Synthesising Controllers for Quadrotors via Imitation Learning

Yan Zong, Ningyun Lu, Bin Jiang
College of Automation Engineering
Nanjing University of Aeronautics and Astronautics
Nanjing, China
{y.zong, luningyun, binjiang}@nuaa.edu.cn

***Abstract*—In this work, we present a quadrotor control solution using neural networks with imitation learning. Instead of mapping raw sensory variables to control outputs, we leverage a feature engineering method to design neural network input attributes, according to expert controller inputs. This improves the capability of the learning-based control policies on tracking different trajectories. Moreover, the neural network outputs of the collective thrust and torque can be directly fed to the quadrotor control allocator, without requiring any further data transformation. During the training process, we also use the Data Aggregation (DAgger) algorithm for the improved control performance. The simulation results show that the quadrotors, using the trained neural network controller, can achieve expert-like behaviour (i.e. tracking square trajectories).**

***Index Terms*—Quadrotor, imitation learning, data aggregation**

I. Introduction

Quadrotors have been widely applied in many domains (e.g. transportation and rescue). The Flight Control System (FCS) in a quadrotor mainly consists of three parts: guidance, navigation and control. The above three algorithms work together to guarantee the quadrotors track desired trajectories with minimum errors. However, designing a flight control system for quadrotors requires years of expertise and effort. Recently, we already have witnessed the success of artificial intelligence (e.g. machine learning and robotics) [1] in solving complex issues. Many machine learning techniques, including neural network and reinforcement learning, can learn the complicated mapping between the inputs (observations) and outputs (also known as actions or commands). We are interested in finding a single end-to-end control policy to let quadrotors track desired trajectories. Such a control policy can serve as a backup controller in safety-critical applications.

In this paper, we aim to use imitation learning with Data Aggregation (DAgger) to train Neural Networks (NNs) from demonstrations of experts. Based on the inputs and outputs of expert controllers, we design the attributes and output actions of the neural networks. Quadrotors, using the trained neural network control policies, can exhibit expert-like behaviour (i.e. tracking square trajectories in our work).

This work was supported in part by the National Natural Science Foundation of China grant 62303221, in part by the Nanjing University of Aeronautics and Astronautics grants YAH23008 and XCA22049-15, in part by the NUAA HPC.

A. Related Work

Until now, there has been much attention paid in exploiting machine learning algorithms for developing flight control systems. Most of researches use reinforcement learning to train neural networks, and these trained neural network can control quadrotors track desired trajectories (e.g. circles [2], combos [3]). The input attributes of neural networks are usually inertial measurements, namely, position, velocity, rotation matrix and angular velocity. In addition to the inertial variables, visual data and future trajectories are also used as input features [4].

In [2], [4], [5], the neural networks output four motors' thrusts to control quadrotors. The collective thrust and body rate are utilised as the commands [3]. To avoid the unstable issue, the authors in [6] leverage the summation of Proportional and Derivative (PD) control outputs and learning-based controller outputs as actions. Unfortunately, no consensus about selecting appropriate output form has been achieved so far. Instead of adopting raw inertial variables as input attributes, we utilise a feature engineering method (i.e. feature extraction) to help us design input variables. This can improve the capability of using neural network control quadrotors tack different trajectories. We also exploit the collective thrust and torque as control outputs, which can be directly fed to allocators for further processing (see Fig. 2).

Even though reinforcement learning demonstrates comparable performance on quadrotor trajectory tracking, its trial-and-error learning strategy may raise safety concerns during the training process. In the literature, there still exist several studies using imitation learning to control aerial vehicles. For example, in [7], a fully-connected neural network is trained via imitation learning to control fixed-wing unmanned aircraft. The current work shows that both imitation learning and reinforcement learning can attain similar performance in some cases [8]. Hence, we leverage imitation learning train neural networks, and then control quadrotors track desired trajectories. This is of importance in the safety-critical applications.

Moreover, training learning-based technologies for controlling quadrotors belongs to the sequential prediction problem,

where future observations depend on previous predictions and actions. This leads to violating the common (independent identically distributed) i.i.d. assumption in the classical supervised learning approaches. Due to the above issue, controlling quadrotors via learning-based solutions usually suffers from (training and testing) data mismatch and compounding errors [9]. The DAgger algorithm was proposed in [10] to solve this problem. We also utilise this method during neural network training for improving performance.

B. Contributions and Paper Organisation

We present a quadrotor control method using neural networks with imitation learning. Instead of mapping raw inertial variables to control outputs, we leverage a feature extraction solution to design neural network input attributes, according to expert controller inputs. This improves the capability of the learning-based control policies on tracking different trajectories. Moreover, the neural network outputs of collective thrust and torque can be directly fed to the quadrotor control allocator, without any further data transformation. During the training process, we also incorporate the DAgger algorithm for the improved control performance. The simulation results demonstrate that quadrotors, using the trained neural network controller, can track the desired square trajectories.

The rest of this paper is organised as follows: Section 2 presents the problem formulation and quadrotor model. Then, Section 3 shows the neural network input attributes, outputs, structure and training details. Next, Section 4 demonstrates simulation results. Eventually, Section 5 concludes this work.

II. Problem Formation

To synthesise controllers for quadrotors by using imitation learning, we implement the quadrotor dynamics in PyTorch. The following section presents a brief overview of the model implemented in the simulator.

A. A Quadrotor Model

We assume the quadrotor is a six degree-of-freedom rigid body of mass m, and the moment of inertia $\mathbf{J}$ is a constant diagonal matrix $\mathbf{J} = \text{diag}(\mathbf{J}_x, \mathbf{J}_y, \mathbf{J}_z)$. The geometric center is located at the Center of Gravity (CoG) of the quadrotor. The quadrotor also is only under gravity and thrust forces, which are generated by its four propellers [2]. Furthermore, the rotational speed Ω_i of each propeller is simplified as a first-order model with the time constant T_m [4], [11], $\mathbf{\Omega}_{\text{cmd}} = C_m\boldsymbol{\sigma} + \varpi_m$ is the motor speed command vector, where $\boldsymbol{\sigma} \in [\mathbf{0}, \mathbf{1}]$ is the throttle command vector, C_m and ϖ_m are the coefficients.

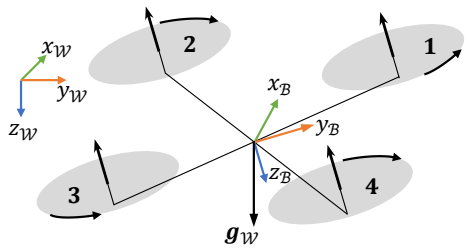

Fig. 1. Quadrotor with numbered propellers under the world and body frames.

The 16-dimensional model can be exploited to describe a quadrotor, and its dynamics is written as

$$\begin{aligned}
\dot{\boldsymbol{p}}_{\boldsymbol{\mathcal{W}}} &= \boldsymbol{v}_{\boldsymbol{\mathcal{W}}} \\
\dot{\boldsymbol{v}}_{\boldsymbol{\mathcal{W}}} &= \boldsymbol{g}_{\boldsymbol{\mathcal{W}}} - \frac{\boldsymbol{f}}{m}\mathbf{R} \\
\dot{\mathbf{\Theta}} &= \mathbf{W} \cdot \boldsymbol{\omega}_{\boldsymbol{\mathcal{B}}} \\
\dot{\boldsymbol{\omega}}_{\boldsymbol{\mathcal{B}}} &= \mathbf{J}^{-1} \cdot (-\boldsymbol{\omega}_{\boldsymbol{\mathcal{B}}} \times (\mathbf{J} \cdot \boldsymbol{\omega}_{\boldsymbol{\mathcal{B}}}) + \mathbf{G} + \boldsymbol{\tau}) \\
\dot{\mathbf{\Omega}} &= \frac{1}{T_m}\left(\mathbf{\Omega}_{\text{cmd}} - \mathbf{\Omega}\right)
\end{aligned} \tag{1}$$

where the position vector $\boldsymbol{p}_{\boldsymbol{\mathcal{W}}}$ and the linear velocity vector $\boldsymbol{v}_{\boldsymbol{\mathcal{W}}}$ are in the world $\mathcal{W}$ frame. $\boldsymbol{g}_{\boldsymbol{\mathcal{W}}} = \begin{bmatrix} 0 & 0 & 9.81\ m/s^2 \end{bmatrix}^T$ is the gravity vector in the world frame. The rotation matrix $\mathbf{R}$ is the rotation from $\mathcal{B}$ to $\mathcal{W}$ [11]. $\mathbf{\Theta} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$ is the rotation angle vector with the roll angle ϕ, pitch angle θ and yaw angle ψ. The matrix $\mathbf{W}$ equals to

$$\mathbf{W} = \begin{bmatrix} 1 & \tan\theta\sin\phi & \tan\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi\cos\theta \end{bmatrix}.$$

$\boldsymbol{\omega}_{\boldsymbol{\mathcal{B}}}$ is the angular velocity (i.e. body rate) vector in the body $\mathcal{B}$ frame. $\mathbf{G}$ is the *gyroscopic torque*. $\boldsymbol{f}$ and $\boldsymbol{\tau}$ are, respectively, the collective thrust and torque produced by four propellers in the body frame; these two variables are obtained from the following expressions:

$$\boldsymbol{f} = \sum_{i=1}^{4} \boldsymbol{f_i},\ \boldsymbol{\tau} = \sum_{i=1}^{4} \left(\boldsymbol{\tau_i} + r_i \times \boldsymbol{f_i}\right), \tag{2}$$

where r_i is the i-th propeller's location in $\mathcal{B}$. Typically, the thrust $\boldsymbol{f_i}$ and torque $\boldsymbol{\tau_i}$, which are generated by the i-th propeller, are assumed to be proportional to the square of the propeller's rotational speed Ω_i [4], [11], yielding

$$\boldsymbol{f_i} = \begin{bmatrix} 0 & 0 & c_T\Omega_i^2 \end{bmatrix}^T,\ \boldsymbol{\tau_i} = \begin{bmatrix} 0 & 0 & c_M\Omega_i^2 \end{bmatrix}^T, \tag{3}$$

the coefficients T_m, C_m, ϖ_m, c_T and c_M can be estimated via the parameter fitting method (see Section 6.3.4 of [11]).

B. Quadrotor Expert Control

In Fig. 2a, the low-level flight control $\boldsymbol{\pi}^*$ of a quadrotor consists of position control (i.e. outer-loop control), attitude control (inner-loop control), control allocation and motor control. Based on the desired position $\boldsymbol{p_d}$, the position control is to calculate the desired roll ϕ_d and pitch θ_d angles, along with the desired total thrust $\boldsymbol{f_d}$. Next, the attitude controller aims to

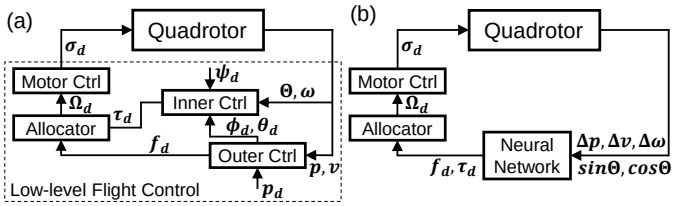

Fig. 2. Quadrotor FCS structure of (a) the expert control policy $\boldsymbol{\pi}^*$, and (b) a neural network controller.

obtain the desired torque $\boldsymbol{\tau_d}$, according to the desired attitude angles (which are from the outer-loop controller and input ψ_d value). The control allocator then can allocate the desired rotational speed $\Omega_{d,k}$ for each propeller. From the above desired rotational speeds $\boldsymbol{\Omega_d}$, the motor controller calculates the desired throttle command $\sigma_{d,k}$ of each motor, which is fed to the quadrotor.

For the position and attitude controllers in the low-level flight control system, we use several Proportional-Integral-Derivative (PID) controllers as controlling strategies. The altitude channel is an uncertainty-free model, a proportional controller is sufficient to deal with this type of system. However, the attitude channel is a dynamic model subject to various uncertainties, we have to exploit a PID controller to compensate for the effects of uncertainties [11]. Moreover, during the control parameter designing, this work ignores the effects of $\mathbf{G}$ and $-\boldsymbol{\omega_B} \times (\mathbf{J} \cdot \boldsymbol{\omega_B})$ in (1). We also adopt the small-angle assumption. To avoid such an assumption from being violated, and further from leading to the crash of a quadrotor, we consider a saturation function in the PID controller. By using the above PID control $\boldsymbol{\pi}^*$, the quadrotor can track a designated trajectory (see Fig. 5).

This work studies the trajectory tracking problem under the neural network control. Specifically, the PID control $\boldsymbol{\pi}^*$ is an expert controller, and we aim to develop a neural network to imitate this expert controller, and the trained NN also can let the quadrotor track the trajectory. Note that $\boldsymbol{\pi}^*$ is not necessarily the optimal control strategy. This is important because once in cases where the optimal control is found, a neural network can learn this expert controller through imitation learning.

III. Method

In this section, we first describe the input attributes, neural network control outputs, and the NN structure. Then, we present the method used to train our fully-connected neural network control for quadrotors.

A. Input and Output Variables

For the neural network input features, instead of directly using the raw (historical) quadrotor states or (future) reference information of the desired trajectory (e.g. position, velocity, rotation matrix and angular velocity), we exploit (i) the position error, velocity error, and angular velocity error between the reference values and the current quadrotor states at the current time instant t; (ii) the differences in position, velocity and angular velocity between the previous and current instants (i.e. the time $t-1$ and t); (iii) the trigonometric values of the three (reference and current) attitude angles at t.

As a result, the input attributes of a neural network controller are of the 42-dimension. The output $\boldsymbol{u}(t)$ from the neural network is a 4-dimensional vector, which consists of the collective thrust $\boldsymbol{f}$ and torque $\boldsymbol{\tau}$. We use a fully-connected neural networks with four hidden layers, and each layer, respectively, possesses 64-32-16-8 `Tanh` neurons. In this work, we have not tried different neural network structures. Usually, neural networks are versatile, and can handle a variety of problems with a single architecture [6].

B. Training Details

The neural network is trained over a Mean Squared Error (MSE) cost function via the Adam optimiser. However, using only the labelled inputs and output from the expert controller $\boldsymbol{\pi}^*$ may let the trained neural network suffer from compounding errors [10]. To avoid the above issue, we exploit the DAgger strategy. This process consists of rolling out the neural network control policy and labeling the visited states with the expert control [12]. Quadrotors follow the neural network control instead of $\boldsymbol{\pi}^*$ with the probability of $1-\beta$ when collecting new training data. The training details of using DAgger are given in Algorithm 1.

Algorithm 1 DAgger for Neural Network Control Training

initialise $\mathcal{D} \leftarrow \emptyset$, β;
for $i = 0 : N$ **do:**
  $\boldsymbol{\pi}_i = \beta^i \boldsymbol{\pi}^* + (1-\beta^i)\hat{\boldsymbol{\pi}}_i$;
  sample trajectories using $\boldsymbol{\pi}_i$;
  collect dataset: $\mathcal{D}_i = \{(s,\ \boldsymbol{\pi}^*(s))\}$[1];
  aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$;
  train $\hat{\boldsymbol{\pi}}_{i+1}$ on $\mathcal{D}$;
  obtain $\digamma$ via validating $\hat{\boldsymbol{\pi}}_{i+1}$;
end for
return $\hat{\boldsymbol{\pi}}_i$ with the smallest value of $\digamma$;

During NN validation, through comparing the neural network outputs $\boldsymbol{u}(t)$ and the expert outputs $\boldsymbol{u}^*(t)$, we define the accuracy $\digamma$ to measure the performance of a trained neural network:

$$\digamma = \frac{1}{4}\sum_{j=1}^{4}\left(\frac{1}{T}\sum_{i=1}^{T}\|\boldsymbol{u}(t) - \boldsymbol{u}^*(t)\|^2\right)^{\frac{1}{2}}, \tag{4}$$

note that a smaller accuracy $\digamma$ value means better control performance of the neural network.

IV. Simulation Results

We simulate a quadrotor with a mass of $1.4\ kg$, an arm length of $0.225\ m$ and a thrust-to-weight ratio of $2:1$. The simulation duration is 50 seconds, and the step size is $5\ ms$. The training procedure consists of 50 DAgger iterations with

[1] s represents the visited states by $\hat{\boldsymbol{\pi}}_i$, and $\boldsymbol{\pi}^*(s)$ denotes the actions given by the expert controller $\boldsymbol{\pi}^*$.

TABLE I
QUADROTOR CONFIGURATIONS

	Value	Unit
T_m	100	ms
C_m	706.01	$rad \cdot s^{-1}$
ϖ_m	170.47	$rad \cdot s^{-1}$
c_T	1.201×10^{-5}	$N \cdot (rad/s)^{-2}$
c_M	1.574×10^{-7}	$N \cdot m \cdot (rad/s)^{-2}$
$\mathbf{J}_x$	1.563×10^{-2}	$kg \cdot m^2$
$\mathbf{J}_y$	1.563×10^{-2}	$kg \cdot m^2$
$\mathbf{J}_z$	2.636×10^{-2}	$kg \cdot m^2$

30 epochs, and the batch size in each epoch is 20. The learning rate is set to 0.005. In DAgger, the probability β of choosing the expert control while training is decayed by a factor of 0.9 after each DAgger iteration. We only use the quadrotor states between 12.5 and 50 seconds for training and data aggregation. Table 1 summarises the quadrotor configurations.

Fig. 3 shows the evolution of the training cost and validation accuracy during the neural network training. The more DAgger processes are invoked, the better neural network model we can obtain. After 49 DAgger iterations and 14 epoches, we achieve the tiniest validation accuracy of $F = 0.177$. The above findings are also reflected in Fig. 4. We visualise the data aggregation processes. Even though there still exist several DAgger iterations (depicted in a light color) where the quadrotor runs away from the designated square trajectory, the quadrotor trajectories (presented in a darker color) using the neural network control are close to the desired trajectory as we have more data aggregation iterations.

Finally, we use both the neural network and expert controllers to let the quadrotor track the desired time-dependent path in the 50-second simulations. As can be seen from Fig. 5, both types of controllers achieve similar performance. This means that the trained neural network has successfully imitated the behaviour of the expert controller π^*.

V. CONCLUSION

In this work, we present a quadrotor control method using neural networks with imitation learning. Based on the expert controller inputs and outputs, we design neural network input features and output actions. This improves the capability of the learning-based control policies on tracking different trajectories, and also needs no further data transformation. During the training process, we also use the DAgger algorithm for the improved control performance. The simulation results demonstrate that the quadrotors, using the trained neural network controller, can track desired square trajectories.

REFERENCES

[1] European Aviation Artifical Intelligence High Level Group, "The Fly AI Report: Demystifying and Accelerating AI in Aviation/ATM," Mar. 2020.

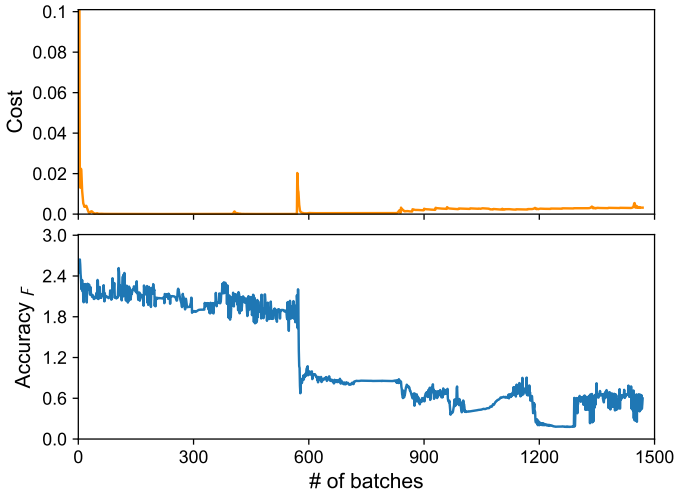

Fig. 3. Training cost and validation accuracy during DAgger.

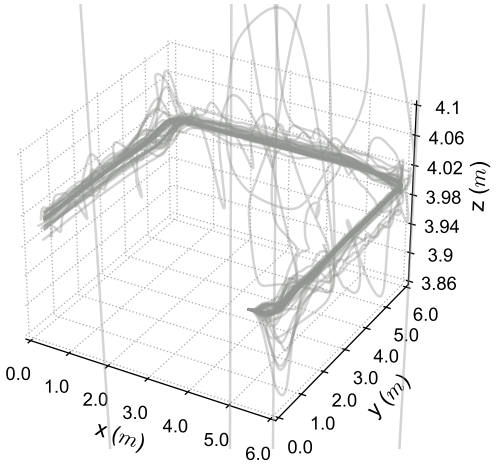

Fig. 4. Intermediate trajectories flown under neural network control within data aggregation.

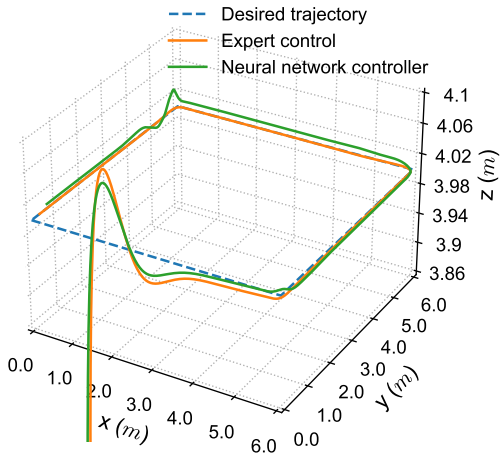

Fig. 5. Quadrotor trajectory tracking via both the expert and neural network control policies.

[2] C. Pi, K. Hu, S. Cheng, and I. Wu., "Low-Level Autonomous Control and Tracking of Quadrotor Using Reinforcement Learning," *Control Eng. Practice*, vol. 95, pp. 1-11, Feb. 2020.

[3] E. Kaufmann, A. Loquercio, R. Ranftl, M. Muller, V. Koltun, and D, Scaramuzza., "Deep Drone Acrobatics," in *Proc. Robot. Sci. Syst.*, Jul. 2020, pp. 1-10.

[4] E. Kaufmann, L. Bauersfeld, and D, Scaramuzza., "A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 10504-10510.

[5] A. Molchanov, T. Chen, W. Honig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme., "Sim-to-(Multi)-Real: Transfer of Low-Level Robust Control Policies to Multiple Quadrotors," in *Proc. Int. Conf. Intell. Robot. Syst. (IROS)*, Nov. 2019, pp. 59-66.

[6] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter., "Control of a Quadrotor with Reinforcement Learning," *IEEE Robot. Autom. Lett.*, vol. 2, no. 4, pp. 2096-2103, Oct. 2017.

[7] D. Shukla, S. Keshmiri, and N. Beckage., "Imitation Learning for Neural Network Autopilot in Fixed-Wing Unmanned Aerial Systems," in *Proc. Int. Conf. Unmanned Aircraft Syst. (ICUAS)*, Sep. 2020, pp. 1508-1517.

[8] A. Kumar, J. Hong, A. Singh, and S. Levine., "When Should We Prefer Offline Reinforcement Learning Over Behavioral Cloning?" in *Proc. Int. Conf. Learn. Representations (ICLR)*, Apr. 2022, pp. 1-36.

[9] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer., "HG-DAgger: Interactive Imitation Learning with Human Experts," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 8077-8083.

[10] S. Ross, G. J. Gordon, and J. A. Bagnell., "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," in *Proc. Int. Conf. Artif. Intell. Statistics (AISTATS)*, Apr. 2011, pp. 627-635.

[11] Q. Quan., Introduction to Multicopter Design and Control, 1st ed. Springer, 2017.

[12] A. Loquercio, E. Kaufmann, R. Ranftl, M. Muller, V. Koltun, D. Scaramuzza., "Learning High-Speed Flight in the Wild," *Sci. Robot.*, vol. 6, np. 59, pp. 1-16, Oct. 2021.