Deep Neural Networks for Improved, Impromptu Trajectory Tracking of Quadrotors

Qiyang Li, Jingxing Qian, Zining Zhu, Xuchan Bao, Mohamed K. Helwa, and Angela P. Schoellig

***Abstract*— Trajectory tracking control for quadrotors is important for applications ranging from surveying and inspection, to film making. However, designing and tuning classical controllers, such as proportional-integral-derivative (PID) controllers, to achieve high tracking precision can be time-consuming and difficult, due to hidden dynamics and other non-idealities. The Deep Neural Network (DNN), with its superior capability of approximating abstract, nonlinear functions, proposes a novel approach for enhancing trajectory tracking control. This paper presents a DNN-based algorithm as an add-on module that improves the tracking performance of a classical feedback controller. Given a desired trajectory, the DNNs provide a tailored reference input to the controller based on their gained experience. The input aims to achieve a unity map between the desired and the output trajectory. The motivation for this work is an interactive "fly-as-you-draw" application, in which a user draws a trajectory on a mobile device, and a quadrotor instantly flies that trajectory with the DNN-enhanced control system. Experimental results demonstrate that the proposed approach improves the tracking precision for user-drawn trajectories after the DNNs are trained on selected periodic trajectories, suggesting the method's potential in real-world applications. Tracking errors are reduced by around 40-50% for both training and testing trajectories from users, highlighting the DNNs' capability of generalizing knowledge.**

I. Introduction

In recent years, quadrotors have been widely used for civilian and law-enforcement purposes, such as providing aerial surveillance, carrying out rescue missions, transporting goods over distance, and performing surveying and inspection tasks [1]–[4]. In all these applications, the quadrotor is required to precisely track a desired trajectory in order to perform the task safely and effectively.

Trajectory tracking for quadrotors poses a challenge on controller design. First, quadrotors are underactuated systems with nonlinear dynamics, making it a difficult control problem. Second, trajectory tracking precision of quadrotors can be affected by many factors, including uncertainty in the turn-rate-to-thrust map, time delays that are difficult to quantify, aerodynamic effects and other unpredictable factors such as friction in the actuators. Third, even in a perfect world, where the system dynamics are known exactly, a given classical controller cannot achieve perfect tracking for any arbitrary, feasible, desired trajectory.

Our goal is to achieve improved trajectory tracking control for quadrotors while taking into account three features that are crucial for most real-world trajectory tracking applications (Figure 1 shows our specific application):

1) Stability of the control system and robustness to reasonable disturbances must be guaranteed to ensure safety of the operation.
2) The system should be able to precisely track a new trajectory without adaptation.
3) The computational resources needed for the control system should be manageable such that the algorithm can be applied to small vehicles with limited computational power.

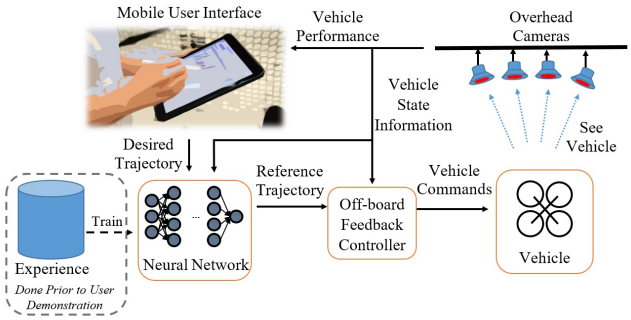

Fig. 1. Block diagram of our interactive "fly-as-you-draw" demo. The user draws an arbitrary trajectory on a mobile device. Upon receiving the new trajectory, the quadrotor immediately takes off and follows the signal, processed by the pre-trained Deep Neural Network (DNN) in the $(x\text{-}z)$-plane. The overhead camera system provides state feedback and also performance feedback to the user. A demo video can be found at `http://tiny.cc/DNN-ImpromptuTracking`.

Simple controllers such as typical proportional-integral-derivative (PID) controllers can achieve adequate performance under certain conditions, for example low speeds and accelerations, while having all the crucial features mentioned above [5], [6]. However, PID controllers are difficult to tune and they tend to behave poorly on more aggressive trajectories. There exist previous works on improving control for quadrotors or other robots, such as learning the dynamics or the inverse dynamics, iterative learning control and Gaussian Process learning. However, we show in Section II that these approaches have drawbacks with respect to the three crucial features we identified above, which are relevant for real-time trajectory tracking.

In this paper, we propose a DNN-based control system which improves the trajectory tracking performance by utilizing past flight experiences. After offline training from relevant flight examples, a generalized model is obtained with the DNN. This model can be evaluated in real-time

The authors are with the Dynamic Systems Lab (www.dynsyslab.org) at the University of Toronto Institute for Aerospace Studies (UTIAS), Canada. Email: {qiyang.li, jingxing.qian, zining.zhu, xuchan.bao}@mail.utoronto.ca, mohamed.helwa@robotics.utias.utoronto.ca, schoellig@utias.utoronto.ca.

to modify the reference signal given to the controller. With no prior knowledge of the system other than the training data, the proposed method demonstrates its ability to reduce trajectory tracking error by compensating for controller imperfections and unknown dynamics. Also, the DNN model is computationally efficient for real-time evaluation and effective even on arbitrary trajectories, not trained on before, making it applicable to impromptu tracking tasks.

To validate the effectiveness of the proposed method and motivate this work, we implement an interactive fly-as-you-draw application, where the quadrotor takes off to follow an arbitrary, hand-drawn trajectory immediately after the user finishes drawing the trajectory. The application uses neural networks, pre-trained with quadrotor flight data collected from periodic training trajectories, to obtain reference signals in real-time for an off-board feedback controller. This process is described in Figure 1. With this interactive application, we evaluate different DNN features, and compare the DNN performance with the baseline nonlinear controller performance. Nine of the 30 user-drawn testing trajectories used in our experiments are shown in Figure 2. Through the experiments, we demonstrate that the proposed approach, with proper feature selection for the DNN learning, is able to consistently enhance trajectory tracking precision for complex, arbitrary hand-drawn trajectories. Moreover, because the DNN serves as a pre-block outside the feedback control loop, the proposed method can be generalized as an add-on to any black-box, stable feedback control system. These characteristics of the proposed approach demonstrate its potential in real-world applications that require highly precise maneuvering, such as monitoring and inspection tasks, aerobatics, skywriting and airborne filming.

The paper is organized as follows. Section II summarizes related previous work on advanced trajectory tracking control. In Section III, we state the problem, followed by the general methodology in Section IV. The experimental setup is presented in Section V, and the corresponding results are presented in Section VI. A brief summary and discussion of the results are presented in Section VII.

II. Related Work

Neural networks (NNs) are a generic approach for approximating functions given a large amount of data. In previous work, NNs have been adopted to introduce modifications to feedback control loops.The papers [7] and [8] use NNs to learn the dynamics of a helicopter and a quadrotor, respectively. In [9], an NN is used for direct inverse control of a quadrotor with promising initial simulated results for hover flight. However, by involving NNs to modify the original feedback control loop, the stability of the control system will likely be affected. When previously unseen inputs are given to the NN that is part of the feedback control loop, the NN might generate unpredictable outputs, leading to instability of the system. Instead, we use DNNs (multi-layer NNs) to learn a model that directly determines the reference inputs to the feedback control loop. The proposed DNNs act as a pre-block outside the original feedback control loop, and run at a lower update rate, which makes the system much less susceptible to instability.

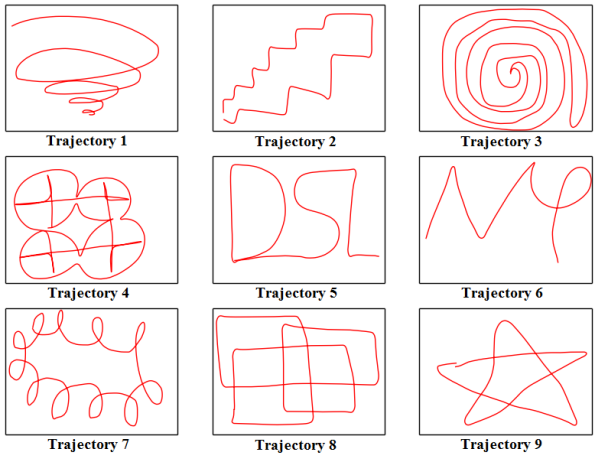

Fig. 2. Nine of the 30 user-drawn trajectories used in testing the performance of the DNN control system. For testing on the quadrotors, the drawn trajectories were modified to ensure the trajectory is bounded by maximum velocity (0.6 m/s) and acceleration (2.0 m/s^2).

Iterative learning control (ILC) is an approach of improving the control precision by repeating the same task and learning from previous executions [10]. Through the repetition of one specific task, ILC learns an updated reference input and achieves high-precision tracking for this particular task. Unlike simple controllers that can fail to achieve aggressive maneuvers, previous work has demonstrated ILC's ability of achieving high control precision on these tasks [11]. One significant drawback of this approach is that the experience of learning one specific task is not transferable to other tasks. Although [12] has shown that linear maps can optimize ILC initialization from previous experience, ILC still has to re-learn through multiple iterations before achieving high precision for a new trajectory. Our approach allows training ahead of time, and the trained model generalizes to arbitrary trajectories without any adaption process. This feature makes it suitable for applications that require the vehicle to complete the desired task with high precision in a timely manner.

Gaussian Process (GP) learning is receiving growing attention in the control community and has been used in various control problems. For instance, an accurate kinematic control of a cable-driven surgical robot is implemented in [13]. Similar to the idea of learning the reference input in our proposed method, the GP learns the reference input to the controller of the surgical robot, improving the tracking precision of the end-effector. Our approach is different from this GP learning approach in two ways: 1) we apply the method on the quadrotor system which has different dynamics compared to surgical robots; 2) we employ DNNs as our learning technique instead of GPs. One advantage of using DNNs is that DNNs can summarize data using a fixed-size model. A GP model gets bigger as more data is collected, making the model large in terms of required storage and

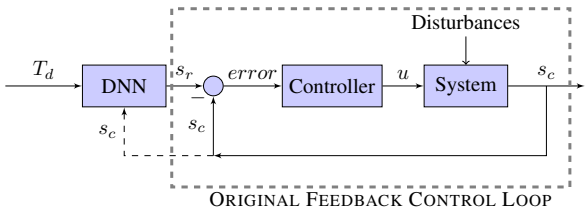

Fig. 3. The modified control system with the DNN block in front of the original baseline controller; the controller takes in the reference state (produced by the DNN) and the current state to control the system.

computationally expensive to evaluate. In contrast, when the size of the data set increases, DNNs adjust their parameters to better fit the training data without increasing the model size. Since modeling complex relations usually requires a large set of training data, the invariant model size makes the DNNs more promising on control systems with complex dynamics, especially when computation is limited.

III. Problem Statement

For a given dynamic system with a baseline feedback controller (see Figure 3), the problem is to learn a mapping from the desired trajectory T_d and the current state $\mathbf{s}_c$ to the reference input $\mathbf{s}_r$ of the baseline controller, in order to enhance the tracking performance of the overall system for arbitrary desired trajectories. We define the desired trajectory $T_d = \{\mathbf{s}_{d,1}, \mathbf{s}_{d,2}, \ldots, \mathbf{s}_{d,N}\}$ as a sampled trajectory containing N consecutive time steps, where $\mathbf{s}_{d,t}$ represents the desired state at the t^{th} time step. Learning is done off-line, and the learned mapping is applied in real-time.

IV. Methodology

A. Supervised Learning with Deep Neural Network Model

Our approach builds upon supervised learning with DNN. This learning process requires the preparation of a large number of labeled training examples and the training of DNN on these examples. Each labeled training example consists of an input and expected output pair to describe what the function should output according to a specific input. The training of DNN involves back-propagation to minimize the loss over all training examples [14], defined as the Euclidean distance between the network's output and expected output. After learning from the labeled training examples, the DNN can summarize a mapping from the training inputs to the training outputs.

A feed-forward DNN with rectified linear units (ReLU) is used to learn the mapping formulated in Section III. To prepare the training examples for learning the target mapping from the flying data, we use the actual trajectory as training inputs and the reference signal as the labeled output. The idea behind this selection is that if the actual trajectory was the desired trajectory, then the DNN should provide this saved reference signal to achieve perfect tracking. The specification of input and outputs for the DNN will be discussed in detail in Section V-D.

B. DNN as Reference Generator

The proposed method modifies the control system design by adding a DNN block in front of the controller. At each time step t, the trained DNN modifies the control signal in real time by giving the reference input $\mathbf{s}_{r,t}$ to the controller based on the desired trajectory T_d as well as the current state of the quadrotor $\mathbf{s}_{c,t}$.

Figure 3 highlights the difference between the original control system and the proposed one. The reference states generated by the DNN over N consecutive time steps form the reference trajectory $T_r = \{\mathbf{s}_{r,1}, \mathbf{s}_{r,2}, \ldots, \mathbf{s}_{r,N}\}$, where $\mathbf{s}_{r,t}$ is the reference state generated by the DNN at the t^{th} time step. Also, the actual trajectory $T_c = \{\mathbf{s}_{c,1}, \mathbf{s}_{c,2}, \ldots, \mathbf{s}_{c,N}\}$ completed by the vehicle over the N consecutive time steps is observed. The actual trajectory T_c is expected to closely match the desired trajectory T_d.

In control systems, current state feedback enables the control system to reject external disturbances if the controller is designed properly. Similarly, the extra loop introduced in our proposed system enables the DNN to adjust its output reference according to the current state to compensate the disturbances. We choose the feedback rate for this extra loop to be much lower than the original control loop to ensure that the stability of the original control system is not disrupted by the DNN signals. For example, in our experiments on the quadrotor control system, we design our DNN to send reference states at 7 Hz, which is 10 times slower than the control loop operating at 70 Hz. Therefore, DNN control signals can be non-intrusive to the original control system.

C. Feature Selection for the DNN

Ideally, at each time step t, the DNN would receive all the given information (both the entire desired trajectory T_d and the current state $\mathbf{s}_{c,t}$), and produce an optimal reference state $\mathbf{s}_{r,t}$ as the input to the controller to minimize the quadrotor's tracking error. However, this makes input dimension huge, and requires exponentially increasing amount of training data. Therefore, selecting proper state information for DNN is crucial for making the DNN learning effective.

A minimum feature selection is to only use the current desired state $\mathbf{s}_{d,t}$ and the current state $\mathbf{s}_{c,t}$ as the input, but this configuration may not be able to provide information for the DNN to model the hidden dynamics including time delay which may deteriorate the tracking performance. Hence, we consider including future desired states into the DNN input. With the additional future information, the properly-trained DNN is expected to plan the control ahead of time by considering future desired states and improve control performance. In this paper, we investigate the influence of future states on DNN learning. It is hypothesized that the DNN can give a better performance when the desired states in the near future are introduced into the DNN input.

To validate this hypothesis, we conducted experiments on a quadrotor in real-world environment to investigate 1) the effect of selecting different state information, including future desired states and current state feedback, as the input of the DNN on the DNN performance, and 2) the

generalizability of this method for improving the tracking performance for different trajectories overall.

V. Experiment Setup

A. The Quadrotor Model and Experiment Platform

This subsection provides a glimpse of quadrotor dynamics as well as the experiment platform. For more details about the quadrotor dynamics and control, readers are referred to [5].

A typical quadrotor consists of a symmetrical cross-shaped frame with four propellers mounted at the end of four arms. The full state of the quadrotor consists of 12 components. The translational position of the quadrotor's center of mass is defined as $\mathbf{p} = (x, y, z)$ and the attitude, represented by Euler angles roll, pitch and yaw, is defined as (ϕ, θ, ψ). In addition to translational position and attitude, the full state of the quadrotor includes the translational velocity, $\mathbf{v} = (\dot{x}, \dot{y}, \dot{z})$, and the rotational velocity, $\boldsymbol{\omega} = (p, q, r)$.

The experiments are conducted on a Parrot AR.Drone 2.0 quadrotor. This commercial quadrotor suits the needs of this study as it features highly nonlinear dynamics, complex aerodynamics that are hard to model, and most importantly, an unmodified black-box, which is an on-board controller that controls the vehicle's roll, pitch and yaw by adjusting motor forces. The quadrotor's states are all measured by the overhead *Vicon* motion capture system. The system features eight 4-mega pixel Vicon cameras running at 200 Hz. A similar experimental setup is described in detail in [6]. The baseline control system used in this paper consists of two controllers: the on-board controller and the off-board controller. The off-board controller is a nonlinear controller, composed of a nonlinear transformation and standard PD controller. It is implemented using the open-source Robot Operating System (*ROS*). The controller runs at 70 Hz, receives the quadrotor's current state and the reference, and outputs to the on-board controller the desired roll, pitch, yaw velocity and z velocity (ϕ_{cmd}, θ_{cmd}, r_{cmd}, $\dot{z}_{cmd}$). The on-board controller runs at 200 Hz, receives the four commands from the off-board controller, and adjusts the four motor thrusts $F_1, \ldots, F_4$ accordingly. The DNN feedback loop runs at 7 Hz, which is 10 times slower than the off-board controller.

B. Task Performance

Each task performed by the quadrotor involves following one of the pre-defined, desired trajectories T_d in the (x-z)-plane, where these trajectories are hand drawn through our interactive application (Figure 1). The error function for each task is defined as the root-mean-square (RMS) error of N pairs of (x, y, z)-coordinates sampled at 7 Hz, the DNN feedback loop sampling rate, between the desired trajectory, T_d, and the observed trajectory, T_c:

$$E(T_c, T_d) = \sqrt{\frac{1}{N}\sum_{t=1}^{N} \|\mathbf{p}_{c,t} - \mathbf{p}_{d,t}\|^2}, \qquad (1)$$

where $\|\mathbf{p}_{c,t} - \mathbf{p}_{d,t}\|$ is the Euclidean norm, while $\mathbf{p}_{d,t}$ and $\mathbf{p}_{c,t}$ are the position coordinates sampled at the t^{th} time step from the desired trajectory T_d and the observed trajectory T_c, respectively. The quadrotor in the experiment repeats each task with and without the aid of the trained DNN. The percentage reduction in errors between corresponding flights is identified as the improvement of our method on this specific task:

$$I(T_{\text{w/ DNN}}, T_{\text{w/o DNN}}) = \left(1 - \frac{E_{\text{w/ DNN}}}{E_{\text{w/o DNN}}}\right) \times 100\%, \qquad (2)$$

where $E_{\text{w/ DNN}}$ and $E_{\text{w/o DNN}}$ are the RMS errors in (1) with and without the DNN, respectively.

C. DNN Input-Output Specification

In general, the trained DNN provide a mapping from the current and selected desired states to the reference state:

$$\{\mathbf{s}_{c,t}, \mathbf{s}_{d,t_1}, \ldots, \mathbf{s}_{d,t_L}\} \rightarrow \mathbf{s}_{r,t}, \qquad (3)$$

where $\mathbf{s}_{c,t}$ is the vehicle's current state at the t^{th} time step, and $\{\mathbf{s}_{d,t_1}, \ldots, \mathbf{s}_{d,t_L}\}$ are L selected desired states from the desired trajectory T_d. Each of the states ($\mathbf{s}_c$, $\mathbf{s}_d$ and $\mathbf{s}_r$) mentioned above contains the full state of the quadrotor along with the translational acceleration on z-direction, $\ddot{z}$. Among all translational accelerations, $\ddot{x}$, $\ddot{y}$ and $\ddot{z}$, only $\ddot{z}$ is included in these states because the controller we used in the experiment only requires $\ddot{z}$ along with the full state of the quadrotor, $\{x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r\}$, as its inputs. In summary, the state of the vehicle is defined as $\mathbf{s} = \{x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r, \ddot{z}\}$.

Based on this general input-output mapping provided by the DNN, we explore three different configurations of the DNN to investigate the influence of including the future desired states and/or the current state feedback as inputs to the DNN (as discussed in Section IV-C):

- DNN with future desired states and the current state feedback;
- DNN with future desired states and without the current state feedback;
- DNN without future desired states and with the current state feedback.

All configurations consider one or more desired states at different time steps over the entire flight path as part of the input. Since we hypothesize that using future desired states can enhance tracking performance, we focus on the scenario where we only select the desired states $\{\mathbf{s}_{d,t_1}, \ldots, \mathbf{s}_{d,t_L}\}$ from the current desired state and the future desired states, i.e., we select $t_i = t + \Delta_i$, where $\Delta_i \geq 0$, for $i = 1, \ldots, L$.

For the first configuration with the current state feedback, the current observed state $\mathbf{s}_{c,t}$ and the L selected desired states, $\{\mathbf{s}_{d,t+\Delta_1}, \ldots, \mathbf{s}_{d,t+\Delta_L}\}$, are given to the DNN at each time step. The actual input to the DNN consists of two major parts. The first part includes $L + 1$ sets of $\{\dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r, \ddot{z}\}$ from the current observed and the selected desired states. The second part includes L sets of desired positions relative to the current observed position $\{\mathbf{p}_{d,t+\Delta_1} - \mathbf{p}_{c,t}, \ldots, \mathbf{p}_{d,t+\Delta_L} - \mathbf{p}_{c,t}\}$, where $\{\mathbf{p}_{d,t+\Delta_i}\}$ and $\mathbf{p}_{c,t}$ are the position components from the selected desired

states and the current observed state, respectively[1]. The input to the DNN for the second configuration is similar to the first configuration with only $\mathbf{s}_{c,t}$ being replaced by $\mathbf{s}_{d,t-1}$, and consequently $\mathbf{p}_{c,t}$ being replaced by $\mathbf{p}_{d,t-1}$, where $\mathbf{p}_{d,t-1}$ is the position components from $\mathbf{s}_{d,t-1}$. The last configuration is a special case of the first configuration, in which the current desired state $\mathbf{s}_{d,t}$ is the only selected desired state in the DNN input ($L = 1$ and $\Delta_1 = 0$). For this configuration, the DNN does not consider the future states, and generates the reference state only based on the current observed state and the current desired state.

The output of the DNN in the three configurations is the reference state $\mathbf{s}_{r,t}$. In our experiment, we reduce data complexity by only learning the difference between the reference state $\mathbf{s}_{r,t}$ and the current desired state $\mathbf{s}_{d,t}$ in translational position and velocity components.

D. Data Collection for DNN Training

To train the DNN, we need to collect the state data from real-world flights and select the training data. To that end, we design a 400-second trajectory that oscillates sinusoidally in all the x, y and z-directions with different combinations of amplitudes to cover the feasible state space as much as possible. In particular, each of the three directions has its own oscillating frequency (0.27 Hz, 0.20 Hz and 0.13 Hz for the x, y and z-directions respectively). We also gradually increase the amplitudes from 0 to 2 m in all directions. On this trajectory, the quadrotor can reach a maximum velocity of 1.5 m/s and a maximum acceleration of 4 m/s^2. The maxima for rotational velocity $\boldsymbol{\omega}$ and rotational acceleration $\dot{\boldsymbol{\omega}}$ are 0.4 rad/s and 1 rad/s^2 respectively.

Using the baseline control system to follow the designed training trajectory, we collect a $\{\mathbf{s}^*_{c,t}, \mathbf{s}^*_{d,t}\}$ pair at each time step, where $\mathbf{s}^*_{c,t}$ is the current observed state and $\mathbf{s}^*_{d,t}$ is the current desired state[2]. Approximately 10,000 raw data pairs are collected at a 7 Hz rate from four flights on the training trajectory, and then organized in the consecutive time order.

We select and re-organize these raw data pairs to establish the labeled training set. Recall that the DNNs aim to learn a mapping from the current state $\mathbf{s}_{c,t}$ and L selected desired states $\{\mathbf{s}_{d,t+\Delta_1}, \ldots, \mathbf{s}_{d,t+\Delta_L}\}$ to the reference state $\mathbf{s}_{r,t}$ that should be given to the controller at the current time step. For any pair $\{\mathbf{s}^*_{c,t}, \mathbf{s}^*_{d,t}\}$ in the data set, we consider $\{\mathbf{s}^*_{c,t+\Delta_1+1}, \ldots, \mathbf{s}^*_{c,t+\Delta_L+1}\}$. If we treat $\mathbf{s}^*_{c,t}$ as the current observed state $\mathbf{s}_{c,t}$ of the quadrotor and $\{\mathbf{s}^*_{c,t+\Delta_1+1}, \ldots, \mathbf{s}^*_{c,t+\Delta_L+1}\}$ as the selected desired states $\{\mathbf{s}_{d,t+\Delta_1}, \ldots, \mathbf{s}_{d,t+\Delta_L}\}$, then $s^*_{d,t}$ may be selected as the reference state $s_{r,t}$, given that $s^*_{d,t}$ is a point in a feasible reference sequence achieving perfect tracking with one sample delay. The mapping $\{\mathbf{s}^*_{c,t}, \mathbf{s}^*_{c,t+\Delta_1+1}, \ldots, \mathbf{s}^*_{c,t+\Delta_L+1}\} \rightarrow \mathbf{s}^*_{d,t}$, therefore, is an approximate mapping of $\{\mathbf{s}_{c,t}, \mathbf{s}_{d,t+\Delta_1}, \ldots, \mathbf{s}_{d,t+\Delta_L}\} \rightarrow \mathbf{s}_{r,t}$. Thus, for each pair $\{\mathbf{s}^*_{c,t}, \mathbf{s}^*_{d,t}\}$ in the data set, we can form a training pair (a, b) for learning the approximate mapping, where $a = \{\mathbf{s}^*_{c,t}, \mathbf{s}^*_{c,t+\Delta_1+1}, \ldots, \mathbf{s}^*_{c,t+\Delta_L+1}\}$ is the input and $b = \mathbf{s}^*_{d,t}$ is the labeled output. A labeled data set can then be obtained by collecting the training pairs (a, b). As a result of training, using the supervised learning technique for our DNNs with the labeled data (as discussed in Section IV-A), the DNNs are expected to learn an approximate mapping from the inputs to the output.

E. DNN Training

We train six different DNNs that share the same input states to find the mapping in (3), one for each of the position and velocity elements of the reference state $\mathbf{s}_r$, $\{x, y, z, \dot{x}, \dot{y}, \dot{z}\}$. Other elements in $\mathbf{s}_r$, $\{\phi, \theta, \psi, p, q, r, \ddot{z}\}$, maintain intact. We construct the DNNs in such a way because in preliminary experiments we observed that training of the six outputs might require different number of iterations to converge. For example, velocity and position in the z-axis are easier to train whereas positions in x-y plane are much harder to train. We also observe that the six DNNs demonstrate comparable performance after 2,000 training iterations. However, it is possible to merge them to jointly learn the six outputs. We construct the DNNs using *TensorFlow*, an open-source library originally developed by Google. Each DNN consists of four fully connected hidden layers, and each layer contains 128 neurons. 90% of the collected raw data pairs are used for training, while the rest are used for validation. Adam optimizer is used to tune the weight and bias parameters in the DNNs, and the learning rate is set at 0.0003 [15]. To prevent over-fitting, a dropout rate of 0.5 is used [16]. For each training iteration, 30 training pairs are used. 2,000 iterations are done for training each output[3].

VI. Experimental Results

A. Impact of Future States on DNN Tracking Performance

The influence of introducing future desired states as part of the input to the DNNs is investigated. It is expected that the DNNs can use the future desired states information to better "plan" the flying path for the future desired states while compensating for the effect of hidden dynamics, including time-delay and other factors. In Figure 4, we show that the DNNs with future desired states and current state feedback performs significantly better than the baseline control and the DNNs with current state feedback but without future desired states. Also, Figure 5 highlights that the reference trajectories produced by the DNNs trained with future desired states and the current state feedback effectively correct the tracking error in both the x- and z-directions.

B. Impact of Feedback on DNN Tracking Performance

We also investigate the influence of removing the current state feedback from the DNN inputs. Our experiments are conducted on the DNNs trained with future desired states. To remove the feedback loop, we keep the DNNs while replacing the current state feedback with the desired state from the previous time step during actual flight, as discussed

[1]Relative position information, instead of absolute position information, is used in order to reduce input data dimension.

[2]The superscript * indicates a state from the training data.

[3]The 30 training pairs are randomly selected from the 90% of the raw data pairs used for training.

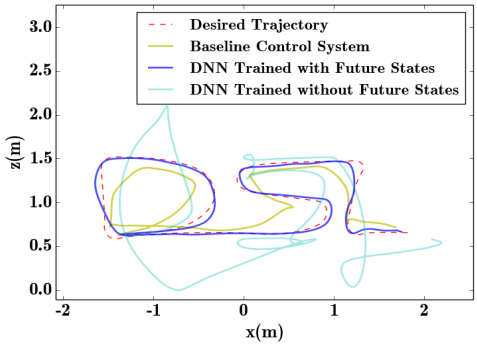

Fig. 4. Observed trajectories resulting from DNNs with two future desired states ($\Delta_1 = 4$, +0.6 s, and $\Delta_2 = 6$, +0.9 s, blue), DNNs without future desired states (cyan), and the baseline control system (yellow). The desired trajectory is Trajectory 5 in Figure 2. It consists of three hand-written letters "DSL" (red dashed). The RMS tracking errors, E, are 0.1926 m, 0.758 m and 0.434 m respectively. Overall, the proposed method with given future states achieved 56% improvement over the baseline controller.

TABLE I

TRACKING ERRORS ON ONE TRAJECTORY FOR THREE DIFFERENT CONTROL SYSTEMS

	Baseline Controller	DNN without Feedback	DNN with Feedback
RMS Error, E (m)	0.360	0.232	0.144
Peak Error (m)	0.605	0.497	0.356
Improvement, I (%)	-	35.6	59.9

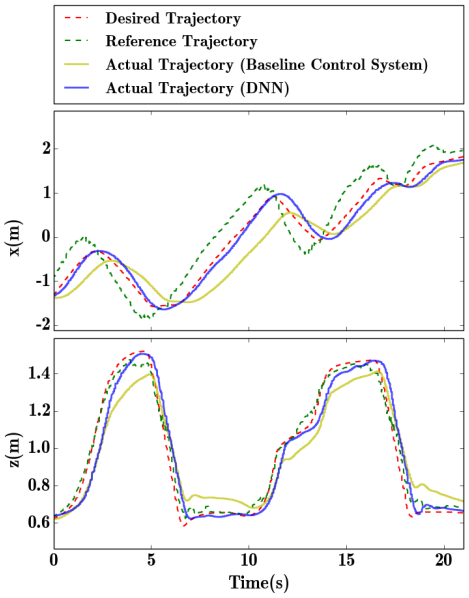

Fig. 5. The x and z performances of Figure 4. With the aid of the DNNs trained with future desired states, tracking error is significantly reduced in both axes. Note that the green dashed line shows the reference trajectory calculated by the DNNs. The graph of y performance is not shown since the y-components of velocity and position in the desired trajectory are zero. The RMS error for the proposed control system is 0.1481 m (x-axis), 0.0905 m (y-axis) and 0.0833 m (z-axis). The RMS error for the baseline control system is 0.401 m (x-axis), 0.1163 m (y-axis) and 0.1189 m (z-axis). The percentage improvement on each axis is 63% (x-axis), 22% (y-axis), 30% (z-axis).

in Section V-C. Table I highlights that the DNN with current state feedback performs better than the DNN without current state feedback, while both obtain considerable improvements over the baseline control system. The fact that the DNN without the current state feedback can still have a comparable improvement offers us an alternative offline method of improving tracking performance. It makes our approach more versatile, especially when computational resources are not sufficient to support real-time calculations during flights.

C. DNN Tracking Performance on Arbitrary Trajectories

To investigate generalizability of the trained DNNs on different trajectories, we evaluate the performance of the trained DNNs with future desired states and current state feedback on various trajectories. On a 50s segment from the 3-D training trajectory (as discussed in Section V-D), the DNNs outperform the baseline controller by 36%. The performance of the trained DNNs is also evaluated on unseen trajectories, including 30 different hand-drawn trajectories and one specific trajectory (Trajectory 4 on Figure 2) with different velocity profiles[4]. In Figure 7, we show that the DNNs with future desired states and the current state feedback reduce the RMS tracking errors by 43% on average over the 30 testing trajectories and training trajectory segment. Similar improvement is also obtained on one specific trajectory with different speeds as shown in Figure 6. Therefore, the DNNs trained with future desired states are capable of reducing the tracking error for trajectories with various shapes and speeds by a large margin, demonstrating its generalizability on different unseen trajectories. Figure 8 presents a long-exposure image of a quadrotor following the letters "DSL" written by a visitor. Note that this is different from Trajectory 5 shown in Figure 2 since this is the write-up of another visitor.

[4]The 30 drawn trajectories all have a maximum velocity of 0.6 m/s and a maximum acceleration of 2 m/s^2. For Trajectory 4, the speed is changed by scaling the time domain along the desired trajectory.

VII. CONCLUSIONS

In this paper, we have presented a DNN-based reference learning method, able to learn from flight data and improve trajectory tracking control in quadrotor systems. By introducing information about future desired states in training data, the DNNs were able to account for system delay and hidden dynamics as shown from the significant reduction in tracking error overall. The main advantages of this proposed approach shown from our experiments are that 1) this approach can be applied to various control systems with complex dynamics while ensuring stability of the systems; 2) it requires no prior knowledge of the system to train the DNNs, and the trained DNNs can be applied to any unseen trajectories without any adaptation process; and 3) with wise feature selection and sufficient DNN training, this approach can be computationally efficient with a very small model, while demonstrating good performance on general trajectories. We have shown these advantages through the implementation of an interactive "fly-as-you-draw" application, illustrating

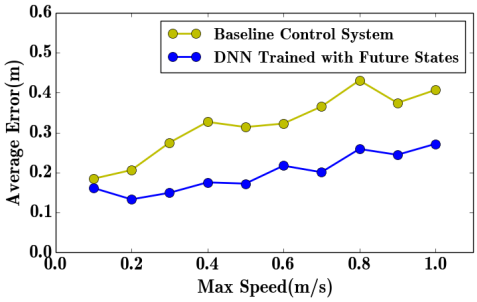

Fig. 6. The average performance of the DNNs trained with two future desired states (Δ_1 = 4, +0.6 s, and Δ_2 = 6, +0.9 s ahead of the current time) and the baseline control on Trajectory 4 in Figure 2 with different speeds. The trajectories with different speeds were obtained from the drawn trajectory by changing the velocity bound. For the same trajectory, DNNs are able to obtain consistent improvement in tracking performance over different speeds.

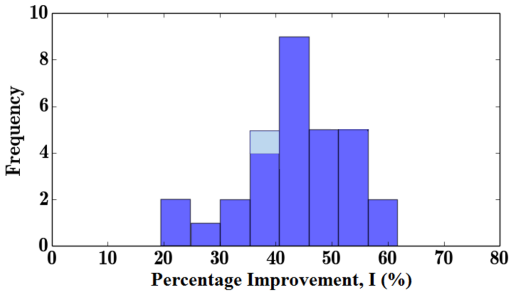

Fig. 7. The improvements, *I*, made by the proposed control system trained with two future desired states ($\Delta_1 = 4$, +0.6 s, and $\Delta_2 = 6$, +0.9 s ahead of the current time) compared to the baseline control on 30 testing trajectories and a 50 s segment from the training trajectory. DNNs achieved 36% improvement on the training trajectory segment and it is represented by light-blue block on the graph. On average, 43% improvement is obtained by the control system with DNNs.

that the proposed method was readily applicable to various real-world trajectory tracking tasks. However, the overall improvements of this method are still limited by training data for the DNNs. Intelligent choices of learning targets and effective neural network designs are potential extensions to enhance the trajectory tracking performance.

REFERENCES

[1] "Central American Drug Compound Recon," October 2010. [Online]. Available: https://www.aeryon.com/casestudies/centralamericadrug

[2] C. Gothner, "Deputies using drones as search-and-rescue tools," August 2016. [Online]. Available: http://www.channel3000.com/news/deputies-using-drones-as-searchandrescue-tools/41297542

[3] S. Shaw, "7-Eleven Teams with Flirtey for First Ever FAA-Approved Drone Delivery to Customer's Home," July 2016. [Online]. Available: http://corp.7-eleven.com/news/07-22-2016-7-eleven-teams-with-flirtey-for-first-ever-faa-approved-drone-delivery-to-customer-s-home

[4] N. Michael, J. Fink, and V. Kumar, "Cooperative manipulation and transportation with aerial robots," *Autonomous Robots*, vol. 30, no. 1, pp. 73–86, 2011.

[5] Q. Lindsey, N. Michael, D. Mellinger, and V. Kumar, "The grasp multiple micro-uav testbed," *IEEE Robotics & Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.

[6] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea, "A platform for aerial robotics research and demonstration: The Flying Machine Arena," *Mechatronics*, vol. 24, no. 1, pp. 41–54, 2014.

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

[8] N. Mohajerin and S. L. Waslander, "Modular deep Recurrent Neural Network: Application to quadrotors," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, pp. 1374–1379.

[9] M. T. Frye and R. S. Provence, "Direct Inverse Control using an Artificial Neural Network for the Autonomous Hover of a Helicopter," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, pp. 4121–4122.

[10] A. P. Schoellig, F. L. Mueller, and R. D'Andrea, "Optimization-based iterative learning for precise quadrocopter trajectory tracking," *Autonomous Robots*, vol. 33, pp. 103–127, 2012.

[11] F. L. Mueller, A. P. Schoellig, and R. D'Andrea, "Iterative learning of feed-forward corrections for high-performance tracking," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 3276–3281.

[12] M. W. Michael Hamer and R. D'Andrea, "Knowledge Transfer for High-Performance Quadrocopter Maneuvers," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 1714–1719.

[13] J. Mahler, S. Krishnan, M. Laskey, S. Sen, A. Murali, B. Kehoe, S. Patil, J. Wang, M. Franklin, P. Abbeel, *et al.*, "Learning accurate kinematic control of cable-driven surgical robots using data cleaning and gaussian process regression," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2014, pp. 532–539.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[15] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint, arXiv:1412.6980*, 2014.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

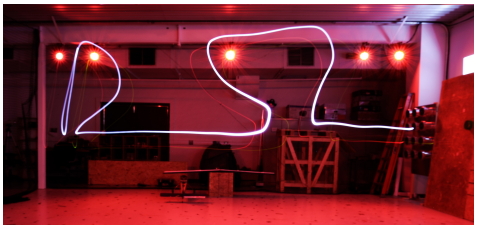

Fig. 8. A long exposure image of a quadrotor following the letters "DSL" with the aid of DNNs. The trajectory is drawn by a visitor. The DNNs take in 2 future desired states ($\Delta_1 = 4$, +0.6 s, and $\Delta_2 = 6$, +0.9 s) and the current state feedback.