Article

The International Journal of Robotics Research
1–22
© The Author(s) 2020
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/0278364920953902
journals.sagepub.com/home/ijr
SAGE

Deep neural networks as add-on modules for enhancing robot performance in impromptu trajectory tracking

Siqi Zhou[1], Mohamed K Helwa[1,2] and Angela P Schoellig[1]

Abstract

High-accuracy trajectory tracking is critical to many robotic applications, including search and rescue, advanced manufacturing, and industrial inspection, to name a few. Yet the unmodeled dynamics and parametric uncertainties of operating in such complex environments make it difficult to design controllers that are capable of accurately tracking arbitrary, feasible trajectories from the first attempt (i.e., impromptu trajectory tracking). This article proposes a platform-independent, learning-based "add-on" module to enhance the tracking performance of black-box control systems in impromptu tracking tasks. Our approach is to pre-cascade a deep neural network (DNN) to a stabilized baseline control system, in order to establish an identity mapping from the desired output to the actual output. Previous research involving quadrotors showed that, for 30 arbitrary hand-drawn trajectories, the DNN-enhancement control architecture reduces tracking errors by 43% on average, as compared with the baseline controller. In this article, we provide a platform-independent formulation and practical design guidelines for the DNN-enhancement approach. In particular, we: (1) characterize the underlying function of the DNN module; (2) identify necessary conditions for the approach to be effective; (3) provide theoretical insights into the stability of the overall DNN-enhancement control architecture; (4) derive a condition that supports data-efficient training of the DNN module; and (5) compare the novel theory-driven DNN design with the prior trial-and-error design using detailed quadrotor experiments. We show that, as compared with the prior trial-and-error design, the novel theory-driven design allows us to reduce the input dimension of the DNN by two thirds while achieving similar tracking performance.

Keywords

Motion control, learning and adaptive systems, aerial robotics, trajectory tracking, neural networks

1. Introduction

As continued advancements in algorithms, actuation, and sensor technology push robots into more complex environments, increasingly sophisticated methods for controlling robot motion are needed. In particular, controllers that are capable of high-accuracy trajectory tracking are becoming increasingly important in robot applications where safety and/or efficiency are essential. For example: in search and rescue, where robots must operate in close proximity to people (Liu and Nejat, 2013); in advanced manufacturing, where robot arms must efficiently follow pre-designed trajectories to perform complex manipulation tasks (Brogårdh, 2007); or in industrial inspection, where unmanned aerial vehicles fly in close proximity to facilities to enable visual inspection (Nikolic et al., 2013).

The trajectory tracking problem has been extensively studied in the control literature. Among various techniques, the proportional–integral–derivative (PID) controller is often used in trajectory tracking applications. However, tuning PID parameters is typically time-consuming, and the performance of a PID controller can be conservative (Åström and Hägglund, 2004). Moreover, control theory shows that a standard PID control architecture cannot achieve exact tracking for arbitrary trajectories (Francis and Wonham, 1976).

[1]University of Toronto Institute for Aerospace Studies, Toronto, Canada
[2]Electrical Power and Machines Department, Cairo University, Egypt

Corresponding author:
Siqi Zhou, University of Toronto Institute for Aerospace Studies, Dynamic Systems Lab, 4925 Dufferin Street, Toronto, Ontario, Canada, M3H 5T6.
Email: siqi.zhou@robotics.utias.utoronto.ca

In addition to the PID controller, model-based techniques such as model predictive control (MPC) have been studied for finding optimal control commands that lead to accurate and agile robot motions (Liniger et al., 2015). Moreover, inversion-based feedforward approaches have been widely applied to achieve high-accuracy tracking (Devasia et al., 1996; Hirschorn, 1979). However, one general limitation of model-based approaches is the reliance on a sufficiently accurate dynamic model of the system, which is difficult to obtain in practice. Adaptive control (Slotine and Li, 1987) and robust control (Spong, 1992) strategies have been used to address uncertainties in system parameters. Yet while these approaches typically guarantee stability, they do not take past experience into account, and the same errors are repeated from trial to trial if the same reference is given.

As robot dynamics and operating environments become ever more complex, researchers are increasingly turning to learning-based approaches to address the resulting model uncertainties. These learning-based approaches have been applied successfully to manipulators (Levine et al., 2015), bipedal robots (Da et al., 2017), autonomous cars (Drews et al., 2017), and unmanned aerial vehicles (Bansal et al., 2016; Tang and Kumar, 2018), to name a few. A common learning-based approach that yields high-accuracy tracking is iterative learning control (ILC). In ILC, the tracking performance is improved by adjusting control inputs or reference signals in repeated trials (Bristow et al., 2006; Schoellig et al., 2012; Tayebi, 2004). In addition to ILC, reinforcement learning (RL)-based approaches have also been proposed to iteratively optimize the tracking performance (Kiumarsi et al., 2014; Pane et al., 2016; Zhang et al., 2016a). Apart from iterative approaches, there are also various works on improving the tracking performance of classical model-based controllers by learning the uncertain or unknown system dynamics with techniques such as Gaussian processes (GPs) (Helwa et al., 2018; Nguyen-Tuong and Peters, 2008), neural networks (NNs) (He et al., 2016; Yan and Wang, 2014), and support vector machines (SVMs) (Iplikci, 2006). Alternatively, these learning techniques have also been applied to improve the tracking performance by approximating inverse dynamic models in inversion-based feedforward approaches (Nguyen-Tuong and Peters, 2010; Schaal et al., 2002).

In this article, we consider the impromptu tracking problem. That is, we aim to achieve high-accuracy tracking of arbitrary, feasible trajectories from the first attempt. Motivated by the success of the learning-based control approaches for robot control, we present a deep neural network (DNN)-based approach for enhancing the impromptu tracking control performance of black-box systems. This paper is motivated by our previous work (Li et al., 2017), in which a DNN add-on module was used to improve the performance of quadrotors in tracking arbitrary, hand-drawn trajectories. The proposed DNN-enhancement architecture is illustrated in Figure 1. During the training phase, the input, output, and state of the baseline system are recorded for training a DNN module. Then, during the testing phase, the DNN module is pre-cascaded to the baseline system to adapt the reference signals to establish an identity mapping from the desired output to the actual output. In the work of Li et al. (2017), experiments on 30 arbitrary, hand-drawn trajectories show that the DNN-enhancement control architecture effectively reduces the tracking error of the quadrotor vehicle by 43% on average as compared with the baseline controller. As compared with the other learning-based tracking control approaches, the proposed DNN-approach has the following advantages:

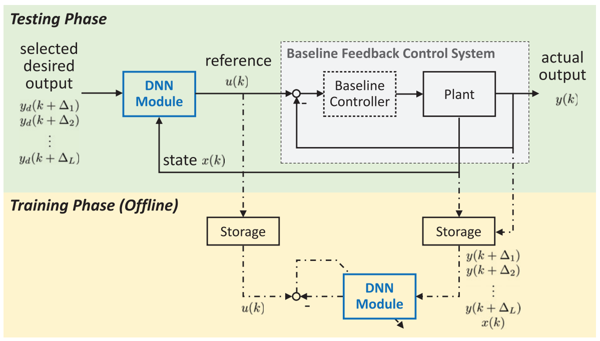


Fig. 1. The DNN-enhancement control architecture: during the training phase (shaded yellow region), a baseline system is treated as a black box, and the reference u, output y, and state x are recorded for training a DNN module. During the testing phase (shaded green region), the DNN module is pre-cascaded to the baseline system and adjusts the reference $u(k)$ based on the current state $x(k)$ and a set of selected future desired output $y_d(k+\Delta_i)$ to enhance the tracking performance of the baseline system, where $k \in \mathbb{Z}_{\geqslant 0}$ is the discrete-time index and $\Delta_i \in \mathbb{Z}_{>0}$.

- Unlike the iterative learning methods (ILC approaches and some RL-based approaches such as Pane et al. (2016)), the proposed DNN approach can be directly used for tracking arbitrary, feasible trajectories without further adaptations during the testing phase, and, consequently, it satisfies the impromptu tracking requirement.
- Compared to more common approaches (such as forward or inverse dynamic learning) where the learning component typically resides in the main control loop, we use the DNN module as an add-on block that is placed outside of the closed-loop system to improve the tracking performance. This add-on approach enables black-box control systems to be improved retrospectively.
- As discussed in the following section, the proposed approach is less prone to instability than other inverse-based approaches because the DNN loop can be run at a lower rate than the baseline control loop (Li et al., 2017). Moreover, the proposed architecture can potentially lead to better learning-enhanced performance as

the closed-loop system has a more repeatable behavior (Mueller et al., 2012).

While experiments from Li et al. (2017) have shown that the DNN approach shown in Figure 1 is effective for quadrotors, the DNN module was designed by trial and error, and guidelines for systematically applying the approach to other robotic platforms were not given. In Zhou et al. (2017), we presented preliminary theoretical results on the DNN-based approach based on a single-input–single-output (SISO) system formulation. In this article, we provide a comprehensive theoretical study of the DNN-based approach and support it with both simulations and extensive experimental results. In particular, based on a multi-input–multi-output (MIMO) system formulation, our study in this work includes: (1) characterizing the underlying function represented by the DNN module, (2) identifying a necessary condition for the DNN approach to be effective, (3) deriving a condition that allows for further improving the DNN training data efficiency, and (4) analyzing the stability of the overall DNN-enhanced system given the presence of modeling errors in the DNN module. These theoretical insights are illustrated in simulation and verified with extensive quadrotor experiments.

We note that we focus our attention on minimum phase systems in this work; in Zhou et al. (2018), we discussed a separate issue of applying the DNN approach to non-minimum phase systems (i.e., systems with unstable inverse dynamics).

We also note that, in the previous work (Li et al., 2017), a DNN was chosen as the learning technique to construct the add-on block. This design decision was motivated by the fact that the amount of memory and computational cost of the forward pass of the DNN is fixed as more data is collected. In contrast to nonlinear regression methods such as GPs, the relatively fixed memory and computational cost allow the DNN model to be implemented on robot platforms where onboard computational resources are limited (Li et al., 2017). Following Li et al. (2017), we use DNNs as the learning technique in this work; however, the presented theoretical insights can be potentially generalized to other nonlinear regression techniques.

2. Related work on NN-based inverse control

The DNN module in the proposed control architecture (Figure 1) aims to establish an identity mapping from the desired output to the actual output (Li et al., 2017). In the literature of NN-based control, common approaches that have a similar objective include *direct inverse control, feedback-error learning control*, and *adaptive inverse control*.

In direct inverse control, an NN is trained to approximate the inverse dynamics of the open-loop plant, and is pre-cascaded to the plant as the controller to achieve exact tracking (Hunt et al., 1992; Suprijono et al., 2015). Early literature such as Jordan and Rumelhart (1992) and Kawato (1990) compared different approaches for training the NN inverse model and discussed details concerning practical implementation. For example, Jordan and Rumelhart (1992) pointed out that an NN directly trained with the reversed input–output data from the open-loop plant is not "goal-directed": the training objective of minimizing the regression error of the model output does not directly reflect the control objective of minimizing the tracking error of the system. To address these concerns, training schemes such as the distal teacher (Jordan and Rumelhart, 1992) have been proposed. However, apart from these discussions, a fundamental drawback of the direct inverse control approach is the lack of robustness against disturbances in the system. This drawback is attributed to the fact that the NN inverse model is often used as the only controller of the system.

To address the issues with direct inverse control, Kawato (1990) proposed a feedback-error learning scheme. This approach employs a feedback control loop, where the input command to the plant is the sum of the signal from the feedback controller and feedforward signal from an NN-based inverse model. In contrast with typical direct inverse control, the error signal for training the NN is the output of the feedback controller instead of the typical regression errors based on the plant input–output data. Although practical considerations such as the "goal-directness" issue and robustness issue are addressed in the feedback error learning approach, the training of the NN requires a plant in the loop, which may not be desired in the early training phase.

Another inversion-based approach for trajectory tracking problems is adaptive inverse control, in which the parameters of an NN controller are updated online with guaranteed stability (Chen and Khalil, 1995; Ge and Zhang, 2003; Zhang et al., 2016b). A limitation of the adaptive-NN approach is that the number of adaptive parameters and, hence, the online computational cost, can be large. Moreover, an appropriate initialization for the NN parameters are typically needed for convergence (Chen and Khalil, 1995).

Overall, despite the similarity in the control objective, there are fundamental differences between the proposed DNN control architecture in Figure 1 and the common NN-based inverse control architectures. One of the differences is that the proposed DNN control architecture modifies the reference of a stabilized closed-loop system, while the common NN-based inverse control approaches directly modify the input to the open-loop plant. From a practical perspective, this difference has two potential benefits: (i) by introducing the DNN as an outer loop that runs at a lower rate as compared with the baseline system, the overall approach is less prone to stability issues (Li et al., 2017); (ii) as the closed-loop system partially compensates for non-repeated disturbances, the response of the closed-loop system is more repeatable than that of the open-loop plant (Mueller et al., 2012). Thus, learning to adapt the reference of a closed-loop system can be potentially more effective for achieving good tracking performance. In contrast to adaptive inverse control, in which high-accuracy tracking control and stability of the plant are simultaneously achieved by the designed NN parameter update laws, the proposed DNN approach achieves stabilization through the design of the baseline controller, and tracking performance is enhanced separately by the pre-cascaded DNN module. This approach of decoupling the stabilization and tracking performance enhancement problems can greatly simplify the DNN design and training in practical applications. We furthermore investigate the effectiveness of the proposed DNN approach for the problem of impromptu tracking, and verify this experimentally by testing whether quadrotors are able to accurately fly arbitrary, hand-drawn trajectories from the first attempt. Although the NN-based inverse control approaches in the literature provide theoretical foundations for designing high-accuracy tracking controllers, their ability to track arbitrary, feasible trajectories has not been thoroughly demonstrated in experiments.

3. Problem formulation

Our objective is to enhance black-box control systems to achieve high-accuracy, impromptu tracking. In our previous work (Li et al., 2017), with quadrotors as the test platform, a DNN-enhancement control architecture (Figure 1) was proposed to establish an identity mapping from the desired output y_d to the actual output y. In this work, we aim to provide a platform-independent formulation of the proposed DNN-enhancement control architecture (Li et al., 2017). This formulation includes:

(O1) identifying the underlying function that should be represented by the DNN module in order to establish an identity mapping from y_d to y;
(O2) identifying necessary conditions for the approach to be effective;
(O3) deriving guidelines for systematically selecting the inputs and outputs of the DNN module;
(O4) analyzing the stability of the DNN-enhanced system in the presence of regression errors; and
(O5) characterizing a condition that allows for further improving data efficiency of the DNN training.

In the following discussion, we first consider linear time invariant (LTI) MIMO baseline systems represented by the following state space model

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned} \tag{1}$$

where $k \in \mathbb{Z}_{\geqslant 0}$ denotes the discrete-time index, $x \in \mathbb{R}^n$ is the system state, $u \in \mathbb{R}^m$ is the reference signal sent to the baseline system, $y \in \mathbb{R}^m$ is the system output, and A, B, and C are constant matrices of appropriate dimensions. After presenting the insights from the linear system formulation, we then extend the discussion to nonlinear MIMO baseline systems represented by

$$\begin{aligned} x(k+1) &= f(x(k)) + g(x(k))u(k) \\ y(k) &= h(x(k)) \end{aligned} \tag{2}$$

where $f(\cdot)$, $g(\cdot)$, and $h(\cdot)$ are matrices of smooth functions with appropriate dimensions. Note that, in the discussion of this work, we focus on square MIMO systems having the same number of inputs and outputs. This is not a restrictive formulation for tracking applications, because systems (1) and (2) typically represent baseline closed-loop systems, and each output in y has a corresponding reference input in u.

In deriving the theoretical insights for this article, we make the following assumptions:

(A1) the baseline system is input-to-state stable (Jiang and Wang, 2001); for the nonlinear system (2), we additionally assume that the state can be bounded by

$$\|x\|_\infty \leqslant L_1\ \|u\|_\infty + L_2\ \|x_0\|\ + L_3 \tag{3}$$

where $\|\cdot\|_\infty$ denotes the infinity norm, $x_0 \in \mathbb{R}^n$ is the initial state, and L_1, L_2, and L_3 are constant, positive scalars;
(A2) at any instant k, a preview of n future time steps of the desired trajectory (i.e., $\{y_d(k), y_d(k+1), \ldots, y_d(k+n)\}$) is available, where n is the system order;
(A3) the DNN module has a feedforward architecture and globally Lipschitz activation functions.

Note that assumptions (A1)–(A3) are not restrictive. Assumption (A1) on the stability of the baseline closed-loop system can be achieved by proper controller designs with well-developed control techniques even in the absence of a detailed or highly accurate dynamic model of the system. The inequality (3) in assumption (A1) holds for input-to-state stable linear systems; for nonlinear systems, this is an additional assumption that we use to provide a theoretical guarantee on stability of the overall control system. For assumption (A2), a preview of n steps of the desired trajectory is usually available in practice, and does not prevent combinations with online trajectory generation algorithms. For assumption (A3), although we use feedforward neural networks (FNNs) in this work, the proposed approach can be potentially adapted for use with other nonlinear regression techniques (e.g., GPs, recurrent NNs). The globally Lipschitz condition in assumption (A3) holds for the commonly-used activation functions such as the rectified linear unit (ReLU), sigmoid, and hyperbolic tangent.

4. Theoretical insights

In this section, we provide four theoretical insights to achieve the objectives (O1)–(O5) stated in Section 3. We begin our discussion with a background on the inversion of dynamic systems in Section 4.1. We then build on this conceptual overview in Section 4.2 to derive the underlying function to be modeled by the DNN module to establish an identity mapping between the desired and actual outputs, and identify conditions that are necessary for the proposed DNN approach to be effective. Building on the insight regarding the underlying function modeled by the DNN module, we provide guidelines for systematically selecting the inputs and outputs of the DNN module in Section 4.3, provide a proof of stability of the overall control system in the presence of DNN regression errors in Section 4.4, and derive a condition that allows us to further improve the data-efficiency of the DNN training in Section 4.5.

4.1. Background on system inversion

Starting with the DNN-enhancement control architecture in Figure 1, Li et al. (2017) initially designed a DNN module with $y_d(k)$ and $x(k)$ as input and $u(k)$ as output to enhance the tracking performance of the quadrotor baseline control system. The experiments of Li et al. (2017) show that the DNN module is able to enhance the tracking performance of the baseline system only after $y_d(k)$ in the DNN input is replaced by certain future desired outputs $\{y_d(k+\Delta_1), y_d(k+\Delta_2), \ldots, y_d(k+\Delta_L)\}$ with $\Delta_1, \Delta_2, \ldots, \Delta_L \in \mathbb{Z}_{>0}$ selected based on trial and error. As shown in Section 4.2, this experimental observation can be explained by associating the DNN module with the inverse dynamics of the baseline system.

In order to facilitate the following discussions, in this subsection, we state the formal definition of the *vector relative degree* (Isidori, 1995; Jang et al., 1994), and discuss its connection to the system inverse. In the following discussions, we use $h \circ f$ to denote the composition of the functions h and f, and f^i to denote the ith composition of the function f with $f^0(x(k)) = x(k)$ and $f^i(x(k)) = f \circ f^{i-1}(x(k))$.

Definition 4.1 (Vector relative degree). *The nonlinear MIMO system (2) has a vector relative degree $(r_1, r_2, \ldots, r_m)$ at an operating point (x_0, u_0) if*

(i) $\frac{\partial}{\partial u_j} h_i \circ f^p(f(x) + g(x)u) = 0$, $\forall i = \{1, 2, \ldots, m\}$, $\forall p = \{1, 2, \ldots, r_i - 2\}$, $\forall j = \{1, 2, \ldots, m\}$ *for every point (x, u) in some neighborhood of (x_0, u_0), where u_j is the jth element of the input u, and h_i is the ith element of the vector function h; and*

(ii) *the decoupling matrix $D_n(x, u) \in \mathbb{R}^{m \times m}$ with elements $[D_n(x, u)]_{ij} = \frac{\partial}{\partial u_j} h_i \circ f^{r_i - 1}(f(x) + g(x)u)$ has full rank at the operating point*.

Note that, from the first condition (i) of Definition 4.1, if we focus on an output dimension y_i, the relative degree r_i can be interpreted as the number of sample delays between changing any of the inputs u_j, $j = 1, \ldots, m$, and changing the output y_i. Given that both (i) and (ii) of Definition 4.1 are satisfied, the relative degree r_i associates the value of an output y_i at time step $k + r_i$ with a non-zero input u applied at time step k. The decoupling matrix $D_n(x, u)$ in the second condition (ii) of Definition 4.1 is the collection of the Jacobian of $y_i(k + r_i)$ with respect to the input u; the non-singularity condition requires that the outputs $y(k + r) = [y_1(k + r_1) \ \cdots \ y_m(k + r_m)]^{\mathrm{T}}$ are influenced by the input $u(k)$ in non-repeated (linearly independent) ways.

Remark 4.1 (Vector relative degree for linear systems). *As a the special case of Definition 4.1, the linear MIMO system (1) has a vector relative degree $(r_1, r_2, \ldots, r_m)$ if:*

(i) $C_i A^p B_j = 0$, $\forall i = \{1, 2, \ldots, m\}$, $\forall p = \{1, 2, \ldots, r_i - 2\}$, $\forall j = \{1, 2, \ldots, m\}$, *where C_i is the ith row of the matrix C and B_j is the jth column of the matrix B; and*

(ii) *the decoupling matrix $D_l \in \mathbb{R}^{m \times m}$ with elements $[D_l]_{ij} = C_i A^{r_i - 1} B_j$ has full rank.*

Note that, from Definition 4.1, for MIMO systems with a well-defined vector relative degree, one may relate the future output $y(k + r)$ to the current state $x(k)$ and input $u(k)$. Based on this concept, in the following subsections we formalize the DNN-based approach proposed in Li et al. (2017), develop theoretical insights for systematically designing the DNN module, and provide comments on its practical implementation.

4.2. Underlying function modeled by the DNN module

In this subsection, we show that, given the system representations in (1) and (2), an identity mapping from the desired output y_d to the actual output y is achieved if the DNN module learns the output equation of the inverse dynamics of the baseline system. Owing to this association with inverse dynamics, a necessary condition for the proposed approach to be effective is that the baseline system has stable inverse dynamics. For simplicity, we start our discussion with the linear system (1) and then extend the results to the nonlinear system (2). Note that although we start our discussion with known system models, we later demonstrate that implementing the proposed DNN-enhancement approach requires only minimal knowledge about the baseline system (e.g., its order and relative degree). This required knowledge can typically be determined from simple dynamic models or step response experiments.

By applying the definition of the vector relative degree in Remark 4.1 to the linear system (1), we can relate the input u and the output y of the baseline system by

$$y_i(k + r_i) = C_i A^{r_i} x(k) + C_i A^{r_i - 1} B u(k) \tag{4}$$

or, in augmented form,

$$y(k+r) = C_l x(k) + D_l u(k) \tag{5}$$

where $y(k+r)=[y_1(k+r_1)\cdots y_m(k+r_m)]^{\mathrm{T}}$, $C_l = [(C_1 A^{r_1})^{\mathrm{T}} \cdots (C_m A^{r_m})^{\mathrm{T}}]^{\mathrm{T}}$, and D_l is the decoupling matrix of system (1).

Let $y_d(k+r) = [y_{1,d}(k+r_1) \ \cdots \ y_{m,d}(k+r_m)]^{\mathrm{T}}$ be the desired output corresponding to $y(k+r)$. As the decoupling matrix D_l has full rank by condition (ii) of the vector relative degree definition in Remark 4.1, it can be shown that if we choose the following control law

$$u(k) = D_l^{-1}(-C_l x(k) + y_d(k+r)) \tag{6}$$

then $y(k+r) = y_d(k+r)$, or exact tracking, is achieved. Thus, for the proposed DNN-enhancement control architecture in Figure 1 and system (1), the DNN module should be trained to approximate (6) to establish an identity mapping between y_d and y. If we consider y_d as the input and u as the output, Equation (6) is, in fact, the output equation of the inverse dynamics of system (1).

Note that the first condition *(i)* in Remark 4.1 implies that the relative degree r_i associated with the output dimension i is the smallest integer such that $C_i A^{r_i-1} B_j \neq 0$ for any input dimension j. As briefly noted in Section 4.1, the relative degree r_i is the number of sample delays between applying an input u to the system and first seeing its effect in the particular output y_i. This inherent delay from input to output is a well-known fact for discrete-time linear systems. By training the DNN module to approximate (6), the inherent delay of the system is compensated for by the preview of the future desired output $y_d(k+r)$. In practice, at a particular time k, a preview of r steps of the desired trajectory (where $r \leqslant n$) is not challenging to satisfy with online or offline trajectory generation algorithms; the non-causality in (6) is, thus, not an issue in practical applications.

We next generalize the previous discussion to nonlinear systems. By assuming the system (2) has a well-defined vector relative degree, and applying Definition 4.1, we can relate the input u and output y of the nonlinear MIMO system (2) by

$$y_i(k+r_i) = h_i \circ f^{r_i-1}(f(x(k)) + g(x(k))u(k)) \tag{7}$$

or, in an augmented form,

$$y(k+r) = h \circ f^{r-1}(f(x(k)) + g(x(k))u(k)) \tag{8}$$

where $h \circ f^{r-1}$ is a vector of composition functions with the ith element being $h_i \circ f^{r_i-1}$. As discussed in Sun and Wang (2001) and Jang et al. (1994), by assuming $y(k+r)$ is affine in the input $u(k)$, the decoupling matrix $D_n(x,u)$ is independent of u and (8) becomes

$$y(k+r) = h \circ f^{r}(x(k)) + D_n(x(k))u(k) \tag{9}$$

where $h \circ f^r$ is a composite function with the ith element being $h_i \circ f^{r_i}$. This special case holds for nonlinear mechanical systems such as robot manipulators (Jang et al., 1994). As the decoupling matrix D_n has full rank by the second condition (ii) in Definition 4.1, exact tracking (i.e., $y(k+r) = y_d(k+r)$) can be achieved by choosing the control law

$$u(k) = [D_n(x(k))]^{-1}(-h \circ f^r(x(k)) + y_d(k+r)) \tag{10}$$

for the affine case in (9), and it is reasonable to assume that

$$u(k) = F(x(k), y_d(k+r)) \tag{11}$$

for the general case in (8), where $F: \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^m$ is a vector of nonlinear functions.

Based on the above results, we now present our insight on the underlying function modeled by the DNN module, and describe the conditions that are necessary for the learning-based approach to be effective.

Insight 4.1 (Underlying function and necessary conditions). *Consider the DNN-enhancement control architecture in Figure 1. In order to establish an identity mapping between the desired output y_d and the actual output y, the DNN module should approximate the output equation of the baseline system's inverse dynamics. Owing to the association with inverse dynamics, two necessary conditions for the learning approach to be effective are: (i) the baseline system has a well-defined (vector) relative degree; and (ii) the baseline system has stable zero dynamics.*

By inspecting the control laws in (6) and (11), it can be seen that the ideal control law that leads to exact tracking is dependent on the current $x(k)$ and the future desired output $y_d(k+r)$ for either the linear or nonlinear case, where r is the vector relative degree. In practice, when training the DNN module to approximate the control law for achieving exact tracking, we do not require a detailed dynamic model of the system. Instead, we need only identify the vector relative degree r of the baseline system. Experimentally, for the linear system (1) and the special case of the nonlinear system (2) where $y(k+r)$ is affine in $u(k)$, one can identify the vector relative degree of the baseline system through m step response experiments detailed as follows. In each of the m experiments, the system is initialized at an equilibrium point, and one element of the input, u_j, is activated. Without loss of generality, we assume the equilibrium is the origin. After the m experiments, one may determine the minimum number of time delays between the output y_i and the inputs u_j for all j; the minimum number of time delays for the output dimension y_i is the estimated relative degree r_i associated with the particular output dimension. After estimating the relative degree for each output dimension, it remains to check the non-singularity condition *(ii)* in Definition 4.1. From the m experiments, one may construct a matrix $\widetilde{D}$, where the jth column of $\widetilde{D}$ is $[y_1(r_1) \cdots y_m(r_m)]^{\mathrm{T}}$ from the jth experiment. By inspecting (5) and (9), it can be shown that the non-singularity condition (ii) in Definition 4.1 can be examined from the rank of $\widetilde{D}$.

The stability of the zero dynamics of the linear system (1) is equivalent to the stability of the system's inverse dynamics, and is characterized by the zeros of the system transfer function. In practice, for linear systems, we may infer the stability of zero dynamics from characteristics of the system's step responses such as undershoot and zero crossings (Hoagg and Bernstein, 2007). The zero dynamics of the nonlinear system (2) is the system's invariant dynamics when the input $u(k)$ is chosen such that $y(k)=0$ for all k. For nonlinear systems, achieving stable zero dynamics is a necessary but not sufficient condition for achieving stable inverse dynamics (Sussmann, 1990). Hence, a necessary condition for applying the proposed DNN-learning approach to either the linear system (1) or the nonlinear system (2) is that the baseline system has stable zero dynamics.

4.3. DNN input selection

In this subsection, we identify the necessary and sufficient inputs of the DNN module to compute the reference $u(k)$ of the baseline system (1) and (2) to achieve exact tracking. By designating the output of the DNN module as $\mathcal{O}=\{u(k)\}$, we can determine the appropriate DNN input $\mathcal{I}$ for either the linear or the nonlinear case based on the following insight.

Insight 4.2 (DNN input selection). *In order to establish an identity mapping from y_d to y, the necessary and sufficient input of the DNN add-on module is $\mathcal{I}=\{x(k),\ y_d(k+r)\}$, where $y_d(k+r)=[y_{1,d}(k+r_1)\ \cdots\ y_{m,d}\ \ (k+r_m)]^{\mathrm{T}}$ and $r=(r_1,\dots,r_m)$ is the vector relative degree of the system.*

Insight 4.2 directly follows from the fact that the DNN should approximate the baseline system inverse to achieve unity mapping between y_d and y and from (6) and (11) of the system inverse.

The implementation of Insight 4.2 requires knowledge or estimation of the full state of the system x. In many robotics applications, linearization techniques are used for the baseline system controller designs, and this often leads to decoupled linear dynamics. Some examples include ground vehicles in which the dynamics in the two-dimensional position space can be converted into decoupled integrators with the point-ahead linearization technique (Giesbrecht et al., 2009), and fully actuated manipulators in which the dynamics in the joint space can be turned into decoupled double integrators with feedback linearization (Helwa and Schoellig, 2016). In cases where the full state of the system is not available, but where the closed-loop dynamics can be approximated as a decoupled MIMO linear system, we can derive an alternative DNN input selection.

In deriving the alternative input selection, we first equivalently represent system (1) by $Y(z)=H(z)U(z)$, where

$$H(z)=C(zI-A)^{-1}B \tag{12}$$

and $U(z)$ and $Y(z)$ are the z-transform of the input and output of the baseline system, respectively. To show the main idea, we first consider the special case of a SISO linear system (i.e., $m=1$). Without loss of generality, we assume that the SISO system is represented by a transfer function of the following form:

$$H(z)=\frac{Y(z)}{U(z)}=\frac{\beta_{n-r}z^{n-r}+\beta_{n-r-1}z^{n-r-1}+\cdots+\beta_0}{z^n+\alpha_{n-1}z^{n-1}+\cdots+\alpha_0} \tag{13}$$

where α_i and β_i are scalar constants, and r and n are the relative degree and degree of the system, respectively. By calculating the inverse system of (13) and applying inverse z-transformation, it can be shown that the reference $u(k)$ for achieving exact tracking is

$$\begin{aligned}u(k)=&\frac{1}{\beta_{n-r}}y_d(k+r)+\frac{\alpha_{n-1}}{\beta_{n-r}}y_d(k+r-1)+\cdots\\&+\frac{\alpha_0}{\beta_{n-r}}y_d(k-n+r)-\frac{\beta_{n-r-1}}{\beta_{n-r}}u(k-1)\\&-\frac{\beta_{n-r-2}}{\beta_{n-r}}u(k-2)-\cdots-\frac{\beta_0}{\beta_{n-r}}u(k-n+r)\end{aligned} \tag{14}$$

Based on (14), we can alternatively select the DNN input for a SISO linear baseline system to be $\mathcal{I}=\{y_d(k-n+r:k+r),u(k-n+r:k-1)\}$, where the column ":" abbreviates consecutive discrete-time indexes.

The transfer matrix $H(z)$ of a decoupled MIMO linear system (1) is a diagonal matrix; the dynamics between each input–output pair (u_i,y_i) can be considered separately, where $i\in\{1,\dots,m\}$. As outlined in Section 4.2, one can execute m experiments to identify the relative degree r_i for each output y_i. Similar to the SISO scenario discussed above, in the case of the decoupled MIMO linear system, we can consider each reference dimension separately and train m networks with the input of each network being $\mathcal{I}_i=\{y_{d,i}(k-n+r_i:k+r_i),u_i(k-n+r_i:k-1)\}$ and output being $\mathcal{O}=\{u_i(k)\}$, where r_i is the relative degree corresponding to the ith output dimension, and $y_{d,i}$ denotes the ith desired output dimension.

Insight 4.3 (Alternative input selection for decoupled MIMO linear systems). *Based on the transfer function formulation, we can derive an alternative, sufficient input selection of the DNN module for a decoupled MIMO linear system. For this case, we propose using a DNN module with m independent networks: one for each of the baseline system reference dimensions. The input to the ith network in the DNN module is $\mathcal{I}_i=\{y_{d,i}(k-n+r_i:k+r_i),$*

$u_i(k - n + r_i : k - 1)\}$, where r_i is the relative degree corresponding to the output dimension y_i.

In comparison with the input selection based on the state space representation (Insight 4.2), the implementation of the alternative input for the linear systems does not require the estimation of the full state $x(k)$ of the system and instead only requires the identification of the order of the system n, which (for robots such as multi-link manipulators) can be determined from the laws of physics. Note, however, that this transfer function approach is derived for decoupled linear systems; the state space approach is applicable to more general cases. When the state of the system is available, applying the state space approach has additional advantages. One advantage of the state space approach is the current state feedback to the DNN module. This additional feedback from the baseline system can help compensate for the initial errors and disturbances along the trajectory. Moreover, the input selection based on the state space approach typically leads to a DNN with a lower input dimension than the transfer function approach. As an example, for a SISO linear system, the dimension of the DNN inputs derived from the transfer function and the state space approaches are $(2n - r + 1)$ and $(n + 1)$, respectively. This reduced DNN input dimension implies that the amount of data required to cover the operational space is potentially less, and thus the DNN training can be made more efficient by using the state space approach.

4.4. Stability

In this subsection, we restrict our discussion to minimum phase systems (i.e., systems with stable inverse dynamics), and prove the stability of the overall DNN-enhancement control system in the presence of DNN modeling errors:

$$\| u(k) - \hat{u}(k) \| \neq 0 \tag{15}$$

where $u(k)$ corresponds to the exact inverse in (11) (respectively (6) for the linear system case) and $\hat{u}(k)$ corresponds to the reference outputted by the DNN module trained based on the system input–output data. Note that, in the ideal case, where the DNN models the inverse dynamics exactly, the response from the desired output to the actual output is the identity mapping, and the overall system is input-to-state stable. However, in the presence of modeling errors, owing to the state feedback connection to the DNN module (see Figure 1), the stability of the overall system needs to be assessed. In this subsection, we show that under Assumptions (A1) and (A3), the DNN-enhanced system with the proposed input selection as in Insight 4.2 is input-to-state stable if the regression error of the DNN module is sufficiently small.

By assumption (A3), the DNN module has a feedforward architecture, and the activation functions are globally Lipschitz; because the DNN is a composite of linear combinations of Lipschitz functions, the output of the DNN module, $\hat{u}$, is globally Lipschitz in its inputs, x and y_d. In particular, we can bound the output of the DNN module by

$$\| \hat{u} \|_\infty \leq L_4 \, \| x \|_\infty + L_5 \, \| y_d \|_\infty \tag{16}$$

where L_4 and L_5 are positive, constant scalars. Moreover, because we consider a baseline system that is minimum phase (i.e., has a stable inverse dynamics), the reference $u(k)$ corresponding to the exact inverse in (11) is bounded (i.e., $\| u \|_\infty < \infty$). As a result of the global Lipschitz condition of the DNN module in (16) and the boundedness of u, an upper bound on the modeling error of the DNN module can be derived as follows:

$$\| u - \hat{u} \|_\infty \leq \| \hat{u} \|_\infty + \| u \|_\infty \tag{17}$$

$$\leq L_4 \, \| x \|_\infty + L_5 \, \| y_d \|_\infty + L_6 \tag{18}$$

where $L_6 = \| u \|_\infty$ is the bound on the exact inverse reference u of the minimum phase baseline system.

Lemma 4.1 (Stability). *Consider the DNN-enhancement control architecture (Figure 1) and the case where the baseline system is minimum phase. Under assumptions (A1) and (A3), the overall DNN-enhanced system is input-to-state stable if $L_1 L_4 < 1$, where L_1 and L_4 are constant scalars defined in (3) and (18), respectively.*

Proof. By assumption (A1), the baseline system is input-to-state stable, and with $\hat{u}$ as the system input, the state of the system is bounded by

$$\| x \|_\infty \leq L_1 \, \| \hat{u} \|_\infty + L_2 \, \| x_0 \| + L_3 \tag{19}$$

By combining the bound on the regression error in (18) and the bound on state in (19), the following is obtained:

$$\| x \|_\infty \leq L_1 \, \| u - \hat{u} \|_\infty + L_1 \, \| u \|_\infty + L_2 \, \| x_0 \| + L_3 \tag{20}$$

$$\leq L_1 L_4 \, \| x \|_\infty + L_1 L_5 \, \| y_d \|_\infty + L_7 \tag{21}$$

where $L_7 = L_1 L_6 + L_1 \, \| u \|_\infty + L_2 \, \| x_0 \| + L_3$. Based on (21), if

$$L_4 < \frac{1}{L_1} \tag{22}$$

is satisfied, then the state of the system is bounded by

$$\| x \|_\infty \leq \frac{L_1 L_5 \, \| y_d \|_\infty + L_7}{1 - L_1 L_4} \tag{23}$$

which is bounded by a constant for bounded input y_d. Thus, if $L_4 < \frac{1}{L_1}$, then the DNN-enhanced system is input-to-state stable. □

Note that, by examining (19) and (18), L_1 is a constant characterizing the maximum possible gain of the baseline system, whereas L_4 is a constant associated with the regression error of the DNN model. Hence, the condition in (22) implies that if the regression error of the DNN module is sufficiently small, then the overall DNN-enhancement control architecture is input-to-state stable.

One can note the similarity between condition (22) and the well-known small gain theorem in robust control (Francis and Khargonekar, 1995).

Also note that the globally Lipschitz activation function assumption in assumption (A3) is sufficient to show (18) but not necessary; condition (18), and hence the proof of the lemma, may be satisfied in other scenarios.

4.5. Difference learning scheme for improving the training efficiency

In this subsection, we derive a condition that allows us to further improve the data-efficiency of the proposed DNN-enhancement approach. This discussion is motivated by the DNN design in Li et al. (2017), where the position terms in the DNN input and output are taken relative to the current desired and actual positions in order to simplify the training process. The basic idea of this difference learning scheme is that with the relative positions (instead of the absolute positions), the function modeled by the DNN becomes invariant under spatial translations, which reduces the amount of data needed to cover the operation space. Based on the theoretical formulations presented in the previous subsections, we derive in this section a necessary condition for the effectiveness of the difference learning scheme. This necessary condition will be further illustrated with quadrotor experiments in Section 6.

In order to motivate this insight on the difference learning scheme, we first focus our discussion on a SISO linear system represented by the transfer function representation in (13). Recall that, for system (13), the control law for achieving exact tracking is described by (14), and the corresponding DNN input–output selection for learning the system inverse is $\mathcal{I}=\{y_d(k-n+r:k+r), u(k-n+r:k-1)\}$ and $\mathcal{O}=\{u(k)\}$, where ":" is used to abbreviate consecutive time indexes. With the difference learning scheme, we aim to train a DNN that depends only on a set of relative terms: $\Delta_{y_d}(k+p):=y_d(k+p)-y_d(k)$ for $p\in\{-n+r,\ldots,r\}$ and $\Delta_u(k+p):=u(k+p)-y_d(k)$ for $p\in\{-n+r,\ldots,0\}$, where y_d is the desired output, u is the reference of the baseline system, k is the current time index, and p is a shift in the time index. Note that, in this work, we aim to enhance the tracking performance of square MIMO baseline systems, and we assume that there is a one-to-one correspondence between the reference u and the output y, and hence a one-to-one correspondence between the reference u and the desired output y_d. For a position tracking system as an example, the terms Δ_{y_d} and Δ_u can be intuitively interpreted as the relative position vectors from the current desired position $y_d(k)$ to a past/future desired position $y_d(k+p)$ and a past reference position $u(k+p)$.

Lemma 4.2 (Difference learning for SISO linear systems). *Consider a SISO linear baseline system (13) and the DNN-enhancement control architecture in Figure 1. A difference learning scheme can be applied to improve the data efficiency of the DNN module if and only if the baseline system has a unity DC gain.*

Proof. Starting from the control law in (14), it can be shown that by subtracting $y_d(k)$ on both sides of the equation, and adding and subtracting $\frac{1}{\beta_{n-r}}y_d(k)$, $\frac{1}{\beta_{n-r}}\sum_{i=0}^{n-r-1}\beta_i y_d(k)$ and $\frac{1}{\beta_{n-r}}\sum_{i=0}^{n-1}\alpha_i y_d(k)$ on the right-hand side, Equation (14) can be written as

$$\begin{aligned}\Delta_u(k) &= \tfrac{1}{\beta_{n-r}}\Delta_{y_d}(k+r)+\tfrac{\alpha_{n-1}}{\beta_{n-r}}\Delta_{y_d}(k+r-1)+\cdots\\ &\quad+\tfrac{\alpha_0}{\beta_{n-r}}\Delta_{y_d}(k-n+r)-\tfrac{\beta_{n-r-1}}{\beta_{n-r}}\Delta_u(k-1)\\ &\quad-\tfrac{\beta_{n-r-2}}{\beta_{n-r}}\Delta_u(k-2)-\cdots-\tfrac{\beta_0}{\beta_{n-r}}\Delta_u(k-n+r)\\ &\quad+\underbrace{\frac{1}{\beta_{n-r}}\left(1-\sum_{i=0}^{n-r}\beta_i+\sum_{i=0}^{n-1}\alpha_i\right)y_d(k)}_{\triangleq s(y_d(k))}\end{aligned}\tag{24}$$

In this expression, the only non-relative time-dependent term is the last term

$$s(y_d(k))=\frac{1}{\beta_{n-r}}\left(1-\sum_{i=0}^{n-r}\beta_i+\sum_{i=0}^{n-1}\alpha_i\right)y_d(k)$$

on the right-hand side. Thus, one may express the control law for achieving exact tracking in terms of the relative terms Δ_{y_d} and Δ_u (and, hence, apply the difference learning scheme) if and only if $s(y_d(k))=0$. For arbitrary $y_d(k)$, the condition $s(y_d(k))=0$ is equivalent to

$$\frac{\sum_{i=0}^{n-r}\beta_i}{1+\sum_{i=0}^{n-1}\alpha_i}=1\tag{25}$$

For system (13), the condition in (25) is equivalent to the condition that system (13) has a unity DC gain, i.e., it achieves zero steady-state errors for step reference inputs. □

Note that, in this work, we consider a baseline system with an underlying feedback controller (Figure 1). In practice, tracking step reference inputs is a common requirement for controller designs, and this can be often achieved with well-established classical controller design techniques (Franklin et al., 1994). As we demonstrate in Section 6.5, when the baseline system is able to track step reference inputs with sufficiently small errors, the difference learning scheme can significantly reduce the amount of data required for training the DNN module.

In the following discussion, we prove the same result for the MIMO state space formulation for the special case of a position/velocity-like system.

Definition 4.2 (Position/velocity-like system). *System (1) is called position/velocity-like if it has the following properties: (i) the output of the system y is the first m elements of the state vector (i.e., $x_1,\ldots,x_m$); and (ii) for step* *reference inputs, the remaining elements of the state vector (i.e., $x_{m+1}, \dots, x_n$) are zero at steady state.*

Examples of position/velocity-like systems include, but are not limited to, mechanical systems with a position-velocity state space (e.g., industrial manipulators). Similar to the SISO transfer function scenario, we identify a necessary condition that allows us to express the control law in (6) in relative terms $\Delta_x(k) = x(k) - [y_d(k)^{\mathrm{T}}\, 0 \cdots 0]^{\mathrm{T}}$, $\Delta_{y_d}(k+r) = y_d(k+r) - y_d(k)$, and $\Delta_u(k) = u(k) - y_d(k)$. In particular, we prove the following lemma.

Lemma 4.3 (Difference learning for MIMO linear systems). *Consider a position/velocity-like MIMO system and the DNN-enhancement control architecture in Figure 1. A DNN design based on the state space approach (Insight 4.2) and the difference learning scheme is able to achieve exact tracking only if the baseline system has zero steady-state errors for step reference inputs.*

Proof. Suppose by the way of contradiction that the DNN-based approach achieves exact tracking for arbitrary feasible trajectories and the baseline system does not achieve zero steady-state error for an arbitrary step reference input $u(k) = a$, where $a \in \mathbb{R}^m$ is a constant vector. Hence, we have

$$y_{ss} = K u_{ss} = K a \tag{26}$$

where $K \in \mathbb{R}^{m \times m}$ is a constant non-zero matrix characterizing the DC gains of the system, and $K \neq I_m$ by assumption, where I_m denotes the identity matrix. Note that, when K is a zero matrix, the system has zero DC gain; it can be easily shown that the mapping $\{\Delta_x(k), \Delta_{y_d}(k+r)\} \rightarrow \{\Delta_u\}$ is one-to-many and cannot be represented by the DNN module (Jordan and Rumelhart, 1992). Next, for the case where K is non-zero, by assumption, the DNN module is able to achieve exact tracking $y_{ss} = y_d(k)$ for an arbitrary step input $y_d(k) = b$, where $b \in \mathbb{R}^m$ is a constant vector. With the difference learning scheme, the inputs to the DNN module are Δ_x and Δ_{y_d} and the output is Δ_u. When exact tracking is achieved, $\Delta_x = 0$ and $\Delta_{y_d} = 0$, while $\Delta_u = c$, where $c \in \mathbb{R}^m$ is a constant corresponding to the bias of the DNN model (i.e., the output of the DNN model when the inputs are zero). At the steady state, the reference of the baseline system is $u_{ss} = b + c$. From (26), the system output at the steady state is $y_{ss} = K(b + c)$. As exact tracking is achieved by assumption, $y_{ss} = b$ and

$$Kc = (I_m - K) b \tag{27}$$

As K is non-zero and $K \neq I_m$ by assumption, Equation (27) implies that the bias of the DNN, c, is correlated with the step input vector b. For a typical feedforward DNN, the bias c is a fixed vector determined from the training algorithm. The dependency of the bias c on the system desired output $y_d(k) = b$ leads to a contradiction. Thus, a DNN module trained with the difference learning scheme cannot achieve exact tracking for a baseline system for which the steady-state error for step reference inputs is not zero. □

Note that, in the previous discussion, the input and output of the DNN module are taken relative to the current desired output $y_d(k)$. In practice, the input and output of the DNN module can be alternatively taken relative to the current actual output $y(k)$ to additionally compensate for initial tracking errors or disturbances.

Based on the above theoretical results for linear systems, we present the following important insight.

Insight 4.4 (Necessary condition for applying the difference learning scheme). *In order to reduce the amount of training data, a difference learning scheme can be applied to the input and output selection of the DNN module. However, as shown in the theoretical analysis for the linear system formulations, for the DNN approach with the difference learning scheme to be effective, the baseline system controller needs to be designed such that the system response has zero or sufficiently small steady-state errors for step reference inputs.*

This insight is motivated from the linear system formulations. As nonlinear systems can be approximated by a set of piecewise linear/affine systems with arbitrary accuracy (Helwa and Caines, 2015), it is reasonable to expect that the necessary condition is also required for the nonlinear system (2). In Section 6, we verify this necessary condition for nonlinear systems with quadrotor experiments.

5. Simulation

In this section, we illustrate Insights 4.1–4.3 by considering two linear MIMO baseline closed-loop systems. The two systems have the same state equation:

$$x(k+1) = \begin{bmatrix} 0.2 & 1 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.6 \end{bmatrix} x(k) + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0.5 \end{bmatrix} u(k) \tag{28}$$

The output equations of the two systems are defined in (29) and (30), respectively:

$$y(k) = \begin{bmatrix} 0.35 & 0.35 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} x(k) \tag{29}$$

$$y(k) = \begin{bmatrix} -0.35 & 0.35 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} x(k) \tag{30}$$

Note that the two systems we consider have identical dynamics and differ only in the output equations and, hence, the locations of zeros.[1] In particular, system (29) has a stable (minimum phase) zero at -0.8, while system (30) has an unstable (non-minimum phase) zero at 1.2. Both systems have three stable poles at $\{0.2, 0.5, 0.6\}$ and a vector relative degree of $(1, 1)$.

Upon introducing the DNN architecture and training in Section 5.1, we first follow the state space approach to select the input of the DNN module and show the necessity of stability of the baseline system zero dynamics (Insights

4.1 and 4.2) in Section 5.2. After verifying the first two insights, we illustrate in Section 5.3 the efficacy of the alternative DNN input selection derived from the transfer function formulation (Insight 4.3). Note that, for this simulation study and with known system matrices (A, B, C), we can compute (6) and use it as the ground truth to assess the proposed DNN approach.

5.1. DNN architecture and training

For comparison purposes, the DNN architecture and training trajectories are identical for all simulation cases presented in this section. Matlab's Neural Network Toolbox is used for implementing the DNN modules. The DNNs are a fully connected feedforward networks with two hidden layers; each hidden layer consists of 20 hyperbolic tangent activation units. The training trajectories are 25 sinusoidal trajectories with different combinations of amplitudes and frequencies; the amplitudes range between 1 and 5, and the frequencies range between 0.024 and 1.25 Hz. Note that the architecture of the DNN modules (i.e., the number of hidden layers and the number of neurons) is chosen such that the DNNs have sufficient modeling complexity. In this study, we set aside part of the training data as the validation set and use a standard validation procedure to ensure that the DNN modules do not overfit or underfit the training data (Bishop, 2006). We further test the sufficiency of the training data by running the DNN-enhanced system on untrained trajectories. In general, the DNN architecture and the training trajectories are not restricted to the particular choices we made, but one should validate the trained DNN module for its generalizability.

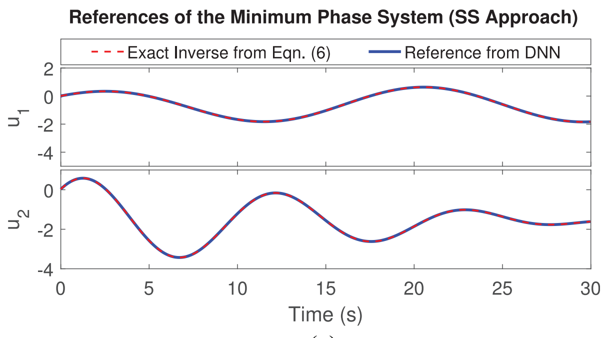
As shown in Figure 1, the baseline systems we consider are feedback systems with reference input u and output y. Our goal is to use a DNN module to enhance the tracking performance of a baseline system by adjusting the reference input u sent to the baseline system. In the training phase, the responses of the baseline systems ($x(k)$, $y(k)$, $u(k)$) are recorded at 70 Hz for constructing the training datasets, which consist of labeled input–output pairs ($\mathcal{I}$, $\mathcal{O}$). The DNN input $\mathcal{I}$ and output $\mathcal{O}$ are defined for each simulation case as follows. In the first set of simulations, the state space approach (Insight 4.2) is examined. For both system (29) and system (30), the input and output of the DNN module are selected as $\mathcal{I}_{ss} = \{x(k), y_{d,1}(k+1), y_{d,2}(k+1)\}$ and $\mathcal{O} = \{u(k)\}$, where $y_{d,i}$ denotes the ith element of y_d. In the second set of simulations, we focus on the minimum phase system (29). Based on the transfer function approach (Insight 4.3), the input and output of the DNN module are $\mathcal{I}_{tf} = \{y_{d,1}(k-2:k+1), y_{d,2}(k-2:k+1), u(k-2:k-1)\}$ and $\mathcal{O} = \{u(k)\}$, where ":" abbreviates consecutive discrete-time indexes. Note that, in the construction of the training dataset, the data pairs ($\mathcal{I}$, $\mathcal{O}$) are randomly sampled from the 25 training trajectories with balanced proportions to prevent the model from overfitting a particular frequency.

The Levenberg–Marquardt algorithm is used for training the weight and bias parameters of the DNN module. In the first set of simulations, the training objective is to minimize the mean squared error between the targets $\mathcal{O}$ and the DNN outputs. For the second set of simulations, we additionally include an L_2 regularization term in the training objective function to help the training algorithm to phase out any unnecessary dimensions in the DNN input $\mathcal{I}_{tf}$; the regularization constant is set to 0.005. In the training of each DNN module, 70% of the data is used for optimizing the model parameters and the rest is used for model validations. The generalizability of the DNN modules is further verified by testing the tracking performance of the overall DNN-enhanced system on test trajectories that differ from the training trajectories.

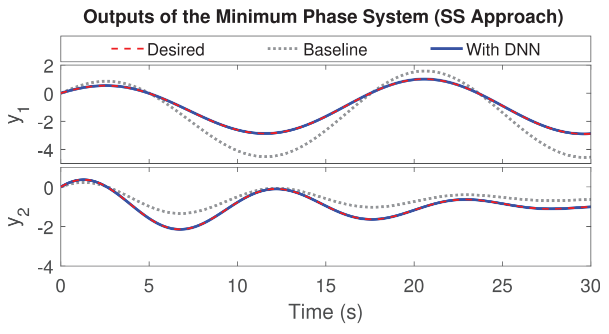
5.2. Simulation 1: Illustrations of underlying function and necessary condition

In this subsection, we illustrate Insights 4.1 and 4.2 by using the state space approach and comparing the DNN-enhanced performance of the minimum phase system (29) and the non-minimum phase system (30). For this simulation illustration, the systems' performances are compared on a test trajectory that differs from those in training: $y_{d,1}(t) = \sin\left(\frac{4\pi}{33}t\right) + \cos\left(\frac{4\pi}{41}t\right) - 1$ and $y_{d,2}(t) = \sin\left(\frac{4\pi}{23}t\right) + \cos\left(\frac{4\pi}{21}t\right) - 1$.

The references and outputs of the DNN-enhanced tracking for system (29) and system (30) are shown in Figures 2 and 3, respectively. It can be seen from Figure 2(a) that, by selecting the DNN input as $\mathcal{I} = \{x(k), y_d(k+r)\}$, the DNN is able to effectively generalize the training data collected from the minimum phase system (29), and outputs references (blue solid line) that coincide with the reference computed based on the exact inverse in (6) (red dashed line). With (6) as the ground truth, the root-mean-square (RMS) modeling error of the DNN module is approximately 7.8×10^{-5}. From Figure 2(b), we see that the reference computed by the DNN module compensates for the magnitude errors in the baseline system response (gray dotted line), and leads to approximately exact tracking (blue solid line and red dashed line). On this particular test trajectory, the addition of the DNN module reduces the RMS tracking error from approximately 1.0 to approximately 2.5×10^{-5}. In this simulated setting, the performance of the proposed DNN approach is only limited by the modeling accuracies and numerical precisions. In contrast to the minimum phase case, the reference for achieving exact tracking is unbounded in the non-minimum phase system case (30) owing to the inherent instabilities of the system inverse dynamics (Hoagg and Bernstein, 2007). In the non-minimum phase case reflected in Figure 3, though with the same architecture and training, the DNN module cannot effectively model the exact inverse in (6) (Figure 3(a)) and leads to worse performance as compared with the baseline system (Figure 3(b)).

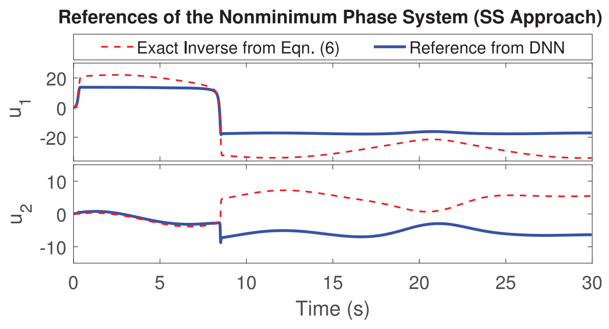


(a)

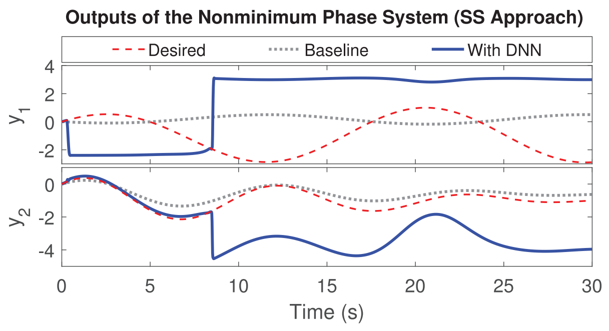


(b)

Fig. 2. The references and outputs of the minimum phase closed-loop system (29) for a desired trajectory with $y_{d,1}(t) = \sin\left(\frac{4\pi}{33}t\right) + \cos\left(\frac{4\pi}{41}t\right) - 1$ and $y_{d,2}(t) = \sin\left(\frac{4\pi}{23}t\right) + \cos\left(\frac{4\pi}{21}t\right) - 1$. (a) References u of the minimum phase system (29) with the state space approach (Insight 4.2). The root-mean-square (RMS) modeling error of the DNN module is approximately 7.8×10^{-5}. (b) Outputs y of the minimum phase system (29). The RMS tracking errors of the baseline system and the DNN-enhanced system are approximately 1.0 and 2.5×10^{-5}, respectively. From (a), the DNN module design based on Insight 4.2 is able to approximate the system's exact inverse equation (6) with high accuracy; from (b), the reference computed by the DNN module is able to compensate for the errors in the baseline system response and approximately achieve exact tracking.



(a)



(b)

Fig. 3. The references and outputs of the non-minimum phase closed-loop system (30) for the desired trajectory shown in Figure 2. (a) References u of the non-minimum phase system (30) with the state space approach (Insight 4.2). The RMS modeling error of the DNN module is approximately 14.5. (b) Outputs y of the non-minimum phase system (30). The RMS tracking errors of the baseline system and the DNN-enhanced system are approximately 2.0 and 4.6, respectively. Owing to the inherent instability of the non-minimum phase system, the reference u for achieving exact tracking is unbounded (Hoagg and Bernstein, 2007). From (a), the DNN module consequently cannot effectively model the exact inverse of system (30); from (b), when the necessary condition of achieving stable zero dynamics in Insight 4.1 is violated, the DNN inverse learning approach cannot be applied directly to enhance the tracking performance of the baseline system.

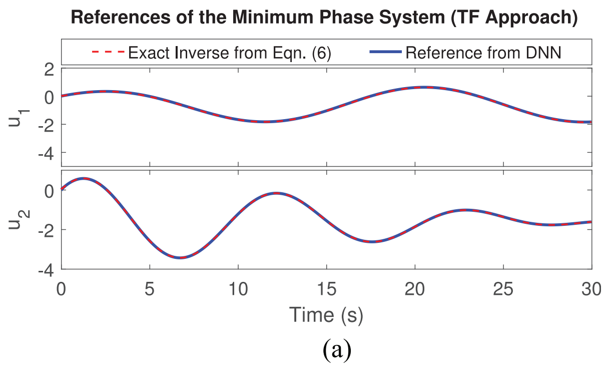
5.3. Simulation 2: Illustrations of the transfer function approach

In the previous subsection, we showed the effectiveness of the state space approach for designing the DNN module to enhance the tracking performance of the minimum phase system (29). In this subsection, we provide a brief discussion on a DNN design based on the equivalent transfer function formulation (Insight 4.3).

Figure 4 shows the references and outputs of the system (29) with the DNN design based on Insight 4.3. From Figure 4(a), we can see that similar to the state space approach, the DNN module design based on the transfer function approach (blue solid line) is able to approximate the reference from the exact inverse equation (6) (red dashed line). For this particular test trajectory, the RMS modeling error of the DNN is approximately 1.2×10^{-2}. Consequently, as shown in Figure 4(b), the output of the DNN-enhanced system (blue solid line) also coincides with the desired trajectory (red dashed line). The RMS tracking error of the DNN-enhanced system is approximately 4.2×10^{-3}. This simulation example shows that the transfer function approach can be equivalently used to enhance the tracking performance of the minimum phase system (29) without relying on the knowledge or estimation of the full state as required by the state space approach.

6. Quadrotor experiments

This section presents the results of quadrotor experiments designed to verify the theoretical insights derived in Section 4. In order to test the effectiveness of the DNN



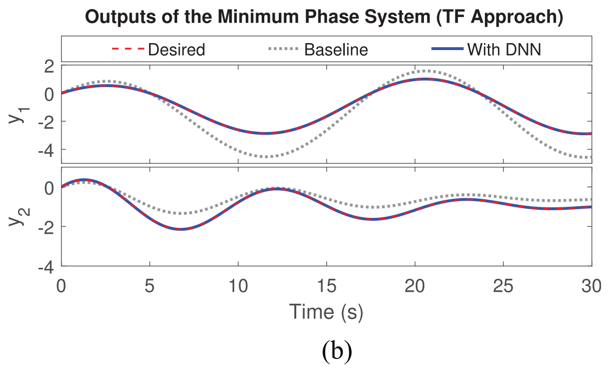


Fig. 4. The references and outputs of the minimum phase closed-loop system (29) for the desired trajectory shown in Figure 2. (a) References u of the minimum phase system (29) with the transfer function approach (Insight 4.3). The RMS modeling error of the DNN module is approximately 1.2×10^{-2}. (b) Outputs y of the minimum phase system (29). The RMS tracking errors of the baseline system and the DNN-enhanced system are approximately 1.0 and 4.2×10^{-3}, respectively. From (a), the DNN module design based on the transfer function approach (Insight 4.3) is an equivalent approximation of the exact inverse equation (6); from (b), as with the state space approach (Figure 2(b)), exact tracking is approximately achieved with the DNN module design based on the alternative transfer function formulation.

module design based on the provided theoretical insights, we adopt the fly-as-you-draw application setup from Li et al. (2017), where visitors are invited to draw desired trajectories on a mobile device, and the desired trajectories are tracked by a quadrotor vehicle. A demonstration video of the experiments is available as Extension 1, and the hand drawings used for evaluating the proposed DNN-enhancement trajectory tracking approach are shown in Figure 5.

In the following discussion, we first introduce the experimental setup, the control architecture, and the DNN architecture and training procedures in Section 6.1. In Section 6.2, we verify the proposed DNN input–output design and demonstrate the generalizability of the DNN module on the same 30 test trajectories. Upon verifying the proposed DNN input–output design, in Section 6.4 we show that the performance of the proposed approach can be pushed further by improving the representativeness of the DNN training dataset. This section is concluded with illustrations of the improved training data efficiency of the difference-learning scheme in Section 6.5. The training data and testing results presented in the following subsections can be found in Extension 2.

Note that, for the convenience of the discussion, we denote the desired trajectory with a subscript d, the references of the baseline system with a subscript r, and the measured states of the quadrotor with a subscript a.

6.1. Experimental setup

6.1.1. Overview. The objective of the experiments is to design a control system such that the center of mass of a quadrotor vehicle $\mathbf{p}_a(k)$ tracks desired trajectories $\mathbf{p}_d(k)$ generated based on arbitrary hand-drawings with high accuracy from the first attempt. In the experiments, we use Parrot AR.Drone 2.0 as the testing platform and implement the control algorithm in the Robot Operating System (ROS) environment.

6.1.2. Desired hand-drawn test trajectories. The desired trajectories to be tracked by the quadrotor are generated with the fly-as-you-draw application (Li et al., 2017). In particular, in order to generate the desired trajectories, we invite visitors to draw on a mobile device, which gives us sets of discrete points sampled at fixed time intervals along the hand-drawings. The distance between two consecutive

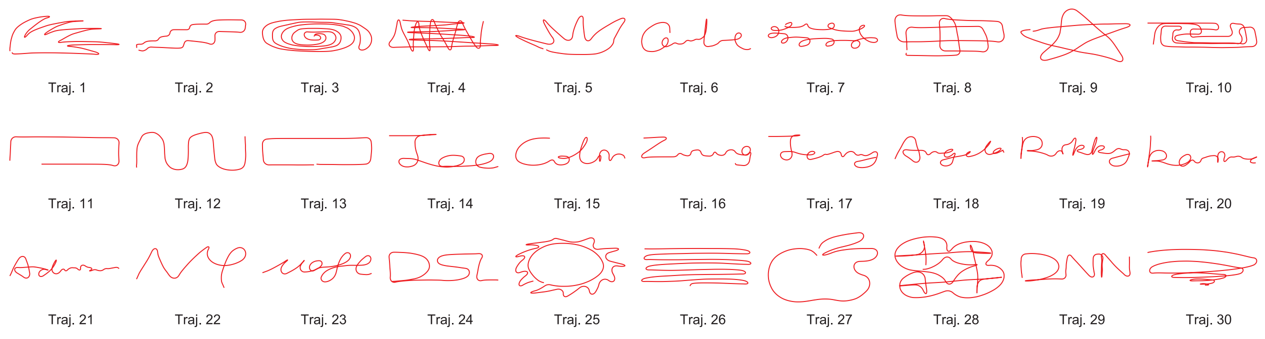


Fig. 5. Illustrations of 30 hand-drawn trajectories for testing the DNN-enhancement approach (Li et al., 2017).

points along a hand-drawing is proportional to the drawing speed. Given the set of sampled points from a hand-drawing, the desired position trajectory for the quadrotor is then interpolated using the sampling interval of the position controller. The speed along the desired trajectory is scaled based on a predefined maximum speed $v_{\max}$ and a predefined maximum acceleration $a_{\max}$, which are defined such that the generated trajectory is feasible for the quadrotor to track.

Given a desired trajectory generated from a hand-drawing, we use the three-dimensional RMS position tracking error as the measure for evaluating the tracking performance of the quadrotor:

$$e_{\text{traj}} = \left(\frac{1}{N}\sum_{k=1}^{N} ||\mathbf{p}_d(k) - \mathbf{p}_a(k)||^2\right)^{\frac{1}{2}} \tag{31}$$

where N is the number of time steps for which the trajectory is defined.

6.1.3. Control architecture. The quadrotor vehicle has 12 states: translational positions $\mathbf{p} = (x, y, z)$, translational velocities $\mathbf{v} = (\dot{x}, \dot{y}, \dot{z})$, attitudes $\boldsymbol{\theta} = (\phi, \theta, \psi)$, and rotational velocities $\boldsymbol{w} = (p, q, r)$. The baseline controller of the quadrotor vehicle consists of (i) an off-board position controller that receives the reference positions and velocities ($\mathbf{p}_r$ and $\mathbf{v}_r$) and outputs the desired roll angle, pitch angle, yaw rate, and z-velocity commands ($\phi_{\text{cmd}}, \theta_{\text{cmd}}, r_{\text{cmd}},$ and $\dot{z}_{\text{cmd}}$) at 70 Hz; and (ii) an on-board attitude controller that adjusts motor thrusts based on the roll angle, pitch angle, yaw rate, and z-velocity commands at 200 Hz. Of particular interest is the off-board position controller, which consists of a nonlinear transformation and PD control; from the internal model principle, it is known that this type of controller cannot be tuned to achieve perfect tracking for arbitrary desired reference frequencies (Francis and Wonham, 1976). We aim to enhance the baseline position controller with our proposed DNN add-on module, which models the output of the inverse dynamics of the baseline control system. In the experiments, we introduce a DNN module design based on Insight 4.2 to adjust the position reference $\mathbf{p}_r$ and the velocity reference $\mathbf{v}_r$ sent to the baseline controller, and compare the tracking performance of the DNN-enhanced controller against that of the baseline controller. As in previous work (Li et al., 2017) and for the robustness of implementation against instability, the DNN-loop in the experiments runs at 7 Hz, which is 10 times slower than the baseline controller.

In the implementation of our baseline position controller and the DNN module, the states of the quadrotor are estimated based on a *Vicon* motion capture system running at 200 Hz. The onboard attitude controller of the ARDrone relies only on onboard sensing (Bristeau et al., 2011), and the onboard attitude estimation and control modules together are a black box for our baseline position controller and DNN module implementation.

6.1.4. Neural network architecture and training. For comparison purposes, the DNN modules used in the experiments have the same architecture and training procedure as in Li et al. (2017). The DNN modules are composed of fully connected FNNs with four hidden layers of 128 ReLU neurons; the python TensorFlow library is used for implementing the DNN module. The training dataset is a set of labeled input–output pairs constructed from the input–output response data of the baseline system on one or more multiple training trajectories (details of the DNN module input–output selections are discussed in Section 6.2). The sampling rate of the baseline system for constructing the training dataset is consistent with the update rate of the DNN module, which is selected to be 7 Hz. Of all the training data collected from the baseline system, 90% is selected for training and the remaining is used for validation. The training loss function is the squared error between the DNN output and the labeled output in the training dataset. The Adam optimizer (Kingma and Ba, 2014) is used for optimizing the weight parameters of the DNN. A dropout rate of 0.5 is used to improve the generalizability of the DNN to unseen inputs (Srivastava et al., 2014).

6.2. Experiment 1: DNN input–output design

Through experimental trial-and-error, Li et al. (2017) found that a DNN module with the following input and output can effectively improve the performance of the baseline system for tracking arbitrary hand-drawn trajectories:

$$\begin{aligned}\mathcal{I}_1 = \{&\mathbf{p}_d(k+4) - \mathbf{p}_a(k),\ \mathbf{p}_d(k+6) - \mathbf{p}_a(k),\\ &\mathbf{v}_a(k),\ \mathbf{v}_d(k+4),\ \mathbf{v}_d(k+6),\ \boldsymbol{\theta}_a(k),\\ &\boldsymbol{\theta}_d(k+4),\ \boldsymbol{\theta}_d(k+6),\ \boldsymbol{w}_a(k),\ \boldsymbol{w}_d(k+4),\\ &\boldsymbol{w}_d(k+6),\ \ddot{z}_a(k),\ \ddot{z}_d(k+4),\ \ddot{z}_d(k+6)\}\end{aligned} \tag{32}$$

$$\mathcal{O}_1 = \{\mathbf{p}_r(k) - \mathbf{p}_d(k),\ \mathbf{v}_r(k) - \mathbf{v}_d(k)\} \tag{33}$$

On the 30 hand-drawn trajectories shown in Figure 5, the average RMS error reduction achieved by the DNN module is approximately 43%. In order to verify Insight 4.2, we repeat the 30 test trajectories from Li et al. (2017) with a DNN module design based on the proposed input–output selection and compare the improved tracking performance with that achieved in Li et al. (2017). Note that we repeated the experiments in Li et al. (2017) on the quadrotors used for this work for comparability.

In order to apply our insights, we first performed simple step response experiments and identified the following properties of the baseline system:

(P1) the responses of the baseline system are approximately decoupled in the x-, y-, and z-direction;

(P2) the relative degrees of the baseline system in the x-, y-, and z- direction are four, four, and three, respectively; and

(P3) zero steady-state error for step reference inputs is approximately achieved in the three directions.

Note that, for obtaining (P2), we identified the relative degrees of the baseline system from experimental data because the baseline system is treated as a black box in our experiment. In practice, one could alternatively derive the relative degrees of a baseline system from a dynamics model; however, the relative degrees determined based on a standard model may differ from that identified experientially due to unmodeled dynamics and/or delays that are present in the physical system.

Given properties (P1)–(P3), we assume decoupled dynamics in the x-, y-, and z-direction and apply Insight 4.2 with the difference learning scheme to obtain the following input and output selection of the DNN module:

$$\begin{aligned}\mathcal{I}_2 = \{&x_d(k+4) - x_a(k), y_d(k+4) - y_a(k), z_d(k+3) \\ &- z_a(k), \dot{x}_d(k+3) - \dot{x}_a(k), \dot{y}_d(k+3) - \dot{y}_a(k), \\ &\dot{z}_d(k+2) - \dot{z}_a(k), \boldsymbol{\theta}_a(k), \boldsymbol{w}_a(k)\}\end{aligned} \tag{34}$$

$$\mathcal{O}_2 = \{\mathbf{p}_r(k) - \mathbf{p}_a(k), \mathbf{v}_r(k) - \mathbf{v}_a(k)\} \tag{35}$$

We note two differences between the DNN from Li et al. (2017) and the proposed DNN design based on Insight 4.2. The first is the DNN input selection. In comparison with the DNN from Li et al. (2017), which has 36 inputs ($\#\mathcal{I}_1 = 36$, where $\#$ denotes cardinality), the DNN design based on Insight 4.2 has only 12 inputs ($\#\mathcal{I}_2 = 12$). Based on the inverse-dynamics formulation, the input selection $\mathcal{I}_2$ represents the necessary and sufficient inputs that allow the DNN add-on module to achieve enhanced tracking performance. Another difference is in the application of the difference learning scheme for the two DNN designs. In particular, for the DNN design from Li et al. (2017), the position elements in the input $\mathcal{I}_1$ are taken relative to the actual output values (subscripted with a), and the position elements in the output $\mathcal{O}_1$ are taken relative to the desired output values (subscripted with d). For the proposed DNN design based on Insight 4.2, the relative terms in the input $\mathcal{I}_2$ and output $\mathcal{O}_2$ are consistently taken with respect to the actual values (subscripted with a). Based on the theoretical discussions of Insight 4.4, we expect the consistency of the relative terms in the proposed design would further improve the capability of the DNN module in correcting for any deviations from the desired trajectories.

We first present the performance comparison between the DNN from Li et al. (2017) and the proposed DNN on one of the test trajectories (Figures 6, 7, and 8). We then summarize the comparison between the two DNN designs on 30 hand-drawn test trajectories (Figure 9). The test trajectories are generated based on the procedure described in Section 6.1.2. The maximum speed and maximum acceleration of the trajectories are $v_{\max} = 0.6$ m/s and $a_{\max} = 2.0$ m/s^2, respectively. Note that, for the experiments, the DNN modules are trained on a 400-second three-dimensional sinusoidal trajectory similar to that used in Li et al. (2017) (Extension 2a). In order to establish a fair comparison, we use the same DNN architecture, training data, and training algorithm for the DNN design based on Li et al. (2017) and the DNN design based on Insight 4.2; the only difference between the two DNNs is the input–output selection.

From Figures 11 and 12, it can be seen that both the DNN from Li et al. (2017) (green solid line) and the DNN design based on Insight 4.2 (blue solid line) are able to reduce the time delays and magnitude errors of the baseline system tracking response (gray dotted line) and lead to quadrotor tracking paths that are closer to the desired hand-drawing (red dashed line). On this test trajectory, the RMS tracking error reduction achieved by the DNN from Li et al. (2017) and the proposed DNN are 45% and 67%, respectively. The trajectory tracking error comparison depicted in Figure 8 shows that the proposed DNN design based on Insight 4.2 achieves similar error reductions to the DNN design from Li et al. (2017) while having far fewer inputs.

Figure 9 (Extension 2b) summarizes the performance comparison between the two DNN modules on the 30 hand-drawn test trajectories studied in Li et al. (2017) (see Figure 5). The plot shows that the proposed DNN module design based on Insight 4.2 (blue bars) leads to similar tracking performance as the DNN module from Li et al. (2017) (green bars). On the 30 test trajectories, the mean RMS error of the baseline system enhanced by the DNN from Li et al. (2017) is approximately 0.17 m, and that of the baseline system enhanced by the proposed DNN is approximately 0.15 m. The corresponding average tracking error reduction achieved by the DNN from Li et al. (2017) and that design based on Insight 4.2 are 49% and 54%, respectively.

From this set of experiments, we verify Insight 4.2 on the proposed DNN input selection. Although the input dimension is reduced by two thirds as compared with the DNN from Li et al. (2017), the DNN module design based on the derived theoretical insight can effectively enhance the quadrotor's baseline system performance. The comparison with the results from Li et al. (2017) further validates the generalizability of the proposed DNN for tracking arbitrary untrained trajectories impromptu.

6.3. Experiment 2: Generalization to different trajectory speeds

In this subsection, we examine the performance of the baseline system and the DNN-enhanced systems for different operating speeds. In particular, we use the trajectory shown in Figure 6 as the test trajectory and scale the time-

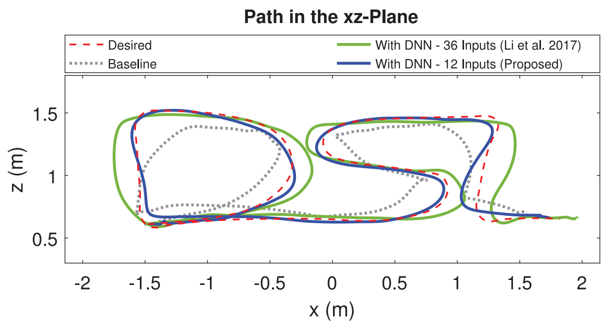


Fig. 6. A comparison of the tracking performance enhancements between the DNN module from Li et al. (2017) and the DNN module design based on Insight 4.2 for a hand-drawn test trajectory (trajectory 24 in Figure 5). On this test trajectory, the RMS tracking error of the baseline system is approximately 0.41 m. The RMS tracking errors of the baseline system enhanced by the DNN module from Li et al. (2017) and the proposed DNN module design based on Insight 4.2 are 0.23 and 0.14 m, respectively, which correspond to 45% and 67% error reductions.

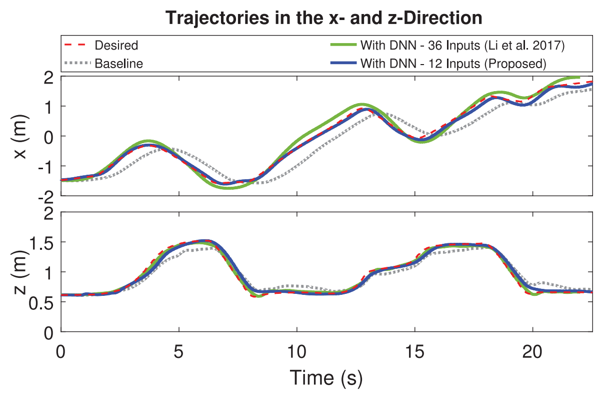


Fig. 7. A comparison of the *x*- and *z*-position trajectories for a hand-drawn test trajectory (corresponding to Figure 6). From the plots, the DNN module from Li et al. (2017) and the DNN module trained based on Insight 4.2 both tend to correct the delays and magnitude errors of the baseline system response. When compared with the DNN from Li et al. (2017), the proposed DNN design based on Insight 4.2 has two thirds fewer inputs while achieving better performance enhancements.

parameterized trajectory based on a set of specified maximum speeds $v_{\max} = \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ m/s. Figure 10 (Extension 2c) summarizes the performance of (i) the baseline system, (ii) the system enhanced with the DNN module design from Li et al. (2017), and (iii) the system enhanced with the DNN module design based on our Insight 4.2. As can be seen from the plot, the tracking performance of the baseline system (gray) degrades quickly as the speed of the test trajectory increases. In contrast, the tracking errors of the systems with the DNN modules (green and blue) remain relatively at a lower constant level. The average RMS tracking error over the trials presented in Figure 10 is 0.35 m for the baseline system, 0.19 m for the system enhanced with the DNN design from Li et al. (2017), and 0.14 m for the system enhanced with the DNN design based on Insight 4.2.

As discussed in Section 4.2, the DNN module in our framework represents the inverse of the baseline system and, theoretically, establishes an identity mapping from the desired output y_d to the actual output of the system y. In the quadrotor experiments, we demonstrate the efficacy of the proposed DNN module approach for reducing the tracking error of the baseline system across multiple operating speeds. We note that the effectiveness of the DNN module generally relies on having training data that sufficiently covers the range of operating speeds of interest. As discussed in Section 7, we would like to explore systematic approaches for training data generation as future work.

6.4. Experiment 3: DNN training dataset

In the previous set of experiments, the performance of the DNN from Li et al. (2017) and the proposed DNN are compared on the basis of training on 400-second sinusoidal trajectories. These trajectories have gradually increasing amplitudes but fixed frequencies in the *x*-, *y*-, and *z*-direction (Li et al., 2017). In this subsection, we show that the performance enhancement achieved by the proposed DNN design can be further improved with a richer training dataset. In particular, we compare two training datasets constructed from the baseline system responses to different training trajectories (Extension 2a):

- Training Dataset 1 based on a 400-second sinusoidal training trajectory; and
- Training Dataset 2 based on the 400-second sinusoidal training trajectory from Training Dataset 1 and 30 additional hand-drawn trajectories.

Note that the 30 hand-drawn trajectories in Training Dataset 2 are different from the 30 trajectories (Figure 5) for evaluating the performance of the DNNs. By adding hand-drawn trajectories to the DNN training, we expect to increase the similarity between the DNN inputs encountered at the training time and the test time. In particular, we expect the arbitrary training hand-drawn trajectories to capture a richer set of cases (e.g., sharp edges) that are non-trivial to define with analytical expressions. By having a more representative training dataset, we can then further reduce the generalization error of the DNN module for impromptu tracking performance enhancements.

Figure 11 (Extension 2b) shows the performance comparison of three DNN-enhanced systems on the 30 test hand-drawn trajectories (Figure 5). From the previous subsection, we show that, on average, the DNN with the proposed inputs (middle histogram in Figure 11) leads to better performance as compared with the DNN from Li et al. (2017) (top histogram in Figure 11). When comparing the proposed DNN trained with Training Dataset 1

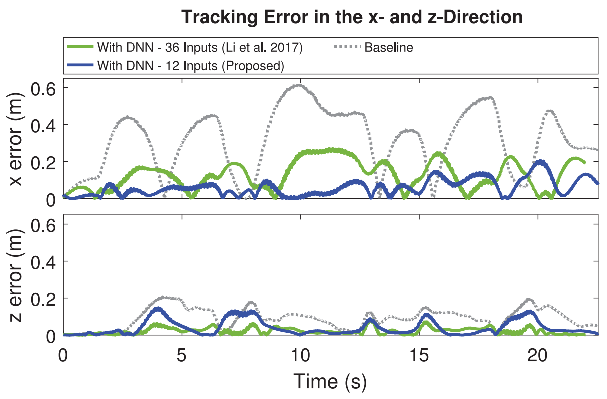


Fig. 8. The tracking errors of the x- and z-position ($|x(k) - x_d(k)|$ and $|z(k) - z_d(k)|$) corresponding to Figure 7. The DNN module from Li et al. (2017) and the DNN module trained based on Insight 4.2 both effectively reduce the peak tracking errors of the baseline system. The peak errors for the baseline system in the x- and z-direction are approximately 0.62 and 0.21 m, respectively. For the DNN design from Li et al. (2017), the peak tracking errors in the x- and z-direction are reduced to approximately 0.27 and 0.09 m, while for the proposed DNN, the peak tracking errors in the x- and z-direction are reduced to approximately 0.21 and 0.15 m.

(middle histogram in Figure 11) and Training Dataset 2 (bottom histogram in Figure 11), we see that the inclusion of the additional hand-drawn trajectories in training further improves the performance of the DNN-enhanced system in tracking arbitrary hand-drawn trajectories. Overall, the proposed DNN trained with Training Dataset 2 increases the average RMS tracking error reduction by 8% as compared with the proposed DNN trained with Training Dataset 1.

We note that, as similarly discussed in Mueller et al. (2012), the learning performance of the DNN approach is limited by the non-repeatable or stochastic error in the baseline system. More explicitly, the non-repeatable or stochastic error corresponds to the variations we see in the baseline system output when an identical reference is given to the baseline system multiple times. In our experimental setup, one primary source of the stochastic error is the noise in the onboard inertial measurement unit (IMU)- and camera-based attitude estimation and control, which we do not have direct access to. In our experiments, the proposed DNN module trained with Training Dataset 2 reduces the average tracking error of the quadrotor on the 30 hand-drawn trajectories to approximately 0.07–0.15 m. This performance is comparable with the standard deviation of the position error of the quadrotor at hover, which is an estimate of the inherent noise in the system and serves as a lower bound on the achievable tracking accuracy. We expect our DNN to achieve lower tracking errors if the response of the baseline system could be made more repeatable.

6.5. Experiment 4: Difference learning

In Section 4.5, we theoretically showed that, in order to apply the difference learning scheme to improve the data efficiency of the DNN training, the baseline system needs to achieve zero steady-state error for step reference inputs. In this subsection, we first illustrate the necessity of the condition by applying the difference learning scheme to DNN modules to enhance (i) the original baseline system where zero steady-state error for step reference inputs is achieved, and (ii) a modified baseline system where the necessary condition is not achieved. In the experiment,

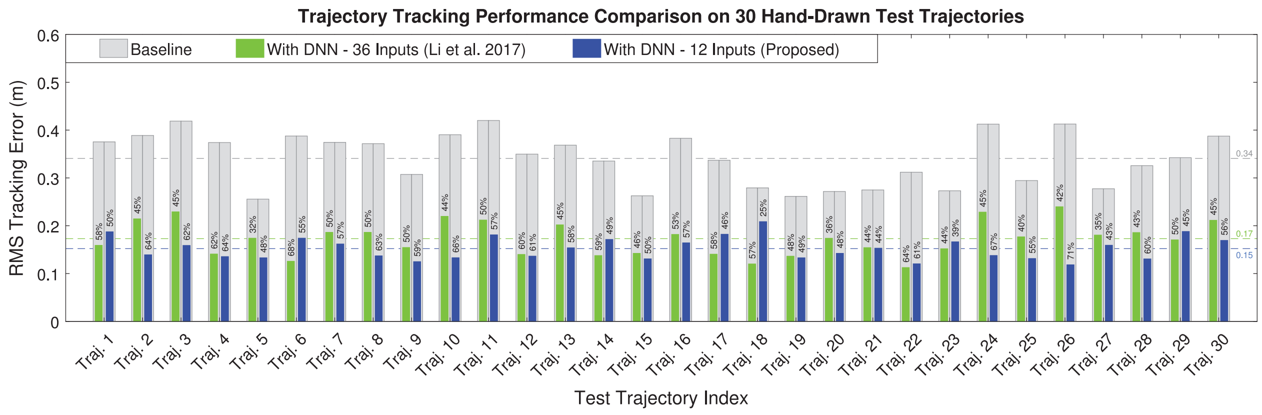


Fig. 9. Comparisons of the tracking performance enhancements between the DNN module from Li et al. (2017) (with 36 inputs) and the proposed DNN module design based on Insight 4.2 (with 12 inputs). In the two sets of experiments, the percentage of the RMS tracking error reductions achieved by the DNN module are indicated above the corresponding bars; the mean RMS error over the 30 trajectories are indicated by the horizontal dashed lines. Note that, for comparability, we repeated the 30 impromptu tracking experiments from Li et al. (2017) on the same quadrotor vehicles and training datasets used for testing the proposed DNN design. From the plot, despite having two thirds fewer inputs, the proposed DNN design based on Insight 4.2 yields a performance comparable with the DNN from Li et al. (2017). On the 30 test trajectories, the average RMS error reduction is 49% for the DNN from Li et al. (2017) and 54% for the proposed DNN design based on Insight 4.2.

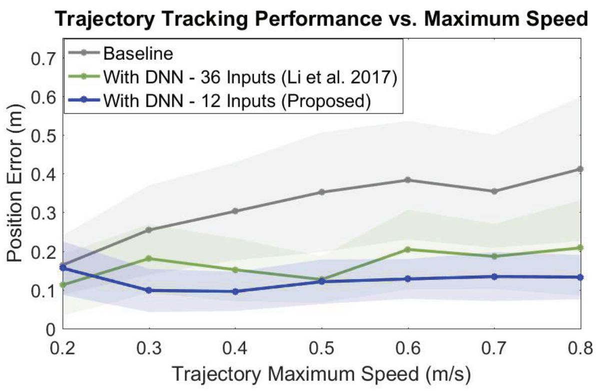


Fig. 10. The tracking performance of (i) the baseline system, (ii) the system enhanced by the DNN module design from Li et al. (2017), and (iii) the system enhanced by the DNN module design based on Insight 4.2 as the trajectory speed increases. The solid lines and the shaded regions in the plot correspond to the mean and the standard deviation of the position error $\mathbf{p}_d - \mathbf{p}_a$ along the test trajectories. In contrast, the tracking error of the baseline system increases significantly with trajectory speed, while the tracking error of the system enhanced with the proposed DNN remains at a lower constant level. The average RMS tracking error over the presented trials is 0.35 m for the baseline system, 0.19 m for the system with the DNN module design from Li et al. (2017), and 0.14 m for the system with the DNN design based on Insight 4.2.

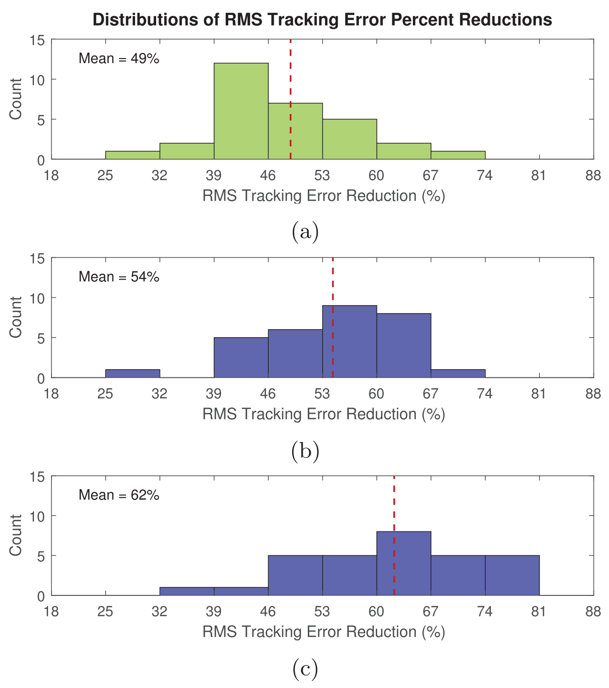


Fig. 11. A comparison of the tracking error reduction achieved by (a) the DNN used in Li et al. (2017) and trained on a 400-second sinusoidal trajectory, (b) the proposed DNN designed based on Insight 4.2 and trained on the 400-second sinusoidal trajectory, and (c) the proposed DNN designed based on Insight 4.2 and trained on the 400-second sinusoidal trajectory and 30 additional hand-drawn trajectories. Note that, for the last case (c), the 30 additional hand-drawn trajectories used for training are different from the 30 test trajectories. The mean percentage error reduction for each distribution is indicated by the vertical dashed line.

the modified baseline system is obtained by multiplying the reference signals z_r sent to the original baseline system by a factor of 0.5. The baseline and DNN-enhanced tracking performance for the two systems are shown in Figure 12 (Extension 2d). The plots show that for the original baseline system (bottom panel), where zero steady-state error for step reference inputs is achieved, the DNN with the difference learning scheme is able to effectively enhance the tracking performance of the baseline system. However, as expected from Insight 4.4, for the modified baseline system (top panel), where the zero steady-state error condition is not satisfied, the DNN trained with the difference learning scheme only partially compensates for the magnitude error and the bias of the modified baseline system.

In order to evaluate the effectiveness of the proposed difference learning scheme for improving the training data efficiency, we next compare a DNN module trained with and a DNN trained without the difference learning scheme. Figure 13 (Extension 2d) shows a comparison of the DNN modules trained with (blue) and without (red) the difference learning scheme for enhancing the tracking performance of the quadrotor baseline system where zero steady-state error for step reference inputs is achieved. In the plot, the RMS tracking errors of the DNN-enhanced systems are compared as the amount of training data varies. Note that, in order to prevent overfitting, the training datasets are randomly sampled from a large training dataset (Training Dataset 2). Here, we use Trajectory 24 (Figure 5) as the test trajectory for evaluating the performance of the DNN-enhanced systems. Figure 13 shows that, for the DNN without the difference learning scheme (red), the RMS tracking increases quickly as the amount of training data reduces. In contrast, the performance of the DNN trained with the difference learning scheme (blue) drops more gradually as the amount of training data decreases. The DNN trained with the difference learning scheme reaches the best observed performance of the DNN without the difference learning scheme (gray dotted line), with approximately 15 times less data.

7. Discussion

In Section 6, we showed that the proposed DNN approach can effectively enhance the impromptu tracking performance of classical controllers. In the proposed approach, the design of the DNN module relies only on the input, output, and state data of the baseline system, as well as basic properties of the system (e.g., the vector relative degree) that can be identified from a set of simple step response experiments. Without requiring a dynamic model

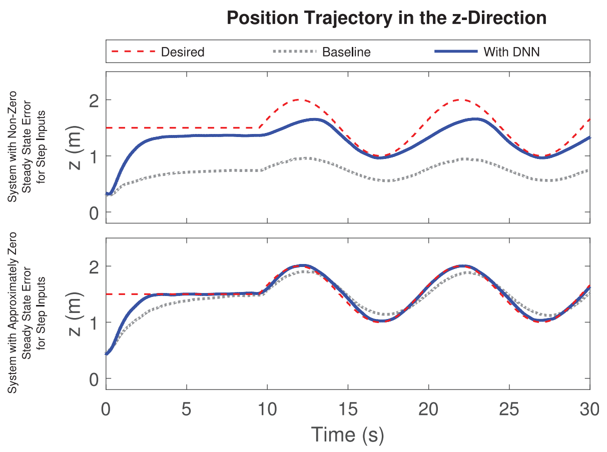


Fig. 12. A comparison of the difference learning scheme as applied on: (i) a baseline system for which zero steady-state error for step reference inputs is not achieved (top); and (ii) a baseline system for which zero steady-state error for step reference inputs is achieved (bottom). When the necessary condition of having a baseline system that achieves zero steady-state error for step reference inputs is not satisfied (see Insight 4.4), the DNN trained with the difference learning scheme cannot effectively compensate for the errors of the baseline system response.

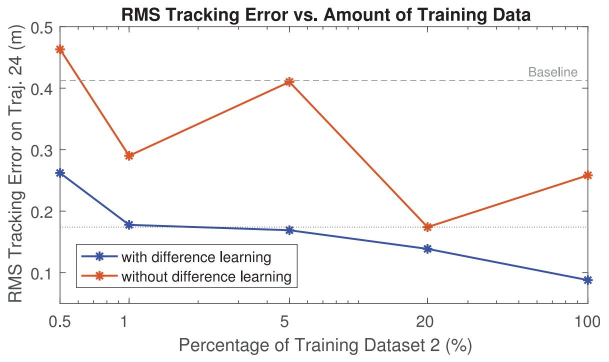


Fig. 13. A comparison of the RMS tracking error versus the amount of data for training the DNNs with (blue) and without (red) the difference learning scheme. The horizontal axis shows the proportion of randomly selected data from Training Dataset 2 described in Section 6.4; the vertical axis shows the RMS error on Trajectory 24 with the DNN-enhanced system (see Figure 5). The plot shows that the DNN trained with the difference learning scheme is able to reach the best observed performance of the DNN trained without the difference learning scheme (indicated by the gray dotted line) with approximately 15 times less training data. Note that the RMS tracking error corresponding to the baseline system is shown as a gray dashed line for reference.

as a prior, the DNN approach can be used to complement black-box control systems: a possible step towards addressing the issue of uncertain or unmodeled dynamics that limit the performance of classical model-based control design approaches.

Despite its advantages, the proposed approach has limitations that require further investigation. The first relates to its applicability to systems with hybrid dynamics, where hybrid control strategies are often applied (Antsaklis, 2000). In our formulation, we represent the baseline system by (1) and (2). For hybrid systems such as bipedal robots or quadrotors with load suspension, however, multiple dynamic equations defined on different regions of the state space must be considered. One trivial approach is to apply the current results to each individual dynamic system and train a set of independent DNN modules to enhance the tracking performance. Accounting for transitions across the boundaries of the dynamic regions is an open question. The hierarchical structure in some typical hybrid control approaches (e.g., supervisory control) naturally encourages a hierarchical learning structure for dealing with this class of systems capturing more complex dynamics.

The second limitation is related to the sufficiency and transparency of the DNN training. In particular, in our proposed approach, the training data of the DNN module needs to sufficiently cover the potential operation space of the baseline system. In the experiments, we trained the DNN modules with large datasets and verified the sufficiency of training by testing the DNN module on untrained trajectories. Although we showed that the proposed approach can significantly reduce the tracking error of the baseline system on arbitrary hand-drawn trajectories, the sufficiency of the training dataset, and hence the performance of the DNN module, are not known prior to the tests on the physical system. One promising direction for examining the data sufficiency is to introduce probabilistic learning to the current DNN-enhancement control framework. Many researchers are investigating approaches that provide uncertainty estimations to deep learning (e.g., Depeweg et al., 2018; Gal and Ghahramani, 2016). As noted by Depeweg et al. (2018), these approaches can be naturally combined with the active learning framework (MacKay, 1992) to guide training data collection. The probabilistic framework and guided data collection can potentially provide indications of insufficient training and further increase training efficiency and transparency.

8. Conclusions

This article presents theoretical and experimental studies of a DNN-based approach for enhancing the tracking performance of black-box control systems for arbitrary feasible trajectories. We considered a MIMO, possibly nonlinear, system as our starting point. In order to achieve an identity mapping from the desired output to the actual output, we established that the DNN module in the proposed control architecture should approximate the output equation of the inverse dynamics of the baseline system. Owing to the association with system inversion, the effectiveness of the proposed approach relies on two necessary conditions that the baseline system has (i) a well-defined vector relative degree and (ii) stable zero dynamics. Second, for the systems satisfying these two necessary conditions, we identified the necessary and sufficient inputs of the DNN module. Third, we verified the insights by repeating the quadrotor experiments in Li et al. (2017). In particular, we showed that with the proposed DNN input selection, the DNN input dimension is reduced by two thirds while achieving similar or better performance on the 30 hand-drawn trajectories in Li et al. (2017). Moreover, in contrast to the quadrotor baseline controller, for which the tracking error increased with the trajectory speed, the tracking errors of the DNN-enhanced systems remained small as the trajectory became more aggressive. By using a richer training dataset, we also showed that the proposed DNN module reduced RMS error by approximately 62% on the average of the 30 testing hand-drawn trajectories. Fourth, using an argument similar to the small gain theorem, we proved that, for systems with stable zero dynamics, the overall DNN-enhanced control system is input-to-state stable if the DNN modeling error is sufficiently small. Fifth, we explored via both theory and experiments the effectiveness of the difference learning scheme for improving the efficiency of the training of the DNNs in the proposed approach. In particular, we derived a necessary condition for the effectiveness of the difference learning approach, and verified this condition via experiments. For the quadrotor impromptu tracking experiments, we showed that the DNN trained with the difference learning scheme is able to achieve comparable tracking performance of a DNN module trained without the difference learning scheme with approximately 15 times less data.

Acknowledgments

We would like to thank Qiyang Li, Jingxing Qian, Zining Zhu, and Xuchan Bao for their contributions to the fly-as-you-draw application. We would also like to thank Hallie Siegel and Mario Vukosavljev for their valuable comments in the preparation of this manuscript as well as the Associate Editor and reviewers for their detailed feedback on this work.

Funding

This research was supported by the Alfred P. Sloan Foundation's Sloan Research Fellowship, the Ontario Ministry of Research and Innovation's Early Researcher Award, the Ontario Centres of Excellence's OCE/SOSCIP TalentEdge Project 27901, the Discovery Grant, Research Tools and Instruments Grant, and the Alexander Graham Bell Canada Graduate Scholarship from the Natural Sciences and Engineering Research Council of Canada (NSERC), as well as the Vector Institute for Artificial Intelligence.

Note

1. Both system (29) and system (30) are controllable and observable and are, thus, minimal state space realizations. The zeros of the MIMO systems are frequencies at which the system matrix of the MIMO systems or the equivalent transfer matrices *H(z)* of the systems drop rank (see Dahleh et al., 2004 for more details). The locations of zeros (and poles) of the systems can be conveniently veried with the Matlab command pzmap.

ORCID iDs

Siqi Zhou https://orcid.org/0000-0001-7240-546X
Mohamed K Helwa https://orcid.org/0000-0001-5101-5526
Angela P Schoellig https://orcid.org/0000-0003-4012-4668

References

Antsaklis PJ (2000) A brief introduction to the theory and applications of hybrid systems. In: *Proceedings of the IEEE Special Issue on Hybrid Systems: Theory and Applications*, pp. 879–887.

Åström K and Hägglund T (2004) Revisiting the Ziegler–Nichols step response method for PID control. *Journal of Process Control* 14(6): 635–650.

Bansal S, Akametalu AK, Jiang FJ, Laine F and Tomlin CJ (2016) Learning quadrotor dynamics using neural network for flight control. In: *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 4653–4660.

Bishop CM (2006) *Pattern Recognition and Machine Learning*. New York: Springer.

Bristeau PJ, Callou F, Vissiere D and Petit N (2011) The navigation and control technology inside the ARDrone micro UAV. *IFAC Proceedings Volumes* 44(1): 1477–1484.

Bristow DA, Tharayil M and Alleyne AG (2006) A survey of iterative learning control. *IEEE Control Systems* 26(3): 96–114.

Brogårdh T (2007) Present and future robot control development — An industrial perspective. *Annual Reviews in Control* 31(1): 69–79.

Chen FC and Khalil HK (1995) Adaptive control of a class of nonlinear discrete-time systems using neural networks. *IEEE Transactions on Automatic Control* 40(5): 791–801.

Da X, Hartley R and Grizzle JW (2017) Supervised learning for stabilizing underactuated bipedal robot locomotion, with outdoor experiments on the wave field. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3476–3483.

Dahleh M, Dahleh MA and Verghese G (2004) *Lectures on Dynamic Systems and Control*. Department of Electrical Engineering and Computer Science, Massachuasetts Institute of Technology.

Depeweg S, Hernandez-Lobato JM, Doshi-Velez F and Udluft S (2018) Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1184–1193.

Devasia S, Chen D and Paden B (1996) Nonlinear inversion-based output tracking. *IEEE Transactions on Automatic Control* 41(7): 930–942.

Drews P, Williams G, Goldfain B, Theodorou EA and Rehg JM (2017) Aggressive deep driving: Combining convolutional neural networks and model predictive control. In: *Proceedings of the Conference on Robot Learning (CoRL)*, pp. 133–142.

Francis B and Khargonekar P (1995) *Robust Control Theory* (*The IMA Volumes in Mathematics and Its Applications*). New York: Springer.

Francis BA and Wonham WM (1976) The internal model principle of control theory. *Automatica* 12(5): 457–465.

Franklin GF, Powell JD, Emami-Naeini A and Powell JD (1994) *Feedback Control of Dynamic Systems*, Vol. 3. Reading, MA: Addison-Wesley.

Gal Y and Ghahramani Z (2016) Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1050–1059.

Ge SS and Zhang J (2003) Neural-network control of nonaffine nonlinear system with zero dynamics by state and output feedback. *IEEE Transactions on Neural Networks* 14(4): 900–918.

Giesbrecht JL, Goi HK, Barfoot TD and Francis BA (2009) A vision-based robotic follower vehicle. In: *Proceedings of SPIE 7332, Unmanned Systems Technology XI*.

He W, Chen Y and Yin Z (2016) Adaptive neural network control of an uncertain robot with full-state constraints. *IEEE Transactions on Cybernetics* 46(3): 620–629.

Helwa MK and Caines PE (2015) Epsilon controllability of nonlinear systems on polytopes. In: *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 252–257.

Helwa MK, Heins A and Schoellig AP (2018) Provably robust learning-based approach for high-accuracy tracking control of Lagrangian systems. *arXiv preprint arXiv:1804.01031*.

Helwa MK and Schoellig AP (2016) On the construction of safe controllable regions for affine systems with applications to robotics. In: *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 3000–3005.

Hirschorn R (1979) Invertibility of multivariable nonlinear control systems. *IEEE Transactions on Automatic Control* 24(6): 855–865.

Hoagg JB and Bernstein DS (2007) Nonminimum-phase zeros — much to do about nothing — classical control revisited Part II. *Control Systems* 27(3): 45–57.

Hunt KJ, Sbarbaro D, żbikowski R and Gawthrop PJ (1992) Neural networks for control systems — A survey. *Automatica* 28(6): 1083–1112.

Iplikci S (2006) Support Vector Machines-based generalized predictive control. *International Journal of Robust and Nonlinear Control* 16(17): 843–862.

Isidori A (1995) *Nonlinear Control Systems*. 3rd ed. New York: Springer.

Jang TJ, Ahn HS and Choi CH (1994) Iterative learning control for discrete-time nonlinear systems. *International Journal of Systems Science* 25(7): 1179–1189.

Jiang ZP and Wang Y (2001) Input-to-state stability for discrete-time nonlinear systems. *Automatica* 37(6): 857–869.

Jordan MI and Rumelhart DE (1992) Forward models: Supervised learning with a distal teacher. *Cognitive Science* 16(3): 307–354.

Kawato M (1990) Feedback-error-learning neural network for supervised motor learning. In: *Advanced Neural Computers*. Amsterdam: Elsevier, pp. 365–372.

Kingma DP and Ba J (2014) Adam: A method for stochastic optimization. *arXiv preprint:1412.6980*.

Kiumarsi B, Lewis FL, Modares H, Karimpour A and Naghibi-Sistani MB (2014) Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics. *Automatica* 50(4): 1167–1175.

Levine S, Wagener N and Abbeel P (2015) Learning contact-rich manipulation skills with guided policy search. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 156–163.

Li Q, Qian J, Zhu Z, Bao X, Helwa MK and Schoellig AP (2017) Deep neural networks for improved, impromptu trajectory tracking of quadrotors. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5183–5189.

Liniger A, Domahidi A and Morari M (2015) Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods* 36(5): 628–647.

Liu Y and Nejat G (2013) Robotic urban search and rescue: A survey from the control perspective. *Journal of Intelligent and Robotic Systems* 72(2): 147–165.

MacKay DJ (1992) Information-based objective functions for active data selection. *Neural Computation* 4(4): 590–604.

Mueller FL, Schoellig AP and D'Andrea R (2012) Iterative learning of feed-forward corrections for high-performance tracking. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 3276–3281.

Nguyen-Tuong D and Peters J (2008) Local Gaussian Process regression for real-time model-based robot control. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 380–385.

Nguyen-Tuong D and Peters J (2010) Using model knowledge for learning inverse dynamics. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2677–2682.

Nikolic J, Burri M, Rehder J, Leutenegger S, Huerzeler C and Siegwart R (2013) A UAV system for inspection of industrial facilities. In: *Proceedings of the IEEE Aerospace Conference*, pp. 1–8.

Pane YP, Nageshrao SP and Babuška R (2016) Actor–critic reinforcement learning for tracking control in robotics. In: *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 5819–5826.

Schaal S, Atkeson CG and Vijayakumar S (2002) Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence* 17(1): 49–60.

Schoellig AP, Mueller FL and D'Andrea R (2012) Optimization-based iterative learning for precise quadrocopter trajectory tracking. *Autonomous Robots* 33(1-2): 103–127.

Slotine JJE and Li W (1987) On the adaptive control of robot manipulators. *The International Journal of Robotics Research* 6(3): 49–59.

Spong MW (1992) On the robust control of robot manipulators. *IEEE Transactions on Automatic Control* 37(11): 1782–1786.

Srivastava N, Hinton G, Krizhevsky A, Sutskever I and Salakhutdinov R (2014) Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15: 1929–1958.

Sun M and Wang D (2001) Analysis of nonlinear discrete-time systems with higher-order iterative learning control. *Dynamics and Control* 11(1): 81–96.

Suprijono H, Wahab W and Kusumoputro B (2015) Optimized direct inverse control to control altitude of a small helicopter. In: *MATEC Web of Conferences*, Vol. 34. EDP Sciences.

Sussmann H (1990) Limitations on the stabilizability of globally-minimum-phase systems. *IEEE Transactions on Automatic Control* 35(1): 117–119.

Tang S and Kumar V (2018) Autonomous flight. *Annual Review of Control, Robotics, and Autonomous Systems* 1(1): 29–52.

Tayebi A (2004) Adaptive iterative learning control for robot manipulators. *Automatica* 40(7): 1195–1203.

Yan Z and Wang J (2014) Robust model predictive control of nonlinear systems with unmodeled dynamics and bounded uncertainties based on neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 25(3): 457–469.

Zhang T, Kahn G, Levine S and Abbeel P (2016a) Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 528–535.

Zhang Y, Tao G and Chen M (2016b) Adaptive neural network based control of noncanonical nonlinear systems. *IEEE Transactions on Neural Networks and Learning Systems* 27(9): 1864–1877.

Zhou S, Helwa MK and Schoellig AP (2017) Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking. In: *Proceedings of the IEEE Conference on Decision and Control (CDC)*, pp. 5201–5207.

Zhou S, Helwa MK and Schoellig AP (2018) An inversion-based learning approach for improving impromptu trajectory tracking of robots with non-minimum phase dynamics. *IEEE Robotics and Automation Letters* 3(3): 1663–1670.

Appendix. Index to multimedia extensions

Archives of IJRR multimedia extensions published prior to 2014 can be found at http://www.ijrr.org, after 2014 all videos are available on the IJRR YouTube channel at http://www.youtube.com/user/ijrrmultimedia

Table of Multimedia Extensions

Extension	Media type	Description
1	Video	Impromptu tracking experiment
2	Data	a. Training data
		b. Testing results on 30 trajectories
		c. Operating speed generalization
		d. Difference learning scheme