and the inertia matrix $\boldsymbol{J}$ assumed diagonal due to the symmetry of the quadrotor. The quadrotor rigid body dynamics are given by Newton-Euler equations [18]:

$$\begin{cases} \dot{\xi} = v, & \text{(2a)} \\ m\dot{v} = -mg\vec{i_3} + \boldsymbol{R}T\vec{b_3} + \delta^v_{nd}, & \text{(2b)} \\ \dot{\boldsymbol{R}} = \boldsymbol{R}\Omega^\times, & \text{(2c)} \\ \boldsymbol{J}\dot{\Omega} = -\boldsymbol{\Omega}^\times \boldsymbol{J}\Omega + \Gamma + \delta^\Omega_{nd}, & \text{(2d)} \end{cases}$$

where T is the total thrust generated by the four rotors, $\Omega^\times$ is a skew-symmetric matrix associated to Ω that verifies for all matrix $\boldsymbol{M} \in M_3(\mathbb{R})$, $\boldsymbol{M} \times \Omega = M\boldsymbol{\Omega}^\times$, and Γ represents the torques applied to the rotors, and δ^v_{nd} and δ^Ω_{nd} the unmodelled and unknown dynamics on linear and angular acceleration respectively. It will be referred as error dynamics in the following. These error dynamics include for instance the gyroscopic effects, ground effects, blade flapping or battery discharge.

In the following sections, we assume that the angular loop of the quadrotor is controlled, with a sufficient speed for the other cascaded control loops. As a result, the model to be

controlled is given by:

$$\begin{cases} \dot{\xi} = v, & \text{(3a)} \\ m\dot{v} = -mg\vec{i_3} + \boldsymbol{R}T\vec{b_3} + \delta_{nd}^v, & \text{(3b)} \\ \dot{\zeta} = \boldsymbol{W^{-1}}(\phi,\theta,\psi)\Omega, & \text{(3c)} \end{cases}$$

where:

$$\boldsymbol{W^{-1}}(\phi,\theta,\psi) = \begin{bmatrix} 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi/c_\theta & c_\phi/c_\theta \end{bmatrix}. \quad (4)$$

Summarizing previous equations, the quadrotor dynamics can be expressed as (5):

$$\dot{X} = f_{X4}(X,U) + \delta_{nd}, \quad (5)$$

where X is the state vector defined by $X := \begin{bmatrix} \xi & v & \zeta \end{bmatrix}^T$, the control vector $U := \begin{bmatrix} T & p & q & r \end{bmatrix}^T$, the error dynamics vector $\delta_{nd} := \begin{bmatrix} \mathbb{0}_3^T & \frac{\delta_{nd}^v}{m}^T & \mathbb{0}_3^T \end{bmatrix}^T$, with $\mathbb{0}_3 := \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ and:

$$f_{X4}(X,U) = \begin{bmatrix} v \\ -g\vec{i_3} + \frac{RT\vec{b_3}}{m} \\ T_r(\phi,\theta,\psi)\Omega \end{bmatrix}. \quad (6)$$

B. Quadrotor Linear Dynamics Model

The proposed control law is derived from a linearized model around given equilibrium points. The quadrotor linear model usually used for control design is obtained from hover conditions. Equilibrium points are given by (7) and (8):

$$X_{eq} = \begin{bmatrix} x_e & y_e & z_e & \mathbb{0}_3 & \mathbb{0}_3 \end{bmatrix}^T, \quad (7)$$

$$U_{eq} = \begin{bmatrix} mg & \mathbb{0}_3 \end{bmatrix}^T. \quad (8)$$

We can then define variation variables $\tilde{X} := X - X_{eq}$ and $\tilde{U} := U - U_{eq}$. Assuming δ_{nd} is negligible, $\delta_{nd}^v \approx \mathbb{0}_3$, the linear model obtained using first order Taylor expansion is given by:

$$\begin{aligned} \dot{\tilde{X}} &= \begin{bmatrix} \boldsymbol{0_3} & \boldsymbol{I_3} & \boldsymbol{0_3} \\ \boldsymbol{0_3} & \boldsymbol{0_3} & \boldsymbol{A_{2,3}} \\ \boldsymbol{0_3} & \boldsymbol{0_3} & \boldsymbol{0_3} \end{bmatrix}\tilde{X} + \begin{bmatrix} \boldsymbol{0_{34}} \\ \boldsymbol{B_2} \\ \boldsymbol{B_3} \end{bmatrix}\tilde{U}, \\ \dot{\tilde{X}} &= \boldsymbol{A}\tilde{X} + \boldsymbol{B}\tilde{U}, \end{aligned} \quad (9)$$

where:

$$\boldsymbol{A} = \frac{\partial f_{X4}}{\partial X}(X,U)\Big|_{(X_{eq},U_{eq})}, \boldsymbol{B} = \frac{\partial f_{X4}}{\partial U}(X,U)\Big|_{(X_{eq},U_{eq})},$$

0_3 refers to the square null matrix of order 3 and $\boldsymbol{I_3}$ the identity matrix of order 3. $\boldsymbol{0_{34}}$ is a null matrix composed of 3 lines and 4 columns. $\boldsymbol{A_{2,3}}$, $\boldsymbol{B_2}$ and $\boldsymbol{B_3}$ matrices are expressed as follow:

$$\boldsymbol{A_{2,3}} = \begin{bmatrix} 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \boldsymbol{B_2} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ m^{-1} & 0 & 0 & 0 \end{bmatrix}, \boldsymbol{B_3} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

III. Deep Neural Network Structure

This section presents the proposed structure for the Deep Neural Network. Networks are used to describe the error dynamics defined by eq. (10), corresponding to the unknown dynamics between the real drone dynamics and the linear acceleration dynamics (δ_e^v).

$$\delta_e^v(\zeta,T) = \dot{v} - (\boldsymbol{A_{2,3}}\zeta + \boldsymbol{B_2}\tilde{U}), \quad (10)$$

According to the universal approximation theorem, one can build a feed-forward network composed of only one hidden layer with a finite number of units to approximate our continuous functions δ_e^v given by (10). Here, it is proposed to use a deep neural network to better fit the data. Based on the nature of the problem, here regression of temporal data, the use of this type of network is well suited.

The activation function used for all hidden layers is the rectified linear unit (ReLU) defined element-wise by $h(\cdot) = max(\cdot, 0)$. Weights and bias are defined as follows: $\boldsymbol{W_{v,1}} \in \mathbb{R}^{\#E_v, N_v}$, $\boldsymbol{W_2} \in \mathbb{R}^{N_u, N_u}$, $\boldsymbol{W_3} \in \mathbb{R}^{N_u, \#S_v}$ $\boldsymbol{M_1} \in \mathbb{R}^{N_u}$, $\boldsymbol{M_2} \in \mathbb{R}^{N_u}$ and $M_3 \in \mathbb{R}^{\#S_v}$. N_u is the number of units of the hidden layer of the network and using the following notation $\#A$ denotes the cardinal of A (fig. 3). E_{DNN} denotes the neural network inputs. Our neural network structure can then be expressed as (11):

$$\hat{\delta}_{nd}^v(E_{DNN}) = \boldsymbol{W_3}^T h(\boldsymbol{W_2}^T h(\boldsymbol{W_1}E_{DNN} + \boldsymbol{M_1}) + \boldsymbol{M_2}) + \boldsymbol{M_3}. \quad (11)$$

We choose $N_l = 2$ for the number of hidden layers and $N_u = 64$ for the number of hidden layers units, as it is sufficient to correctly model the error dynamics and to perform real-time computation. A quick ablation study revealed that for one layer DNN, the network has difficulty to converge and for three layers it does not bring better performances while increasing the learning process time.

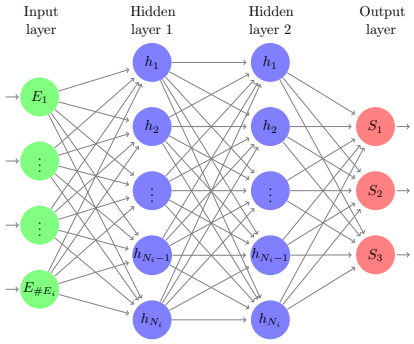

Fig. 3: Deep Neural Network Architecture: a feed-forward network, with two hidden layers of 64 units, with ReLU function as activation.

To learn error accelerations dynamics one aims at finding weights and bias that minimize the mean squared error (MSE) between predictions and observations. Predictions are expected values from the neural network model and observations are the values obtained via measurements. Accelerations measurements are collected from the inertial measurement unit (IMU) of the quadrotor.
Take $\vartheta := (\boldsymbol{W_1}, \boldsymbol{W_2}, \boldsymbol{W_3}, \boldsymbol{M_1}, \boldsymbol{M_2}, \boldsymbol{M_3})$ the parameter to be optimized. Then, training the neural network (11) aims at solving the following optimization problem (12) over the

training data-set:

$$\min_{\vartheta} \mathcal{L}(\vartheta) = \min_{\vartheta} \frac{1}{N} \sum_{k=1}^{N} \|\hat{\delta}^{v}_{e,k}(\vartheta) - \delta^{v}_{e,k}\|_2, \tag{12}$$

where N is the number of elements in the data-set and $\delta^{v}_{e,k}$ are error dynamics computed by taking measurements outputs and commands inputs. Here $\mathcal{L}(\vartheta)$ is the objective function to be minimized, referred as loss function. This problem can be typically solved using traditional gradient descent methods using back-propagated gradients optimizers.

IV. Quadrotor Control

In this section, we describe the cascaded controllers designed to track a given trajectory reference (See fig. 1, step 1). A linear cascaded architecture is used for tuning simplicity and ease of implementation. This control will be next enhanced with a deep neural network feedforward correction term. The DNN correction term aims to get closer to the linear expected behavior. This term is directly designed from the previously built neural network. Thus, one is able to rectify unmodelled dynamics and linearization errors, that are learned dynamics with training data.

A. Linear cascaded controller

The proposed control law is divided into two linear cascaded controllers: the attitude and the position/speed control loops.

Attitude controller: Using (9), it comes that: $\dot{\zeta} = \Omega$. We can directly choose:

$$\Omega = -\boldsymbol{K_\zeta} \left(\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} - \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}_{ref} \right), \tag{13}$$

such that we reach the desired closed loop behavior with $\boldsymbol{K_\zeta}$. Where gain $\boldsymbol{K_\zeta}$ is obtained through a linear quadratic regulation synthesis (LQR) and $\begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T_{ref}$ is the attitude target coming from the position and speed controller.

Position Speed controller: Let us define $u_\zeta := \begin{bmatrix} \phi & \theta & T \end{bmatrix}^T$, $x_1 := \begin{bmatrix} x & y & z \end{bmatrix}^T$ and $x_2 := \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}^T$. Then using (9):

$$\begin{cases} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= \begin{bmatrix} 0 & g & 0 \\ -g & 0 & 0 \\ 0 & 0 & m^{-1} \end{bmatrix} u_\zeta = \boldsymbol{A_\zeta} u_\zeta. \end{cases} \tag{14}$$

Defining $\eta_1 := x_1 - x_{1ref}$ and $\eta_2 := \dot{\eta}_1 := \dot{x}_1 - \dot{x}_{1ref} = x_2 - x_{2ref}$, (14) can be rewritten as:

$$\dot{\eta} = \begin{bmatrix} \boldsymbol{0_3} & \boldsymbol{I_3} \\ \boldsymbol{0_3} & \boldsymbol{0_3} \end{bmatrix} \eta + \begin{bmatrix} \boldsymbol{0_3} \\ \boldsymbol{A_\zeta} \end{bmatrix} (u_\zeta - \boldsymbol{A_\zeta^{-1}} \dot{x}_{2ref}). \tag{15}$$

$$\overline{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{0_3} & \boldsymbol{I_3} \\ \boldsymbol{0_3} & \boldsymbol{0_3} \end{bmatrix}, \overline{\boldsymbol{B}} = \begin{bmatrix} \boldsymbol{0_3} \\ \boldsymbol{A_\zeta} \end{bmatrix}$$

For (15), we propose to use again a linear quadratic regulator:

$$u^*_\zeta = -\boldsymbol{K_\eta} \eta + \boldsymbol{A_\zeta} \dot{x}_{2_{ref}}, \tag{16}$$

where $\boldsymbol{K_\eta}$ is chosen such that we minimize a quadratic cost expressed as follows:

$$J(u_\zeta) = \int_0^{+\infty} (\eta^T(t)\, \boldsymbol{Q}\, \eta(t) + u_\zeta^T(t)\, \boldsymbol{R}\, u_\zeta(t)) dt.$$

Weighting matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ are tuned to reach desired tracking performances.

B. DNN Correction Controller

According to eq. (2b), the real system dynamics can be written as follows:

$$\dot{\eta} = \overline{\boldsymbol{A}} \eta + \overline{\boldsymbol{B}} (u_\zeta - A_\zeta^{-1} \dot{x}_{2ref} + \boldsymbol{A_\zeta}^{-1} \delta^{v}_{e}). \tag{17}$$

The DNN controller is designed as follows:

$$u_\zeta = u^*_\zeta + \Delta u_\zeta, \tag{18}$$

where u^*_ζ is the linear quadratic control (16) and Δu_ζ is the feedforward correction term based on the neural network. The closed loop form of (17) using (18) is:

$$\dot{\eta} = \underbrace{(\overline{\boldsymbol{A}} - \overline{\boldsymbol{B}} \boldsymbol{K_\eta}) \eta}_{\text{Close loop linear behavior}} + \underbrace{\overline{\boldsymbol{B}} (\Delta u_\zeta + \boldsymbol{A_\zeta^{-1}} \delta^{v}_{e})}_{\text{Term to minimize}}. \tag{19}$$

According to (19), the closed loop behavior is the sum of two terms. The first one describes the desired linear behavior, which is fully controlled. The second term needs to be canceled or at least minimized to get closer to the expected linear behavior. This procedure outlines a classical minimization problem (20):

$$\min_{\Delta u_\zeta} \Delta u_\zeta + \boldsymbol{A_\zeta^{-1}} \delta^{v}_{e}(\Delta u_\zeta). \tag{20}$$

Replacing error dynamics by their approximations from the DNN we get:

$$\min_{\Delta u_\zeta} \Delta u_\zeta + \boldsymbol{A_\zeta^{-1}} \hat{\delta}^{v}_{nd}(\Delta u_\zeta). \tag{21}$$

As $\hat{\delta}^{v}_{nd}$ depends on Δu_ζ according to (21), it is a nonlinear minimization problem, that may be time consuming to solve. To deal with it, a proposed solution is to replace u_ζ by u^*_ζ, assuming $u_\zeta \ll u^*_\zeta$. By removing the correction term from the DNN input we can directly deduce the correction term:

$$\Delta u_\zeta = -\boldsymbol{A_\zeta^{-1}} \hat{\delta}^{v}_{nd}(u_\zeta). \tag{22}$$

Knowing the drone mass, we can get the correction to apply to the thrust reference T_{ref} and to the two angular references ϕ_{ref} and θ_{ref} to get closer to the linear behavior. The control architecture is summarized in fig. 4. The correction proposed is only done in the position control loop, as attitude control is done considering (3c), which simply corresponds to the rotation of the frame.

C. Stability Analysis

We derive the main lines here to prove that the deep neural network term does not destabilize the quadrotor. Let us consider the following Lyapunov Function:

$$V(t) = \frac{1}{2} \|\eta(t)\|^2. \tag{23}$$

Defining the prediction error of the DNN as $e_\eta = \delta_e^v - \hat{\delta}_{nd}^v$, and $\lambda = \lambda_{min}(\overline{\boldsymbol{A}} - \overline{\boldsymbol{B}}\boldsymbol{K_\eta})$ be the minimum eigenvalue of the closed loop system matrix.

$$\begin{aligned} \dot{V}(t) &= \eta^T(\overline{\boldsymbol{A}} - \overline{\boldsymbol{B}}\boldsymbol{K_\eta})\eta + \eta^T e_\eta, \\ &\leq \lambda\eta^T\eta + \|\eta\| \cdot \|e_\eta\|, \\ \dot{V}(t) &\leq 2\lambda V + \sqrt{2V}\,\|e_\eta\|. \end{aligned} \tag{24}$$

Assuming $\|e_\eta\|$ is bounded by a constant ε_η and defining $W = \sqrt{V}$ then $\dot{W} = \frac{\dot{V}}{2\sqrt{V}}$ it follows that:

$$\dot{W} \leq \lambda W + \frac{\varepsilon_\eta}{\sqrt{2}}. \tag{25}$$

The solution to (25) given $W(0) = \frac{z_0}{\sqrt{2}}$ is given by (26):

$$W(t) \leq \frac{1}{\sqrt{2}}(\frac{\varepsilon_\eta}{\lambda} + \|z_0\|)e^{\lambda(t-t_0)} - \frac{\varepsilon_\eta}{\sqrt{2}\lambda}. \tag{26}$$

We obtain the following bound:

$$\|\eta(t)\| \leq \|\eta_0\|\, e^{\lambda(t-t_0)} + \frac{\varepsilon_\eta}{\lambda}(e^{\lambda(t-t_0)} - 1). \tag{27}$$

It follows that the solution exponentially converges to a ball, centered at η_0 and with a radius of $\frac{\varepsilon_\eta}{\lambda}$. It is directly proportional to the error made by the neural network.

V. Simulation and Experimental Results

In this section, DNN training and validation are explained (Section V-A). Controllers are next tested in ROS/Gazebo simulation environment (Section V-B) and in experimental flights (Section V-D) in our motion capture room.

We work with ROS combined with Gazebo simulation environment to validate our methodology. To perform angular velocity and low level control, PX4 autopilot is used [19]. Python along with the Keras API [20] are used to shape data collected from ROS, named rosbag, to build and to train the DNN. The code used for the simulations and experiments is made open access and available by following the given GitHub link.

A. Training and Validation Database

The proposed Deep Neural Network in Section III is trained with a database created using the linear cascaded control from Section IV-A, for both simulation and experimental tests (fig. 1, step 2 & 3). It is composed of several basic movements as translations in different planes, steps in all directions, rotations and spiral motions (see fig. 5). The idea is to cover all the possible motions of the drone, to better learn the whole dynamic.

The entire state X, the control inputs U and the linear accelerations $\ddot{X}$ are supposed to be accessible, either by measurements (IMU) or obtained using filters like Extended Kalman Filter (EKF). To validate learned model, one-third of the database is used as validation set. Training and validation sets cover a total of 30 minutes of flight. We propose here to fly with less data as possible to show that we can quickly and correctly learn the error dynamics.

For training process, we used Nesterov Adam optimizer also known as Nadam (Reader is referred to [21] for more information about the algorithm). The following hyper-parameters are set for the DNN training: Learning rate $l_r = 0.05$, $\mu = 0.9$, $\nu = 0.999$, batches of size 256 are used to evaluate gradients and a maximal number of 1000 epochs is defined. An epoch refers to the number of times the algorithm has been run through the whole training data-set. In addition, during training process elements are shuffled and the learning rate is decreased by a factor of $\frac{2}{3}$ when there is no improvement on the loss function. All those hyper parameters were tuned empirically in order to reach a good compromise between speed and convergence of the algorithm.

We propose to include knowledge of the quadrotor dynamics to the DNN eq. (11) by restricting the input variables, using eq. (2b): one provides thrust, Euler angles and linear velocities. Sine and cosine of angles are directly provided to ensure that for the DNN 0 and 2π are equivalent, in the same way as in [14]. Inputs and outputs of the DNN are given by eq. (28) and (29):

$$E_{DNN} = \begin{bmatrix} z & c_\phi & s_\phi & c_\theta & s_\theta & v_x & v_y & v_z & u_{batt} & T \end{bmatrix}^T, \tag{28}$$

$$S_{DNN} = \begin{bmatrix} \ddot{x} & \ddot{y} & \ddot{z} \end{bmatrix}^T. \tag{29}$$

In order to better tackle the ground effect, z component is added to the neural network inputs. The same is applied for battery voltage to handle battery discharge and loss of thrust. Other position components are not included as they do not appear in eq. (2b).

B. Simulations Results

We show results of the cascaded linear controller and it enhanced version with the DNN correction term in ROS/Gazebo simulation environment (fig. 1, step 4). Presented results are given on an unseen trajectory reference, that is to say that they are neither part of the training set nor part of the validation set, see fig. (6). A third controller, a cascaded linear controller with integral action term is also presented for comparison. In order to be compared correctly with the two other controllers presented, the gains of the controller with integral action are chosen in such a way that the closed loop behavior approaches the ideal behavior without integrator, i.e. the behavior of the LQR controller. Thus the gains proportional, integral and derivative gains are chosen so that the mean squared error between the integral LQR and the non-integral LQR is the lowest. To find these gains a brute force approach was used, requiring several hours of calculations.

When the linear cascaded law is used, reached positions reveal tracking errors and slower time response (blue line). Tracking errors can be explained by the absence of integrator in the initial linear controller. Slower time response is due to the linear control design on a nonlinear model. On tests trajectories, tracking errors with the linear controller are around 7 cm on x and y component and for z component it is up to 25 cm.

However, we expected none of those effects if the drone behaves exactly like its linear model (green dashed line).

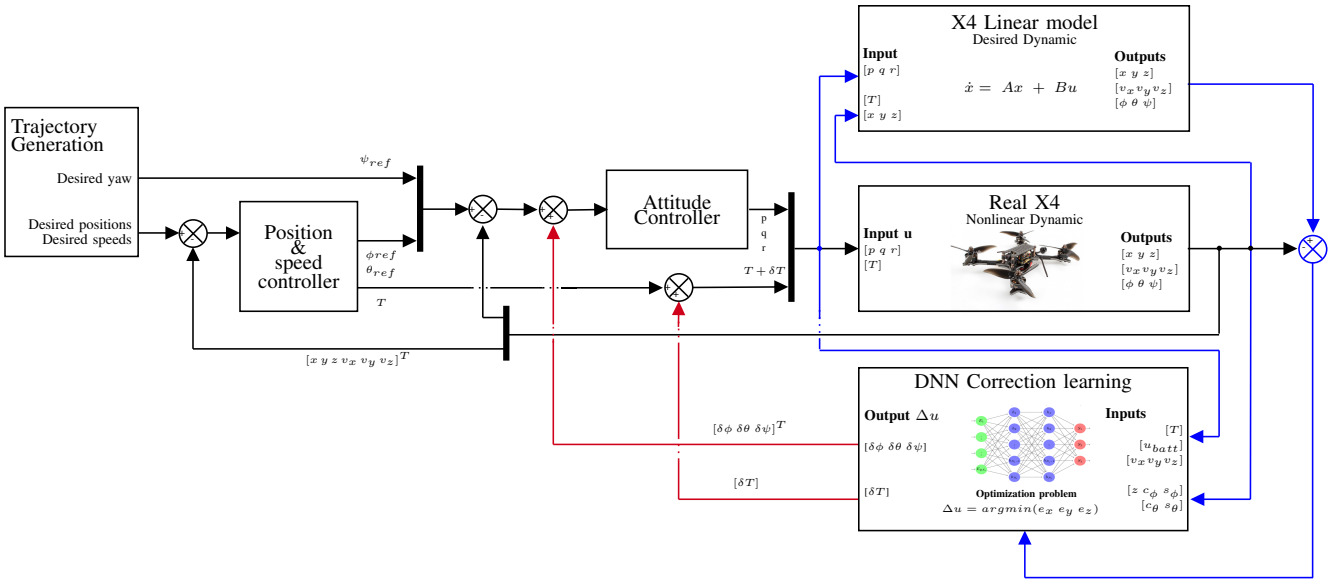

Fig. 4: Proposed Cascaded Controller Architecture for our quadrotor. Attitude and position & speed control loops are running at 100Hz. The DNN correction is learned offline and then is used to correct flight performance with respect to the expected linear dynamic.

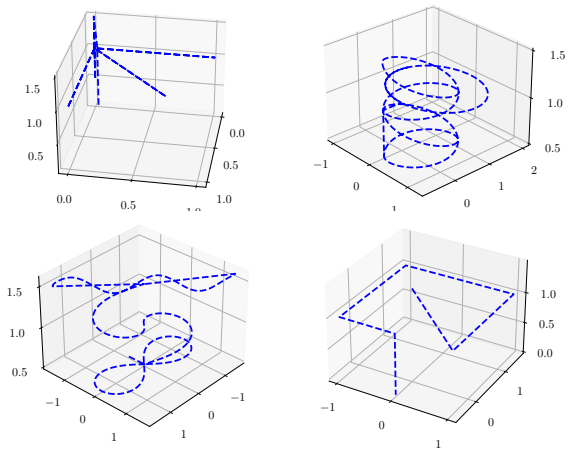

Fig. 5: Example of trajectories : step, circles, spirals, etc. included in the database. Distances are in meters.

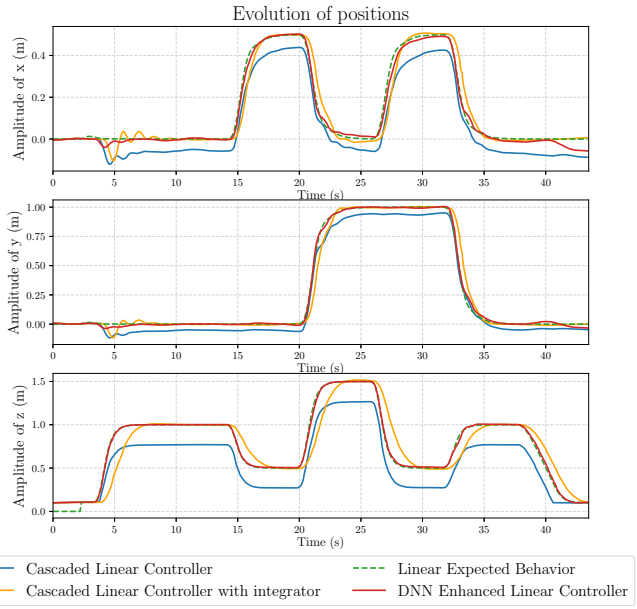

Fig. 6: Evolution of positions for each controller in ROS/Gazebo Simulation Environment test case scenario. The positions of the drone using the cascaded linear controller are represented in blue, with the integrator in yellow, and using the enhanced controller with DNN in red. Green dashed line represents the expected linear behavior on the linear model using cascaded PD linear controller only.

Those undesired errors, due to nonlinear dynamics of the quadrotor, aerodynamic effects, coupling and unmodelled actuator dynamics (open loop Electronic Speed Controller) are mostly corrected with the proposed corrected DNN Enhanced Linear Cascaded Controller. We can notice that the drone behavior (red line) is closer to the expected linear one: the DNN term has significantly decreased errors. Mean squared errors on this test trajectory are presented in tab. I for each axis. The integral controller tuned to be as close as possible to the linear quadratic controller also corrects the behavior but presents a slower time response.

C. Experimental Setup

In our experiments we use a Holybro Kopis 2 drone. The on-board flight controller is a Kakute F7 where PX4 firmware is installed. Internal PID controller is configured for angular rate loop control. Our drone is connected to a ground station via a wifi bridge that is sending our command signal at 100Hz. Space positions are obtained from the motion capture system, also running at 100Hz. The experimental setup is resumed in fig. 7.

D. Experimental Results

We present results of our experiments using section V-C setup. As for simulation, we first created a database using the

	Linear Cascaded Controller	Linear Cascaded Controller with integrator	DNN Enhanced Linear Controller
MSE x-axis	0.0049 m^2	0.0021 m^2	0.00029 m^2
MSE y-axis	0.0032 m^2	0.030 m^2	0.00022 m^2
MSE z-axis	0.0477 m^2	0.0259 m^2	0.00073 m^2

TABLE I: Mean Squared Error (MSE) on each axis for presented scenario (fig. 6) for the three controllers in ROS/Gazebo Simulation Environment.

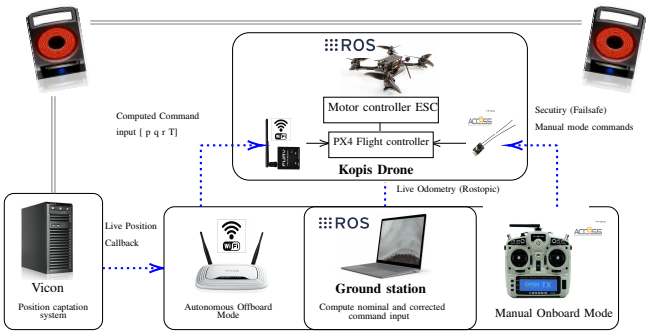

Fig. 7: Experimental setup.

linear cascaded controller on the Kopis drone. We collected data with several scenarios using different batteries, including new and old batteries to better cover their discharge profiles.

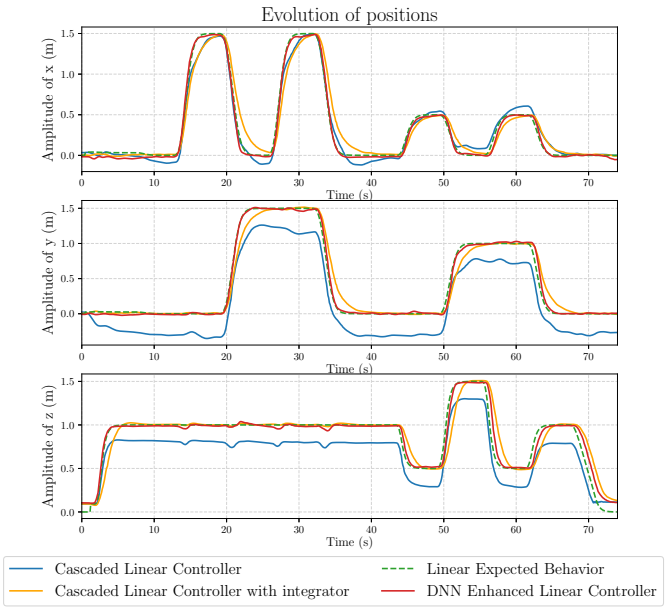

Fig. 8: Evolution of quadrotor positions for each controller, in real flight using our motion capture room (MOCA).

After training our DNN, we tested controllers on an unseen scenario, see fig. 8. Also, to see the impact of battery discharge profile on the controller law, we used a half-full battery for each controller test. The blue line in fig. 8 represents the controller without correction nor integrator: the behavior presents tracking errors. During this flight case, using the linear law, we can notice the huge impact of the battery discharge which is added to the tracking error. At starting, the tracking error is equal to 30% and is increased up to 35% at the end. As expected, the DNN Enhanced Controller completely removed tracking errors and increased flight speed compared to the linear controller. Here the DNN completely compensates the error over the entire test scenario and learned the battery discharge profile as we introduce to the DNN battery discharge tension as an input. Thus, the drone flight behavior is much closer to expected perfect linear dynamic (green dashed line). As for simulation results, the proposed controller is faster and closer to the linear expected behavior than the integral controller. Mean squared errors for experimental test trajectory are presented in tab. II. We invite readers to watch the video of the experiments attached to the paper.

	Linear Cascaded Controller	Linear Cascaded Controller with integrator	DNN Enhanced Linear Controller
MSE x-axis	0.0060 m^2	0.0150 m^2	0.0016 m^2
MSE y-axis	0.0815 m^2	0.0125 m^2	0.0017 m^2
MSE z-axis	0.0380 m^2	0.0198 m^2	0.0046 m^2

TABLE II: Mean Squared Error (MSE) on each axis for presented real flight scenario (fig. 8) for the three controllers.

Near Ground Test Comparison

To better see improvements of the DNN Enhanced Linear Cascaded Controller compared to the Cascaded Controller with an integrator, we tested both controllers in a near ground flight case (see fig. 9), here following a lemniscate at 15cm from the ground. In this type of scenario, ground effect strongly affects the drone general behavior. The same deep neural network, as in the previous part, is used as it was trained on many different scenarios including near ground flights. It is able to correct the behavior and get closer to the linear expected behavior in this highly perturbed situation. Here the integral controller is not rejecting ground effect disturbance, as this effect is not a constant perturbation. The neural term is able to catch those dynamical perturbations and the proposed DNN Enhanced controller, as shown in fig. 10, performs better than the integrator in all three axis.

Fig. 9: The quadrotor flying close to the ground. Trajectories are displayed in fig. 10

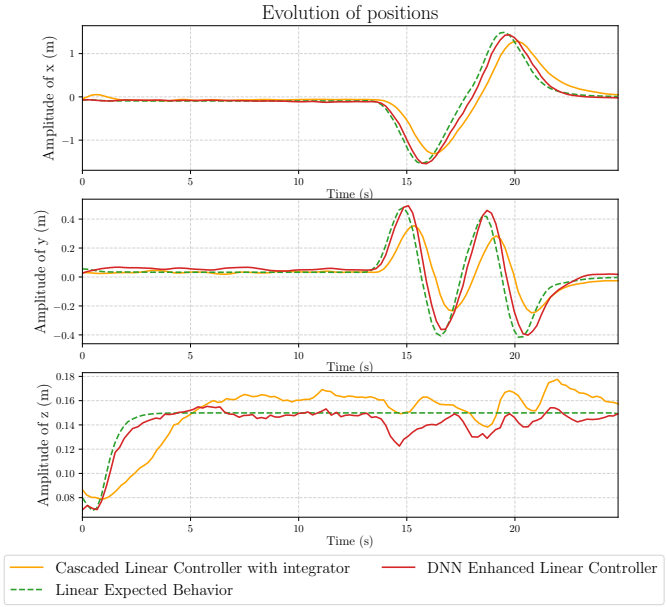

Fig. 10: Evolution of quadrotor positions for Cascaded Linear Controller vs DNN Enhanced Linear Controller. Experiment is carried close to the ground (15cm height). A photo of the quadrotor is provided in fig. 9

VI. Conclusion and Future Work

Rather than tuning a complex nonlinear algorithm, we propose the design of an easy to tune the linear cascaded controller. Then, a learning process is used allowing to correct this basic controller to reach performances initially fixed on a linear quadrotor model. Our experiment shows that with a small amount of flight data collected, the proposed DNN enhancement enables the drone behavior to be closer to a linear one. It allows to reduce undesired overshoots, slow time response and tracking errors. The advantage of this approach is that it is plug-and-play and can be applied to many UAVs present in the industry without the need to find a fine model of the drone. Future work aims at making experiments with online learning and correcting progressively during flight. It can also be extended to learn external perturbations, such as the wind, frame or blades issues, battery unpredicted behavior, etc. Finally, we plan to do a further ablation study to determine the optimal network structure for embedding it in the micro-controller.

References

[1] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*, vol. 114, pp. 649–666, Springer, 2016.

[2] D. Brescianini and R. D'Andrea, "Computationally efficient trajectory generation for fully actuated multirotor vehicles," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 555–571, 2018.

[3] M. Bangura and R. Mahony, "Nonlinear dynamic modeling for high performance control of a quadrotor," in *Proc. of the Australasian Conference on Robotics and Automation*, Dec. 2012.

[4] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2018.

[5] L. Martins, C. Cardeira, and P. Oliveira, "Linear quadratic regulator for trajectory tracking of a quadrotor," in *IFAC Symposium on Automatic Control in Aerospace*, vol. 52, pp. 176–181, 2019.

[6] K. Hunt, D. Sbarbaro, R. Żbikowski, and P. Gawthrop, "Neural networks for control systems - A survey," *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.

[7] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Combining convolutional neural networks and model predictive control," in *Machine Learning Research* (S. Levine, V. Vanhoucke, and K. Goldberg, eds.), vol. 78, pp. 133–142, PMLR, 13–15 Nov 2017.

[8] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2016.

[9] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Beauty and the beast: Optimal methods meet learning for drone racing," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2018.

[10] G. Torrente, E. Kaufmann, P. Foehn, and D. Scaramuzza, "Data-driven MPC for quadrotors," *IEEE Robotics and Automation Letters*, 2021.

[11] N. Mohajerin and S. L. Waslander, "Modelling a quadrotor vehicle using a modular deep recurrent neural network," in *IEEE Int. Conf. on Systems, Man, and Cybernetics*, 2015.

[12] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

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

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

[15] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," *IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2019.

[16] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[17] 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.

[18] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robotics & Automation Magazine*, vol. 19, no. 3, pp. 20–32, 2012.

[19] "PX4/PX4-autopilot software."

[20] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[21] T. Dozat, "Incorporating Nesterov Momentum into Adam," in *Int. Conf. on Learning Representations*, May 2016.