Hailin Ren
Robotics and Mechatronics Lab,
Department of Mechanical Engineering,
Virginia Tech,
Blacksburg, VA 24060
e-mail: hailin@vt.edu

Jingyuan Qi
Department of Physics,
Virginia Tech,
Blacksburg, VA 24060
e-mail: jingyq1@vt.edu

Pinhas Ben-Tzvi[1]
Robotics and Mechatronics Lab,
Department of Mechanical Engineering,
Virginia Tech,
Blacksburg, VA 24060
e-mail: bentzvi@vt.edu

Learning Flatness-Based Controller Using Neural Networks

This paper presents a method to imitate flatness-based controllers for mobile robots using neural networks. Sample case studies for a unicycle mobile robot and an unmanned aerial vehicle (UAV) quadcopter are presented. The goals of this paper are to (1) train a neural network to approximate a previously designed flatness-based controller, which takes in the desired trajectories previously planned in the flatness space and robot states in a general state space, and (2) present a dynamic training approach to learn models with high-dimensional inputs. It is shown that a simple feedforward neural network could adequately compute the highly nonlinear state variables transformation from general state space to flatness space and replace the complicated designed heuristic to avoid singularities in the control law. This paper also presents a new dynamic training method for models with high-dimensional independent inputs, serving as a reference for learning models with a multitude of inputs. Training procedures and simulations are presented to show both the effectiveness of this novel training approach and the performance of the well-trained neural network. [DOI: 10.1115/1.4046776]

Keywords: dynamics and control, machine learning, neural networks, nonlinear systems, robotics

1 Introduction

Flat systems have gained popularity for analyzing and designing controllers for nonlinear systems due to their advantages in trajectory planning and tracking [1]. Flat outputs and their derivatives can be used to express the states and inputs of the original system or the extended system, simplifying the trajectory planning problem to simple algebra [2,3]. With the benefits of using a transformation based on the flatness property, a variety of interpolating functions can be used to design the path in the flat output space. This property attracts research in path planning and controllers design for various under-actuated systems, such as unicycles [4,5], quadcopters [6,7], and open chain manipulators [8].

In the last few decades, with the availability of larger datasets and more powerful computation units, new machine learning techniques have been developed to solve a wide range of problems, such as human pose estimation [9], natural language processing [10], high-level motion/primitive tasks planning [11], and others. Using artificial neural networks to imitate system models and learn suitable controllers is also an active topic in the research community. Inverse kinematics of a redundant manipulator are learned using neural networks in real-time [12]. Three types of compensation methods were proposed to improve the inverse kinematics-based controller for a robot manipulator [13]. Direct online optimization of modeling errors in dynamics is proposed to calculate the error between the real model and the analytical model to improve the control performance [14]. A closed-loop controller is then derived to control a soft pneumatically actuated manipulator using reinforcement learning [15].

In this paper, we present methods to approximate flatness-based controllers using artificial neural networks including two popular case studies: a unicycle and a quadcopter. To deal with the high-dimensional inputs of the flatness-based controller model, we developed a dynamic sampling method to generate data batches for neural network training. Compared to using a static dataset, this dynamic sampling method uses and re-uses the same memory space periodically during the training process, thereby saving the overall memory space occupied. The authors believe that this type of dataset generation will benefit research in learning models with high-dimensional inputs.

2 Preliminary Modeling Analysis

2.1 Kinematic Model of a Unicycle. Referring to a simple car model with two coaxial wheels as shown in Fig. 1(*a*), two wheels are driven independently by two motors. The inputs to the system are the right and left wheel velocities, v_r and v_l, and the configuration of the system can be fully described by the general coordinates (x, y, ψ), which encapsulate the chassis center position and heading angle. The system can be written in driftless affine-form as,

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{pmatrix} = \underbrace{\begin{pmatrix} \sin(x_3) \\ \cos(x_3) \\ 0 \end{pmatrix}}_{g_1(x)} u_1 + \underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}}_{g_2(x)} u_2 \tag{1}$$

where x_1, x_2 stands for y and x positions, respectively, and x_3 for the heading. $u_1 = v_r + v_l/2$ is the rate of heading, $u_2 = v_r - v_l/d$ is the rate of rotation, and d is the wheel separation distance. It's easy to show that the local accessibility distribution of the car system, $\langle g_1(x), g_2(x), [g_1, g_2](x)\rangle$, is of full rank, where $[\cdot, \cdot]$ presents the Lie bracket operation. The local accessibility provides this driftless system with the necessary and sufficient conditions for controllability. It can be easily checked that the model of unicycle system Eq. (1) cannot be statically linearized. However, by extending the

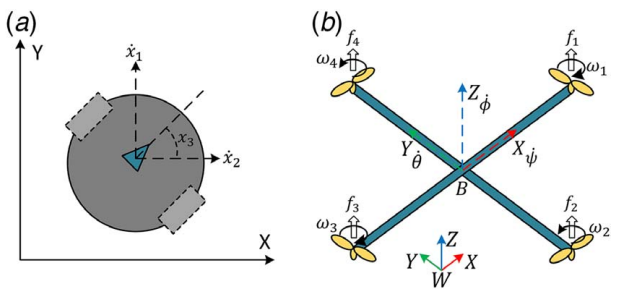

Fig. 1 Kinematic model of robot platforms

[1]Corresponding author.

Manuscript received October 23, 2019; final manuscript received February 26, 2020; published online March 27, 2020. Assoc. Editor: G. M. Clayton.

system based on a new state ($\bar{x}_4 = u_1$) while keeping others the same ($\bar{x}_i = x_i,\ \ i = 1, 2, 3$).

The extended system can be linearized to a new flat system and can also be written in Brunovsky form with a feedback law given as,

$$v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} \ddot{\bar{x}}_1 \\ \ddot{\bar{x}}_2 \end{pmatrix} = \begin{pmatrix} \sin(\bar{x}_3) & \bar{x}_4\cos(\bar{x}_3) \\ \cos(\bar{x}_3) & -\bar{x}_4\sin(\bar{x}_3) \end{pmatrix} \begin{pmatrix} \bar{u}_1 \\ \bar{u}_2 \end{pmatrix} \tag{2}$$

The system is now transformed via static feedback into a system which, in suitable coordinates, is fully linear and controllable. While adapting a classical polynomial control law for the new input variable v with disturbance compensation, the following equations can be obtained:

$$v_i = \ddot{\bar{x}}_i + \lambda_1 \cdot e_{1i} + \lambda_2 \cdot e_{2i} + \lambda_3 \cdot e_{3i}, \quad i \in \{1, 2\} \tag{3}$$

where $e_{11} = \dot{\bar{x}}_1 - \dot{\bar{x}}_{1d}$, $e_{21} = \bar{x}_1 - \bar{x}_{1d}$, $e_{12} = \dot{\bar{x}}_2 - \dot{\bar{x}}_{2d}$, $e_{22} = \bar{x}_2 - \bar{x}_{2d}$, $e_{31} = \int\bar{x}_1 - \int\bar{x}_{1d}$, and $e_{32} = \int\bar{x}_2 - \int\bar{x}_{2d}$ stand for the error signals. x_{1d} and $\dot{x}_{1d}$ represent the desired output signals, corresponding to x_1, $\dot{x}_1$. The coefficients λ_i, $i = 0, \ldots, 5$ are specified in Table 2.

2.2 Dynamic Model of a Quadcopter. A quadcopter that is composed of four rotors is detailed in Fig. 1(*b*). Speeding up or slowing down either rotor pair can control the Yaw angle ψ. The Roll ϕ and Pitch angles θ allow the quadcopter to move in the Y- and X-directions, respectively. The rotor is the primary source of control and propulsion for the unmanned aerial vehicle (UAV). Z–Y–X Euler angles (ψ, ϕ, θ) are applied with the conditions ($-\pi \le \psi < \pi$) for yaw, ($-\pi \le \phi < \pi$) for pitch, and ($-\pi \le \theta < \pi$) for roll, respectively.

Using Newtonian laws about the center of mass, the dynamic equations for the quadcopter are obtained:

$$m\dot{V}_0 = \sum F_{ext}, \quad J\dot{\omega} = -\omega \times J\omega + \sum T_{ext} \tag{4}$$

where the symbol $\times$ is the usual vector product, m is the mass, and J is the inertia matrix. The coordinate frame is attached to the UAV body frame as shown in Fig. 1(*b*). The angular velocity $\omega = (\omega_x,\ \omega_y,\ \omega_z)^T$ and angular acceleration $\alpha = (\alpha_x,\ \alpha_y,\ \alpha_z)^T$ of the body frame with respect to frame **B** are functions of the first- and second-time derivatives of the Euler angles $(\dot{\psi}, \dot{\theta}, \dot{\phi})$ and $(\ddot{\psi}, \ddot{\theta}, \ddot{\phi})$. The notations $\sum F_{ext}$, $\sum T_{ext}$ stand for the vector of external forces and torques, respectively. They can be computed as,

$$\begin{aligned} \sum F_{ext} &= \begin{pmatrix} A_x - (s\psi s\phi + c\psi c\phi s\theta)u_1 \\ A_y - (s\psi c\phi s\theta - c\psi s\phi)u_1 \\ A_z + mg - c\phi c\theta u_1 \end{pmatrix}, \\ \sum T_{ext} &= \begin{pmatrix} A_p + u_2 d \\ A_q + u_3 d \\ A_r + u_4 \end{pmatrix} \end{aligned} \tag{5}$$

where (A_x, A_y, A_z) and (A_p, A_q, A_r) are aerodynamic forces and moments acting on the UAV. u_1 is the resulting thrust of the four rotors defined as $u_1 = (F_1 + F_2 + F_3 + F_4)$; u_2 is the difference of thrust between the left rotor and the right rotor defined as $u_2 = F_4 - F_2$; u_3 is the difference of thrust between the back rotor and the front rotor defined as $u_3 = F_3 - F_1$; u_4 is the difference of torque between the two clockwise-turning rotors and the two counterclockwise-turning rotors defined as $u_4 = C_{fm}[(F_1 + F_3) - (F_2 + F_4)]$, and C_{fm} is the force to moment scaling factor.

To avoid Lie transformation matrices singularity, we replace the real control signals (u_1, u_2, u_3, u_4) with ($\bar{u}_1$, $\bar{u}_2$, $\bar{u}_3$, $\bar{u}_4$). u_1 is delayed by a double integrator, while the other control signals are unaltered based on the following variable transformations.

$$u_1 = \zeta; \quad \dot{\zeta} = \xi; \quad \dot{\xi} = \bar{u}_1; \quad u_2 = \bar{u}_2; \quad u_3 = \bar{u}_3; \quad u_4 = \bar{u}_4 \tag{6}$$

The resulting system then can be described by state space equations in the following form:

$$\dot{s} = \bar{f}(s) + \sum_{i=1}^{4} \bar{g}_i(s)\bar{u}_i, \quad y = h(s) \tag{7}$$

where $s = (x, y, z, \psi, \phi, \theta, \dot{x}, \dot{y}, \dot{z}, \zeta, \xi, p, q, r)$, $o = (x,\ y,\ z,\ \psi)$. Details of the modeling can be found in Ref. [16].

The input–output decoupling problem is solvable for the nonlinear system by means of static feedback, and the vector relative degree r_1, r_2, r_3, r_4 is given by $r_1 = r_2 = r_3 = 4$; $r_4 = 2$. We choose the flatness outputs as

$$o = [y_1, y_2, y_3, y_4]^T = [x, y, z, \psi]^T \tag{8}$$

then obtain a feedback control law,

$$v = [y_1^{r_1}, y_2^{r_2}, y_3^{r_3}, y_4^{r_4}]^T = b(x) + \Delta(\boldsymbol{x})\bar{u} \tag{9}$$

where v represents the new input control signals. The matrix $\Delta(\boldsymbol{x})$ is non-singular everywhere in the region $\zeta \neq 0$, $-\pi/2 < \theta < \pi/2$, $-\pi/2 < \phi < \pi/2$. Therefore, the input–output decoupling problem is solvable for our system by means of a control law in the form of

$$\bar{u} = -\Delta^{-1}(s)b(s) + \Delta^{-1}(s)v \tag{10}$$

The system can be transformed via static feedback into a system which, in suitable coordinates, is fully linear and controllable. While adapting a classical polynomial control law for the new input variable v with disturbance compensation, the following set of equations can be obtained:

$$\begin{aligned} v_1 &= x_d^{(4)} - \lambda_3 \dddot{e}_{11} - \lambda_2 \ddot{e}_{11} - \lambda_1 \dot{e}_{11} - \lambda_0 e_{11} \\ v_2 &= y_d^{(4)} - \lambda_3 \dddot{e}_{12} - \lambda_2 \ddot{e}_{12} - \lambda_1 \dot{e}_{12} - \lambda_0 e_{12} \\ v_3 &= z_d^{(4)} - \lambda_3 \dddot{e}_{13} - \lambda_2 \ddot{e}_{13} - \lambda_1 \dot{e}_{13} - \lambda_0 e_{13} \\ v_4 &= \ddot{\psi}_d - \lambda_5 \dot{e}_5 - \lambda_4 e_5 \end{aligned} \tag{11}$$

where x_d, y_d, z_d, ψ_d represent the desired output signals, corresponding to x, y, z, ψ, respectively, and the error signals $e_{11} = x_0 - x_{0d}$, $e_{12} = y_0 - y_{0d}$, $e_{13} = z_0 - z_{0d}$, and $e_5 = \psi - \psi_d$. The coefficients λ_i, $i = 0, \ldots, 5$ are specified in Table 2. The mass properties of the quadcopter in our case study are chosen as shown in Table 1. Table 2 outlines the controller parameters used in the unicycle controller and the quadcopter controller. In the quadcopter control, only the body positions x, y, z are of interest, and the control parameters of the yaw control are all set to zero, $\lambda_4 = \lambda_5 = 0$.

Table 1 Mass properties of the quadcopter

Parameter	Description	Value	Units
Gravity	g	9.81	(m/s^2)
Quadcopter diameter	d	0.45	(m)
Mass	m	0.468	(Kg)
Quadcopter inertia	$I_{x,y,z}$	$4.9 \cdot 10^{-3}$	($Kg \cdot m^2$)

Table 2 Controller parameters for the unicycle and quadcopter

	λ_0	λ_1	λ_2	λ_3
Unicycle	N/A	0.085	0.0025	0.0001
Quadcopter	0.0025	0.0025	1.0000	0.0100

3 Designing Flatness-Based Controller

3.1 Dynamic Sampling Approach. Generalization over the entire input space is one of the principles to estimate the performance of the trained neural network in machine learning. One way to achieve generalization is to create a large enough dataset that covers the entire input space, while the other is to split the dataset into a training set and a validation set. The former ensures generalization, but is computationally expensive, especially when applied to problems with high-input dimension, and is often infeasible in continuous input problems [17]. The latter helps avoid overtraining in the subset, but is prone to fail when the dataset is not large enough or only partially covers the input space [18].

To deal with the proposed high-dimensional continuous input problems, we propose a dynamic sampling approach that marginally splits the entire workspace into a training set and a validation set. For a problem with n dimension inputs, $x = [x_1, x_2, \ldots, x_n]$, with their lower boundaries x_i^l and upper boundaries x_i^u. Discretization is applied to each input x_i uniformly into n_i bins with a fixed interval δ_i for x_i. The dataset is now converted to a finite set $X_{data} = \{[x_1, x_2, \ldots, x_n]\}$, $x_i \in \{x_i^l, x_i^l + \delta_i, x_i^l + 2\delta_i, \ldots, x_i^u\}$. The size of the dataset X_{data} increases dramatically when the input dimension increases and when the discretization becomes denser. Instead of storing the whole dataset, only the relatively smaller set, the validation dataset X_{valid} is stored during the training process. To recover the continuous input space, sampling is performed based on the discrete data using a predefined distribution. The training procedure with dynamic sampling approach is described in the form of following pseudocode:

procedure Training Network
($\{x_i^l\}$, $\{x_i^u\}$, n_i)
1: Obtain discretization interval δ_i for x_i
2: Randomly generate validation set X_{valid}
3: **for** each training epoch, i **do**
4: **for** each training batch, j **do**
5: Generate discrete training set $x_{batch}^d \in \{x | x \in X_{data} - X_{valid}\}$ based on $\{x_i^l\}$, $\{x_i^u\}$
6: Generate continuous training set x_{batch}^c on the distribution over x_{batch}^d
7: Training neural network using x_{batch}^c
8: **end for**
9: **end for**

Here, we applied i.i.d Gaussian sampling to the selected discrete training set to recover to continuous space $x_{batch}^c \sim p(x_{batch}^d | \{\delta_i\})$. For data along each dimension, the selected discrete value is used as the mean value $\mu = x^d$, while the standard deviation is selected to be the discretization interval $\sigma = \delta$. The generated training data along each dimension obeys Gaussian distribution $x^c \sim \mathcal{N}(x^d, \delta)$. To avoid overfitting on any specific trajectory, the boundaries of the inputs $\{x_i^l, x_i^u\}$ were provided by generating random trajectories using patterns such as a circle, a square, and a line. These input domains were then discretized and divided into training and validation datasets. The discretized datasets were only used in the training process, while the final deployment of the well-trained neural network does not require them.

3.2 Neural Network Design and Training. Neural networks are widely used to better approximate highly nonlinear models or models with uncertainty to increase the performance and stability of controllers. To approximate the flatness-based controller of the unicycle and the quadcopter, multilayer perceptron neural network is used. The proposed neural network consists of the fully connected neurons in each layer with linear activation function in all neurons. The neural network is built using Keras [19] with TensorFlow [20] as the backend. Hyperopt [21] is used to tune the hyperparameters of the neural network including number of layers n_l, number of neurons in each layer n_n, activation functions used for neurons in each layer except the final output layer f_a, and the parameters for the activation if needed f_p. Details of the choices and selected choices of the hyperparameters are presented in Table 3. The proposed neural network consists of five layers with 512 neurons in each layer. An activation function in each neuron is selected to be Leaky ReLU [22] with $\alpha = 0.5$ for the optimal performance.

Table 3 Hyperparameters tuning for neural network

Parameters	Choices	Selected
n_l	[4, 5, 6, 7, 8]	5
n_n	[128, 256, 512, 1024]	512
f_a	["ReLU," "Sigmoid," "Leaky ReLU"]	"Leaky ReLU"
f_p	[0.1, 0.3, 0.4, 0.5, 0.7, 0.9]	0.5

Table 4 Loss weights used for the optimization

ω_i	$\bar{u}_1$	$\bar{u}_2$	$\bar{u}_3$	$\bar{u}_4$
Unicycle	1	1	N/A	N/A
Quadcopter	1000	1,000,000	100,000	100,000,000

Based on control laws in Eqs. (2) and (3) for the unicycle and Eqs. (10) and (11) for the quadcoper, the neural network takes in the error signals and states and outputs the control signals for the model. We define our objective function to minimize the weighted loss function based on the order of magnitude of our controller outputs:

$$\min_{\Theta} L = \sum_{i=1}^{n} \omega_i \|\bar{u}_i - \bar{u}_i^{\Theta}\|_2 \tag{12}$$

where Θ represents the trainable parameters of the neural network, ω_i is the weight for the loss of the ith outputs, $\bar{u}_i$ is the output value from the flatness-based controller and is used as the ground truth, while $\bar{u}_i^{\Theta}$ is the estimated output from the neural network. For the unicycle controller, $n = 2$ outputs are designed in the neural network with the loss weights $\omega_1 = \omega_2 = 1$, while the neural network of the quadcopter controller generates $n = 4$ outputs with loss weights, as shown in Table 4. The training process includes 100 training epochs. Within each training epoch, 1000 training batches are performed, followed by validation using 100 batches of data. Weighted root-mean-square error (WRMSE) is used in the validation process:

$$WRMSE = \sqrt{\sum_{i=1}^{n} \omega_i (\bar{u}_i - \bar{u}_i^{\Theta})^2 / n} \tag{13}$$

Figure 2 shows the training processes of flatness-based controllers for the unicycle and the quadcopter. It can be seen that the validation loss decreases as the training process progresses. It can be seen that the validation error, WRMSE, converges to approximately 4×10^{-3} for the unicycle model and approximately 0.1 for the quadcopter model.

4 Validation of Proposed Approach

To evaluate the performance of the well-trained controller using neural network in real applications, we compared our trained controller to the mathematical model-based controller in three different motions. For the unicycle, point-to-point, square, and circle motions are performed. The quadcopter performs similar motions in 3D with various vertical heights. The simulation was performed using

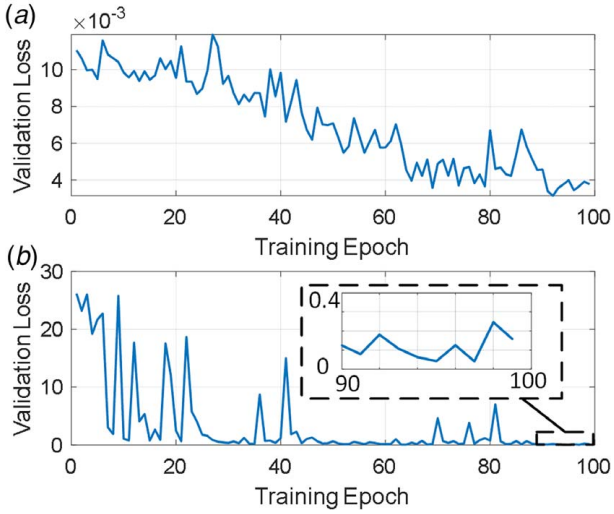

Fig. 2 Training process of (*a*) the kinematic model of the unicycle and (*b*) the dynamic model of the quadcopter

Simulink [23] as shown in Fig. 3. All trajectories are generated using the trajectory planner. The desired trajectories are then sent to the controller, which could be either the well-trained neural network controller or the flatness-based controller that uses an explicit mathematical model. The simulated model uses controller outputs for its calculations.

4.1 Trajectory Planner. Given the diffeomorphism, a point-to-point motion planning and tracking control can be designed in a flatness workspace. For example in the unicycle problem, given the terminal conditions in the Cartesian coordinates at both start time t_0 and final time t_f, $\{x_i(t), \dot{x}_i(t); i \in \{1, 2, 3\}, t \in \{t_0, t_f\}\}$, the terminal conditions can be transformed to the corresponding terminal conditions in the flatness workspace $\{\bar{x}_i(t), \dot{\bar{x}}_i(t), \ddot{\bar{x}}_i(t); i \in \{1, 2\}, t \in \{t_0, t_f\}\}$. To satisfy these six terminal conditions, we choose three types of functions as trajectories for $\bar{x}_i(t); i \in \{1, 2\}$: fifth-order polynomials, trigonometric functions, and piecewise functions. For fifth-order polynomials, we used the following functions:

$$\bar{x}_i(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0 \quad i = \{1, 2\} \tag{14}$$

For trigonometric functions, we choose following functions:

$$\bar{x}_1(t) = r \cdot \cos(2\pi \cdot t/T), \quad \bar{x}_2(t) = r \cdot \sin(2\pi \cdot t/T) \tag{15}$$

where r is the radius of the circle and T is the period in which the unicycle runs around the circle. The desired trajectory is a circle that starts from (10,0) at the top, running clockwise with radius 10 m, all the way back to (10,0). In the simulation, we choose the real initial point to be (−0.5, 8) to test the convergence.

We choose the following formula for our piecewise function:

$$\bar{x}_i(t) = a_{5j} t^5 + a_{4j} t^4 + a_{3j} t^3 + a_{2j} t^2 + a_{1j} t + a_{0j}$$
$$T_j < t \le t_{i+1}, \; i = \{1, 2\}, \quad j = \{1, 2, \ldots, 5\} \tag{16}$$

The desired trajectory $(\bar{x}_1(t), \bar{x}_2(t))$ is a square, going from (0, 0) to (0, 24), then to (24, 24), then (24, 0), and finally back to (0, 0). In the simulation, we choose the real initial point at (−2, 2) to test convergence.

A similar approach was used in the quadcopter process. We choose the same three types of trajectory functions that were used in the unicycle model. For the fifth-order polynomials, we choose the same function as Eq. (14). For trigonometric functions of the quadcopter, we appended one more function to Eq. (15) to accommodate for the motion in the Z-direction:

$$\bar{x}_3(t) = -1 - t/10 \tag{17}$$

where $\bar{x}_3$ is the position along the Z-direction representing height. The desired trajectory, $(\bar{x}_1(t), \bar{x}_2(t), \bar{x}_3(t))$, represents a circle, that starts from (0, 15, −1) at the top, running clockwise with radius 10 m all the way to (0, 15, $-1-10/t$). The quadcopter simultaneously decreases its height at a constant rate. In the simulation, we choose the real initial point at (1,2,3) to test convergence. Equation (16) is used as a piecewise function. The desired trajectory takes the form of a square going from (0,0,0) to (45,0,55), then to (45,45,0), then (0,45,55), and finally back to (0,0,0). In the simulation, we choose the real initial point to be (1,2,3) to test the convergence.

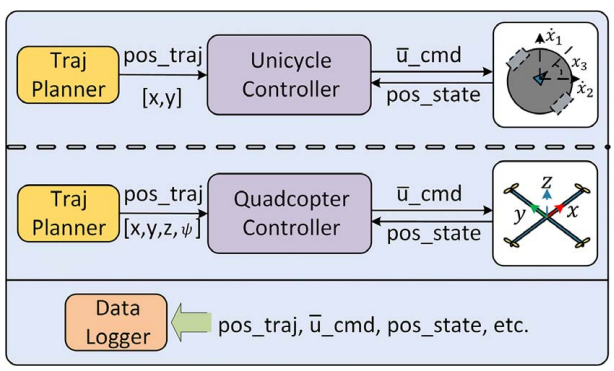

Fig. 3 Simulation system for the unicycle and the quadcopter

4.2 Trajectory Tracking Performance. Three motions of the unicycle were performed and compared in Figs. 4(*a*)–4(*c*) while these of the quadcoper are presented in Figs. 4(*d*)–4(*e*). One of the most straightforward measures of controller performance is evaluating the average deviation from the desired trajectory over the whole tracking process, using root-mean-square error (RMSE),

$$RMSE = \sum_{i=1}^{n} \|f_i - g_i\|_2 \tag{18}$$

where f_i and g_i are the two compared trajectories at timestep i. Another way to measure controller performance is to capture the maximum "overshoot" during the tracking process, using the maximum absolute deviation values,

$$MAD = \max \|f_i - g_i\|_2 \tag{19}$$

where f_i and g_i are the two compared trajectories at timestep i. Table 5 shows the performance criteria based on the three motions. It can be seen that the neural network-based controller obtained similar overall performance compared with the mathematical model-based controller. The slightly increased tracking error obtained from the neural network results is from the training error. To validate the proposed method, the same hyperparameters, instead of the best hyperparameters for individual cases, were selected for both the unicycle and the quadcopter cases, which led to the different tracking performance. For example, larger control inputs were applied in most unicycle cases that led to larger tracking errors. In all the tracking cases, the neural network-based controller followed the desired trajectories with slight differences compared with the original mathematical model-based controllers. These tracking performances successfully validated the proposed dynamic sampling method.

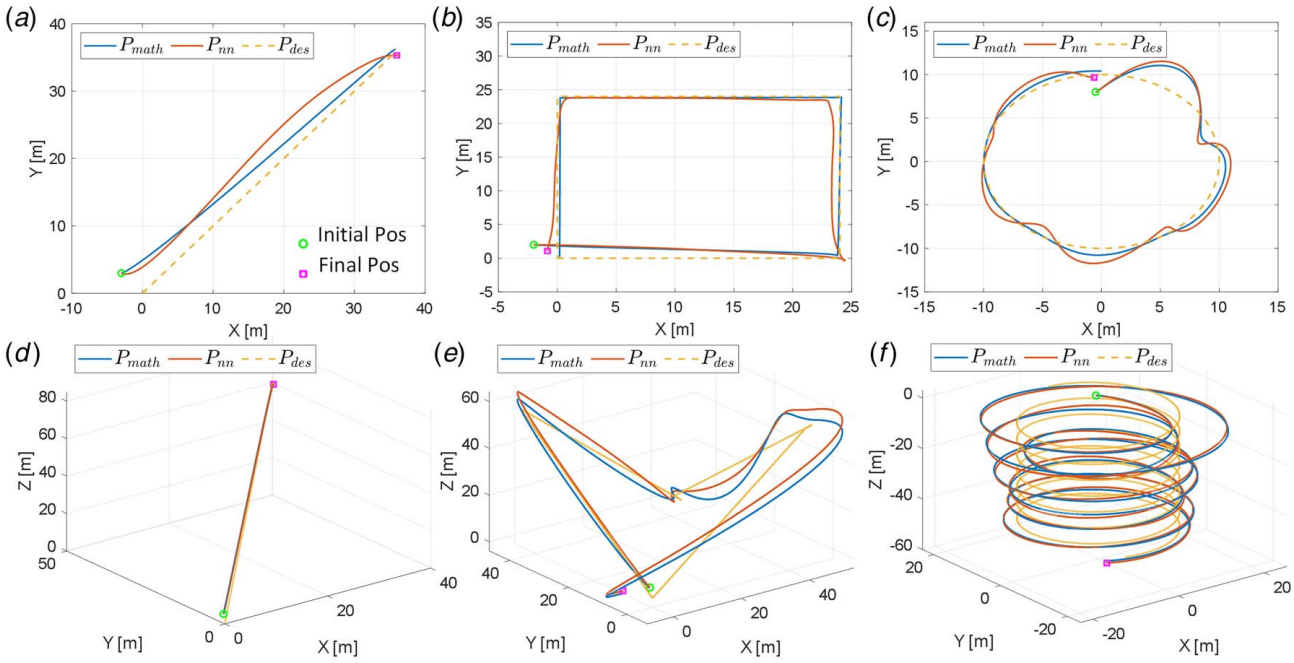

Fig. 4 Point-to-point, Square and Circle motion for (*a*)–(*c*) the unicycle and (*d*)–(*f*) the Quadcopter

Table 5 Performance on different motions

	math[a]		nn[b]	
	RMSE	MAD	RMSE	MAD
		Unicycle		
Linear	2.407	2.007	3.035	2.801
Square	1.014	0.689	1.970	1.665
Circle	0.899	0.700	1.346	1.208
	math		cnn	
	RMSE	MAD	RMSE	MAD
		Quadcopter		
Linear	1.834	1.345	1.836	1.356
Square	1.320	2.393	1.803	3.255
Circle	4.825	9.448	4.754	9.290

[a]math: mathematical model-based controller.
[b]nn: neural network-based controller.

5 Conclusion and Future Work

In this work, flatness-based controllers were accurately approximated using neural networks. Two case studies were presented, including a kinematic model for a unicycle and a dynamic model for a quadcopter. A dynamic sampling method was proposed to avoid large memory allocation during the training for problems with high-dimensional input. A well-trained model was simulated using three different motions, and the imitating performance of our neural network controller was quantified in comparison to a mathematical model-based controller. It was shown that the neural network-based controller was able to emulate two complex nonlinear controllers. The proposed dynamic sampling method was also useful for training high-input dimension neural networks with ground truth modeling.

Future work involves training the neural network controller using real-world data to include the uncertainty in mathematical modeling to make it more robust and stable. Neural network architectures need to be improved to perform adequate learning with limited real-world data, which is expensive to collect.

Acknowledgment

The authors would like to gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

[1] Sira-Ramírez, H., and Agrawal, S., 2004, *Differentially Flat Systems*, Vol. 5, CRC Press, Boca Raton, FL.

[2] Francisco, S., Murray, R. M., Rathinam, M., and Sluis, W., 1995, "Differential Flatness of Mechanical Control Systems: A Catalog of Prototype Systems," Proceedings of the 1995 ASME International Congress and Exposition, San Francisco, CA, Nov. 12–17, ASME.

[3] Soheil-Hamedani, M., Zandi, M., Gavagsaz-Ghoachani, R., Nahid-Mobarakeh, B., and Pierfederici, S., 2016, "Flatness-Based Control Method: A Review of Its Applications to Power Systems," 2016 7th Power Electronics and Drive Systems Technologies Conference (PEDSTC), Tehran, Iran, Feb. 16–18, IEEE, pp. 547–552.

[4] Tang, C. P., 2009, "Differential Flatness-Based Kinematic and Dynamic Control of a Differentially Driven Wheeled Mobile Robot," 2009 IEEE International Conference on Robotics and Biomimetics (ROBIO), Guilin, China, Dec. 19–23, IEEE, pp. 2267–2272.

[5] De Luca, A., Oriolo, G., and Samson, C., 1998, "Feedback Control of a Nonholonomic Car-Like Robot," *Robot Motion Planning and Control*, Springer-Verlag, Berlin, pp. 171–253.

[6] Poultney, A., Kennedy, C., Clayton, G., and Ashrafiuon, H., 2018, "Robust Tracking Control of Quadrotors Based on Differential Flatness: Simulations and Experiments," IEEE/ASME Trans. Mechatronics, **23**(3), pp. 1126–1137.

[7] Cowling, I. D., Yakimenko, O. A., Whidborne, J. F., and Cooke, A. K., 2007, "A Prototype of An Autonomous Controller for a Quadrotor UAV," 2007 European Control Conference (ECC), Kos, Greece, July 2–5, IEEE, pp. 4001–4008.

[8] Agrawal, S., and Sangwan, V., 2008, "Differentially Flat Designs of Underactuated Open-Chain Planar Robots," IEEE Trans. Rob., **24**(6), pp. 1445–1451.

[9] Ren, H., Kumar, A., Wang, X., and Ben-Tzvi, P., 2018, "Parallel Deep Learning Ensembles for Human Pose Estimation," Dynamic Systems and Control Conference, Atlanta, GA, Sept. 30–Oct. 3, ASME, p. V001T07A005.

[10] Young, T., Hazarika, D., Poria, S., and Cambria, E., 2018, "Recent Trends in Deep Learning Based Natural Language Processing [Review Article]," IEEE Comput. Intell. Mag., **13**(3), pp. 55–75.

[11] Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T., 2018, "Learning by Playing - Solving Sparse Reward Tasks From Scratch," Proceedings of Machine Learning Research, Stockholmsmässan, Stockholm, Sweden, July 10–15, PMLR, pp. 4344–4353.

[12] Toshani, H., and Farrokhi, M., 2014, "Real-Time Inverse Kinematics of Redundant Manipulators Using Neural Networks and Quadratic Programming: A Lyapunov-based Approach," Rob. Autonomous Syst., **62**(6), pp. 766–781.

[13] Pane, Y. P., Nageshrao, S. P., Kober, J., and Babuška, R., 2019, "Reinforcement Learning Based Compensation Methods for Robot Manipulators," Eng. Appl. Artif. Intell., **78**(2), pp. 236–247.
[14] Ratliff, N., Meier, F., Kappler, D., and Schaal, S., 2016, "DOOMED: Direct Online Optimization of Modeling Errors in Dynamics," Big Data, **4**(4), pp. 253–268.
[15] Thuruthel, T. G., Falotico, E., Renda, F., and Laschi, C., 2019, "Model-Based Reinforcement Learning for Closed-Loop Dynamic Control of Soft Robotic Manipulators," IEEE Trans. Rob., **35**(1), pp. 124–134.
[16] Faessler, M., Franchi, A., and Scaramuzza, D., 2018, "Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories," IEEE Rob. Autom. Lett., **3**(2), pp. 620–626.
[17] Watkins, C. J. C. H., 1989, "Learning From Delayed Rewards," Ph.D. thesis, King's College, Cambridge, UK.
[18] Reitermanová, Z., "Data Splitting," WDS's 10 Proceedings of Contributed Papers, pp. 31–36.
[19] Chollet, F., et al., 2015, "Keras."
[20] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X., 2015, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," tensorflow.org
[21] Bergstra, J., Yamins, D., and Cox, D. D., 2013, "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures," 30th International Conference on International Conference on Machine Learning, Vol. 28, pp. I–115–I–123.
[22] Maas, A. L., Hannun, A. Y., and Ng, A. Y., 2013, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," 30th International Conference on Machine Learning, June 16–21, Atlanta, GA, ICML.
[23] MathWorks. Simulink - Simulation and Model-Based Design - MATLAB.