Multistep Prediction of Dynamic Systems With Recurrent Neural Networks

Nima Mohajerin, *Member, IEEE*, and Steven L. Waslander, *Member, IEEE*

***Abstract*— In this paper, we address the state initialization problem in recurrent neural networks (RNNs), which seeks proper values for the RNN initial states at the beginning of a prediction interval. The proposed methods employ various forms of neural networks (NNs) to generate proper initial state values for RNNs. A variety of RNNs are trained using the proposed NN initialization schemes for modeling two aerial vehicles, a helicopter and a quadrotor, from experimental data. It is shown that the RNN initialized by the NN-based initialization method outperforms the washout method which is commonly used to initialize RNNs. Furthermore, a comprehensive study of RNNs trained for multistep prediction of the two aerial vehicles is presented. The multistep prediction of the quadrotor is enhanced using a hybrid model, which combines a simplified physics-based motion model of the vehicle with RNNs. While the maximum translational and rotational velocities in the Quadrotor data set are about 4 m/s and 3.8 rad/s, respectively, the hybrid model produces predictions, over 1.9 s, which remain within 9 cm/s and 0.12 rad/s of the measured translational and rotational velocities, with 99% confidence on the test data set.**

***Index Terms*— Multistep prediction, nonlinear dynamic system modeling, recurrent neural networks (RNNs), state initialization.**

I. Introduction

MULTISTEP prediction of dynamic motion remains a challenging problem. The simplifying assumptions required in the process of modeling dynamic systems lead to *unmodeled dynamics*, which, in turn, lead to repeatable and systematic prediction errors. When the prediction is required over very short periods into the future, the unmodeled dynamics may cause a negligible error; however, using the model over longer prediction horizons, the unmodeled dynamics can lead to drastic growth in the prediction error over time. In the discrete-time domain, a prediction required for one time step into the future is referred to as a single-step prediction, and a prediction many steps into the future is referred to as a multistep prediction.

In this paper, the multistep prediction of mobile robotic systems is considered. Recurrent neural networks (RNNs) are mainly employed to learn the dynamics of two aerial vehicles, a helicopter and a quadrotor, from experimental data. The Helicopter data set belongs to the Stanford Helicopter Platform,[1] which has been used in apprenticeship learning [1] and single-step prediction and modeling of the platform [2]. The Quadrotor data set, however, has been specifically collected for this paper.

Multistep prediction has many applications in state estimation, simulation, and control [3], [4]. For example, model-based control schemes, such as model-predictive control (MPC) [5], can extensively benefit from an accurate long-term prediction. Accurate multistep predictions allow for a slower update rate of the MPC, reducing the overall computational burden while maintaining smoothness and accuracy of the resulting control system response. As another example, in a moving vehicle when some measurements, such as GPS readings, are temporarily unavailable, a multistep prediction can account for the missing measurements and approximate the system position and speed over the blackout period.

The classic method of modeling, i.e., modeling from first principles (or white-box methods), suffers from two major difficulties. First, the developed model will contain many parameters which describe the system physical characteristics (mass, drag coefficient, and so on) and must be properly identified prior to using the model. Second, many properties of the system might be too difficult to model explicitly, such as the vortex-ring effect on a quadrotor vehicle [6]. Identifying the parameters of a model can be expensive. For instance, measuring the blade drag coefficient of a quadrotor needs a wind tunnel. Moreover, by changing the system physical properties slightly, the model should be adapted accordingly, which may involve new measurements and cumbersome tests.

On the other hand, learning-based (or black box) modeling relies on the observations from the system to decipher complex nonlinearities acting on the system. There are several black-box architectures, such as polynomial models (the Wiener model, Voltera series, and so on), Fuzzy models [7], [8], and neural networks (NNs) [9], [10]. Regardless of the method, a black-box model has many degrees of freedom (DOFs), depicted as parameters or weights, that should be found based on a set of input–output observations from the system. The search to find the appropriate values for the model parameters is usually done through an optimization process, and since many black-box models are machine learning (ML) methods, the parameter optimization process is frequently referred to as a *learning* process.

Manuscript received May 8, 2018; revised October 27, 2018; accepted December 25, 2018. Date of publication January 31, 2019; date of current version October 29, 2019. This work was supported by the NSERC Canadian Field Robotics Network. *(Corresponding author: Nima Mohajerin.)*

N. Mohajerin was with the Department of Mechanical and Mechatronics Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada. He is now with the Noah's Ark Lab, Huawei Technologies Canada, Markham, ON L3R 5A4, Canada (e-mail: nima.mohajerin@uwaterloo.ca).

S. L. Waslander is with the Institute of Aerospace Studies, University of Toronto, Toronto, ON M3H 5T6, Canada.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2891257

[1]http://heli.stanford.edu/data set/

In recent years, ML has been going through rapid development. Deep learning (DL) [11], [12] is revolutionizing the ML field and solving problems that could not have been solved before, such as speech recognition [13] and object detection and classification in images [14], [15]. Three main components propel this revolution: 1) the newly available hardware capable of performing efficient parallel processing on large numerical models; 2) methods to train these models; and 3) availability of large-scale data sets. Because of the third reason, the applications of DL have been mainly focused on natural language processing and image classification for which such data sets are readily available. However, DL methods can also be used in modeling and control of robotic systems. In fact, in mobile robotics, where the robot is deployed in an unstructured environment with noisy and uncertain sensory information, learning from observation can deal with problems that are either too difficult or too expensive to handle with classical methods. This paper demonstrates some of the capabilities and applicability of ML methods in such applications. Particularly, in this paper, we show how to employ RNNs to accurately predict the behavior of the two robotic systems using input information only, many steps into the future.

Using RNNs in multistep prediction requires a proper state initialization, that is, assigning proper initial values to the neuron outputs at the first step of prediction. The underlying reason is that the RNN recursively updates itself throughout the prediction horizon and therefore exhibits dynamics. Like any other dynamic system, the (transient) response of an RNN, in general, depends heavily on its initial condition. For an RNN with hidden layers, the common approach is to initialize the hidden neuron outputs to zero (or random) values and run the network until the effect of the initial values washes out [16], [17]. Some of the drawbacks in the washout method are as follows.

1) The washout period is not fixed and hard to determine *a priori*, during which the network does not produce a reliable prediction.
2) The network may become unstable during the washout period, leading to prolonged or even failed training sessions.
3) In the training process, the input sequence used in washout does not contribute to the learning process.

The third drawback listed above is more severe when the data set collection process is expensive, which is often the case with experimental data acquisition in the robotics domain.

In this paper, the RNN state initialization problem is carefully studied. An NN-based initialization method, previously proposed in [18], is revisited and expanded. It is shown that the network initialized with the NN-based method significantly outperforms the same network initialized by the washout method. After establishing the effectiveness and efficiency of our NN-based initialization method, the results of black-box modeling of the two aerial vehicles are demonstrated and studied. To further improve the quadrotor model, the hybrid model in [19] is revisited. This paper embodies the capability of NNs in learning unmodeled dynamics of a real and challenging robotic system for multistep prediction, for the first time, and may serve as a basis for future development and application of more sophisticated learning-based methods in modeling, prediction, and control of such systems.

The contributions of this paper can be summarized as follows.

1) State initialization problem for RNNs is defined and addressed with a cascaded architecture. The proposed method outperforms the washout method, which is commonly used for RNN state initialization.
2) Using the proposed state initialization method, a variety of black-box models are trained for multistep prediction of the behavior of two aerial vehicles, a helicopter and a quadrotor, from experimental data sets.
3) As a part of this paper, a data set comprised of quadrotor flights is collected with the total recorded flight time of approximately 3 h and 50 min. The data set is made publicly available.[2]
4) Hybrid models to incorporate the modeling capability of initialized RNNs with physics-based dynamic models are proposed. The hybrid models are trained on the aforementioned data sets and show a significant improvement in multistep prediction.

Following this introduction, in Section II, a brief background as well as the related literature are reviewed. Section III formulates the problems we tackle in this paper—the multistep prediction and the RNN state initialization problems. Section IV describes the washout method and our proposed solutions for the state initialization problem. In Section V, the model architectures that are trained for identifying the helicopter and quadrotor models are described. In Section VI, the proposed state initialization schemes are evaluated against the commonly used washout method and the trained models of the two aerial vehicles are assessed. Finally, Section VII concludes this paper.

II. Background and Literature Review

Feedforward neural networks (FFNNs) have been used extensively in modeling and control of dynamic systems [20]–[25]. In a control problem, they may be employed as a modeling part of a controller in a Lyapunov design approach. In this method using a Lyapunov function, the equations for the evolution of the NN weights are extracted so that the closed-loop controller stabilizes the system [26]–[29]. Since FFNNs lack exhibiting dynamics, they are mainly used as a single-step predictor or a compensator.

RNNs possess dynamics and are universal approximators for reconstructing state-space trajectories of dynamic systems [30]–[32], which make them suitable candidate models for multistep prediction problem. In [30], it is shown that any finite-time trajectory of a dynamic system can be approximated by some RNNs to any desired level of accuracy given a *proper initial state*. This result is extended to discrete RNNs in [31]. This is another aspect to reinforce the importance of a proper state initialization for RNN in modeling dynamic systems.

Nonlinear autoregressive (NAR) models are classic tools to model dynamic systems [10], [33], [34]. Narendra and

[2]https://github.com/wavelab/pelican_data set

Parthasarathy [33] devised methods to use multilayer perceptrons (MLPs) in the NAR eXogenous (NARX) framework. In a discrete-time fashion, the NARX framework implements a dynamic system whose output at any given time, y_k, is a function of the input at that time, u_k, and the system output and input at previous time steps

$$y_k = f(y_{k-1}, \dots, y_{k-\tau_y}, u_k, u_{k-1}, \dots, u_{k-\tau_u})$$

where the length of the memory buffers, i.e., τ_u and τ_y, is usually given or determined through a hyperparameter optimization process. The function $f(.)$ can be realized by various methods. In [33], the function $f(.)$ is realized by an MLP. To avoid confusion, the method to implement this function is added to the NARX abbreviation as a suffix. For instance, if $f(.)$ is realized by an MLP, then the architecture will be referred to as NARX-MLP. An NARX-MLP is essentially an MLP equipped with buffers and feedback connections. Hence, it can be classified as an RNN.

The NARX-MLP architectures are often trained via a *series–parallel* [33] mode, which uses the network target values instead of the network past outputs as the delayed version(s) of the network output. This method is also known as *teacher forcing* [33]. Clearly, this mode converts the NARX architecture into a feedforward one, which, therefore, loses the main advantage of an RNN and limits the ability of the method to represent dynamical systems accurately. On the other hand, training NARX-MLP in a closed-loop form (parallel mode) to model dynamic systems can be difficult due to numerical stability issues in the calculation of the gradient for learning-based optimization [35].

One alternative to the NARX model is to define an internal state, x_k, and use a one-step memory buffer

$$\begin{aligned} x_k &= f(x_{k-1}, y_{k-1}, u_k) \\ y_k &= g(x_k). \end{aligned}$$

This architecture is an example of a recurrent MLP (RMLP). An RMLP is made by one (or a few) locally recurrent layer of sigmoid neurons [36]. RMLPs have been used in a number of dynamic system identification and modeling problems, such as a heat exchanger [37], engine idle operation [38], and wind power prediction [39].

It is not clear whether using RMLPs is more advantageous than NARX-MLPs. However, in [40], it is shown that NARX-MLPs, in a single-step prediction scenario, outperform RMLPs in modeling a real helicopter. NARX RNNs have been extensively studied [41], [42] and used in various modeling and identification problems [43]–[45]. In [46], a radial basis function (RBF) network in an NARX architecture, i.e., NARX-RBF, is used to model and control a quadrotor with three horizontal propellers and one vertical propeller. In [47], another form of NARX-RBF is employed and trained using the Levenberg–Marquardt optimization algorithm to model a small-scale helicopter. Both approaches employ teacher forcing.

Recently, Punjani and Abbeel [2] used rectified linear unit NNs to model dynamics of a real helicopter. Although they have not used RNNs, their data set is also used in this paper to assess the performance of RNNs. In [48], a deep NN, trained by an MPC policy search, is used as a policy learner to control a quadrotor. The network generates one-step policies and has a feedforward architecture. In [49], a few NN architectures, such as MLPs, RMLPs, long-short-term memory (LSTM), and gated recurrent unit cells, are compared against each other in single-step prediction of a few small robotic data sets. In [50], a hybrid of recurrent and feedforward architectures is used to learn the latent features for MPC of a robotic end-effector to cut 20 types of fruits and vegetables. Although the authors use recurrent structure, they also state that using their *transforming recurrent units* (TRUs) in a multistep prediction scheme results in "instability in the predicted values," so they use their proposed network as a one-step predictor. However, the recurrent latent state helps to improve the predictions.

III. Problem Formulation

In this section, the multistep prediction problem is defined for a general nonlinear dynamic system. As discussed earlier, regardless of the system to be modeled by RNNs, a proper state initialization is required; therefore, the state initialization problem is also defined. Note that for practical purposes, we are working in the discrete-time domain.

A. Multistep Prediction With Recurrent Neural Networks

Consider a dynamic system $\mathcal{S}_n^m$ with m input and n output dimensions. The system input and output at a time instance, k, is denoted by $\mathbf{u}_k \in \mathbb{R}^m$ and $\mathbf{y}_k \in \mathbb{R}^n$, respectively. We assume that both the input and the output are measurable at all k values. Considering an input sequence of length T starting at a time instance $k_0 + 1$, $\mathbf{U}_{k_0+1,T} \in \mathbb{R}^m \times \mathbb{R}^T$

$$\mathbf{U}_{k_0+1,T} = [\mathbf{u}_{k_0+1} \quad \mathbf{u}_{k_0+2} \quad \dots \quad \mathbf{u}_{k_0+T}]. \tag{1}$$

The system response to this input is an output sequence denoted by $\mathbf{Y}_{k_0+1,T} \in \mathbb{R}^n \times \mathbb{R}^T$

$$\mathbf{Y}_{k_0+1,T} = [\mathbf{y}_{k_0+1} \quad \mathbf{y}_{k_0+2} \quad \dots \quad \mathbf{y}_{k_0+T}]. \tag{2}$$

The multistep prediction problem seeks an accurate estimate of the system output over the same time horizon, $\tilde{\mathbf{Y}}_{k_0+1,T} \in \mathbb{R}^n \times \mathbb{R}^T$

$$\tilde{\mathbf{Y}}_{k_0+1,T} = [\tilde{\mathbf{y}}_{k_0+1} \quad \tilde{\mathbf{y}}_{k_0+2} \quad \dots \quad \tilde{\mathbf{y}}_{k_0+T}] \tag{3}$$

which minimizes a sum-of-squares error (SSE) measure, over the prediction length, T

$$L = \sum_{k=k_0+1}^{k_0+T} \mathbf{e}_k^\top \mathbf{e}_k \tag{4}$$

$$\mathbf{e}_k = \mathbf{y}_k - \tilde{\mathbf{y}}_k \tag{5}$$

where L is the SSE measure and $\mathbf{e}_k$ is the prediction error at time step k.

An RNN with m input and n output, $\mathcal{R}_n^m$, is a dynamic system. At each time instance k, feeding the input element

$\mathbf{u}_k$ to the RNN, it evolves through two major steps: 1) state update and 2) output generation

$$\mathbf{x}_k(\boldsymbol{\theta}) = \mathbf{f}(\mathbf{x}_{k-1}(\boldsymbol{\theta}), \mathbf{u}_k) \tag{6a}$$
$$\tilde{\mathbf{y}}_k(\boldsymbol{\theta}) = \mathbf{g}(\mathbf{x}_k(\boldsymbol{\theta}), \mathbf{u}_k) \tag{6b}$$

where $\tilde{\mathbf{y}}_k(\boldsymbol{\theta})$ and $\mathbf{x}_k(\boldsymbol{\theta})$ are the RNN output vector and state vector, respectively, at time k. The vector $\boldsymbol{\theta} \in \mathbb{R}^q$ encompasses the network weights and q depends on the architecture of the RNN. The function $\mathbf{f}(.)$ is defined either explicitly (RMLPs) [36] or implicitly (LSTMs) [51] as are discussed in Section V. For $\mathbf{g}(.)$, however, we choose a linear map because modeling a dynamic system by RNN is a regression problem. Note that the network states and/or output may also depend on a *history* of the right-hand side values in (6).

Using RNNs to address multistep prediction problem, we seek an RNN which, given any input sequence $\mathbf{U}_{k_0+1,T}$, produces an output sequence $\tilde{\mathbf{Y}}_{k_0+1,T}(\boldsymbol{\theta})$ which minimizes the mean SSE cost over the prediction interval $[k_0+1, k_0+T]$

$$L(\boldsymbol{\theta}) = \frac{1}{T}\sum_{k=k_0+1}^{k_0+T} \mathbf{e}_k(\boldsymbol{\theta})^\top \mathbf{e}_k(\boldsymbol{\theta}) \tag{7}$$
$$\mathbf{e}_k(\boldsymbol{\theta}) = \mathbf{y}_k - \tilde{\mathbf{y}}_k(\boldsymbol{\theta}) \tag{8}$$

where $\mathbf{y}_k$ is the system output at time $k \in [k_0+1, k_0+T]$ to the input $\mathbf{u}_k \in \mathbf{U}_{k_0+1,T}$. Therefore, the solution to the multistep prediction problem is an RNN, which minimizes L for all possible input–output sequences

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}}(L(\boldsymbol{\theta})). \tag{9}$$

The optimization in (9) is practically impossible because there can be infinite input–output sequences. In practice, a data set is collected by measuring the system input and output and a numerical optimization is carried out to find a minimum of the total cost

$$\begin{aligned} L_{\text{pred}}(\boldsymbol{\theta}) &= \frac{1}{D}\sum_{i=1}^{D} L_i(\boldsymbol{\theta}) \\ &= \frac{1}{TD}\sum_{i=1}^{D}\sum_{k=k_0+1}^{k_0+T} \mathbf{e}_{i,k}(\boldsymbol{\theta})^\top \mathbf{e}_{i,k}(\boldsymbol{\theta}) \end{aligned} \tag{10}$$

where D is the data set size. The data sets employed in the optimization process are comprised of time-series samples in the form of input–output tuples

$$\mathbb{D} = \{s_i = (\mathbf{U}_i(T), \mathbf{Y}_i(T))\} \tag{11}$$

where T indicates that all samples are of the same length.

There are many tricks and tweaks to enhance RNN training in which some of them are used in this paper and will be mentioned in Section VI. For detailed discussions, refer to [17] and [16].

B. State Initialization in Recurrent Neural Networks

In the discrete domain, feedback connections require some form of a memory buffer. In an RNN, *states* are the buffered values since they determine the network output knowing the input and network functions. Depending on the feedback source, there are two types of states: 1) output states, $\mathbf{o}_k$, which encompasses feedbacks from the network output and 2) internal states, $\mathbf{s}_k$, which encompasses feedbacks from within the network. Therefore, the RNN state vector is

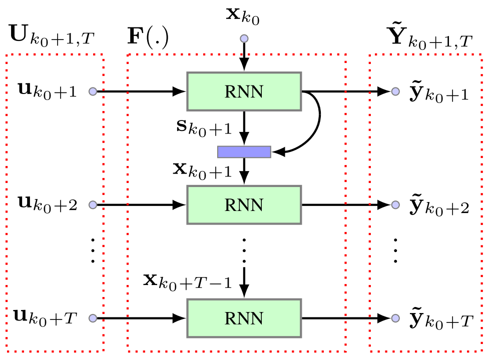

Fig. 1. RNN as a sequence-to-sequence mapper with explicit notation of the initial state [see (14)].

$$\mathbf{x}_k = \begin{bmatrix}\mathbf{o}_k \\ \mathbf{s}_k\end{bmatrix} \in \mathbb{R}^s \tag{12}$$

where s is the states count. Therefore, we can rewrite (6) using single-step buffers

$$\mathbf{o}_{k-1} = \tilde{\mathbf{y}}_{k-1} \tag{13a}$$
$$\mathbf{s}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k) \tag{13b}$$
$$\tilde{\mathbf{y}}_k = \mathbf{g}(\mathbf{x}_k, \mathbf{u}_k) \tag{13c}$$

where we have dropped $\boldsymbol{\theta}$ for the sake of brevity.

Given an initial state, $\mathbf{x}_{k_0}$, we can rewrite the RNN input–output equation as a sequence-to-sequence map

$$\tilde{\mathbf{Y}}_{k_0+1,T} = \mathbf{F}(\mathbf{x}_{k_0}, \mathbf{U}_{k_0+1,T}). \tag{14}$$

The function $\mathbf{F} : \mathbb{R}^s \times \mathbb{R}^m \times \mathbb{R}^T \to \mathbb{R}^n \times \mathbb{R}^T$ symbolizes the operations taking place sequentially inside the RNN, defined by (13). The signal that flows through the RNN, over T time steps, is shown in Fig. 1 and the elements of (14) are highlighted with the dashed boxes. Note that, in this figure, the RNN blocks represent the same network copied over time, that is, the networks' weights as well as the architectures are the same.

From (14), it is evident that the initial states play a key role in the immediate response of the RNN. Therefore, to have an accurate estimate, one should properly *initialize* the RNN, i.e., set the initial states of the RNN, $\mathbf{x}_k$, to values that (10) is minimized. Note that minimizing (10) also requires training the RNN. Therefore, RNN state initialization should be considered as a part of RNN training which should be repeatable in the testing (generalization tests) as well. We will explain this insight in more detail in Section IV.

IV. Solutions to RNN State Initialization Problem

In the context of dynamic system modeling with RNNs, the function that produces the network output, i.e., $\mathbf{g}(.)$ in (13), is a linear mapping. For a prediction generated at the time instance k, the output is

$$\tilde{\mathbf{y}}_k = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \tag{15}$$

where $\mathbf{A} \in \mathbb{R}^n \times \mathbb{R}^s$ and $\mathbf{B} \in \mathbb{R}^n \times \mathbb{R}^m$ are the output layer weights and their elements are included in the weight vector $\boldsymbol{\theta}$; hence, we have dropped $\boldsymbol{\theta}$. Using (12) to expand (15) and letting $k = k_0$, we have

$$\tilde{\mathbf{y}}_{k_0} = \mathbf{A}_s \mathbf{s}_{k_0} + \mathbf{A}_o \mathbf{o}_{k_0} + \mathbf{B}\mathbf{u}_{k_0} \tag{16a}$$
$$\mathbf{A} = [\mathbf{A}_s \;\; \mathbf{A}_o]. \tag{16b}$$

According to (13), at each time instance $k \in [k_0 + 1, k_0 + T]$, the states, $\mathbf{x}_k$, must be updated prior to generating the output, $\tilde{\mathbf{y}}_k$, which requires the knowledge of $\mathbf{x}(k-1)$. Based on the universal approximation property, there exists an RNN* which can approximate the system to be modeled with $\boldsymbol{\epsilon}_k$ error accuracy, where $\boldsymbol{\epsilon}_k$ is the prediction error at time k and can be decreased infinitesimally for all k [31]. Therefore, (16) for RNN* becomes

$$\tilde{\mathbf{y}}^*_{k_0} = \mathbf{A}^*_s \mathbf{s}^*_{k_0} + \mathbf{A}^*_o \mathbf{o}^*_{k_0} + \mathbf{B}^*\mathbf{u}_{k_0} \tag{17}$$

and using the prediction error $\boldsymbol{\epsilon}_{k_0}$

$$\mathbf{y}_{k_0} + \boldsymbol{\epsilon}_{k_0} = \mathbf{A}^*_s \mathbf{s}^*_{k_0} + \mathbf{A}^*_o \mathbf{o}_{k_0-1} + \mathbf{A}^*_o \boldsymbol{\epsilon}_{k_0-1} + \mathbf{B}^*\mathbf{u}_{k_0}. \tag{18}$$

Letting $\boldsymbol{\epsilon}_k \to \mathbf{0}$ for all k

$$\mathbf{A}^*_s \mathbf{s}^*_{k_0-1} \approx \mathbf{c}^* \tag{19}$$

where

$$\mathbf{c}^* = \mathbf{y}_{k_0} - \mathbf{A}^*_o \mathbf{y}_{k_0-1} - \mathbf{B}^*\mathbf{u}_{k_0}.$$

Note that the weights are known at the time of state initialization. However, RNN* is not necessarily known. Therefore, during training an RNN, the state initialization problem for system identification boils down to minimizing the following cost:

$$L_{\text{si}} = |\mathbf{A}_s \mathbf{s}_{k_0} - \mathbf{c}| \tag{20}$$

with respect to $\mathbf{s}_{k_0}$ subject to $a \le \mathbf{s}_{k_0} \le b$. The constraints should enforce the initial states to remain within the range of the function which generates the hidden states. For example, if the states are generated by tanh(.), then $a = -1, b = +1$. Ideally, we want $L_{\text{si}} = 0$; however, since the initialization has to be carried out in both during training and testing phases, the exact solution may lead to overfitting as will be described later in this section.

In this section, three methods of state initialization in RNNs are studied. The first method, which is also referred to by the term *washout*, is described in [17] and perhaps is the most commonly used method. The second method is based on optimizing the initial states along with the network weights [52]. The third method employs NNs and has been proposed previously by Mohajerin *et al.* [19] and is expanded here.

A. RNN State Initialization: Washout

The washout method is based on the idea that running an RNN for a period of time steps attenuates (washes out) the effect of the initial state values. Therefore, one can set the initial states to zero [17], or a random value [16], and run the RNN for a length of time until the effect of the initial values wears off.

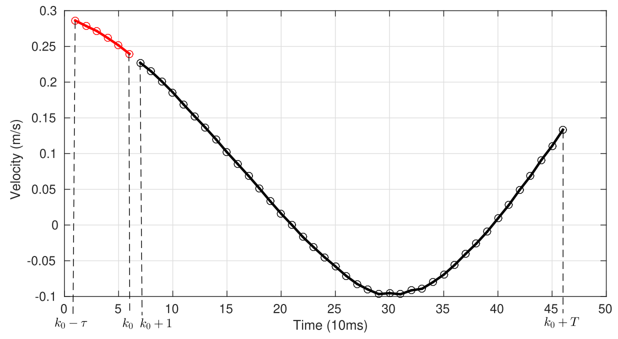

Fig. 2. Dividing a data sample into initialization (red line) and prediction (black line) segments. Each small circle is one measurement from the continuous signal. In this figure, $\tau = 6$ and $T = 40$.

Since there is no deterministic approach to obtain the washout period, it has to be treated as a hyperparameter and reduces the speed of the learning process. Additionally, since RNNs are dynamic systems, during the training process, they may temporarily experience instability. While over the entire learning process the stability of RNNs is chained to the stability of the system being learned, running an unstable instance of an RNN during training may result in extremely large cost values, which cause the learning curve to diverge. To avoid such "blowups," extra measures seem necessary. In this paper, we employ washout as it is stated in [17].

B. RNN State Initialization: Optimization

Another proposed approach to initialize an RNN is to augment the initial states $\mathbf{s}_k$ to the weight vector $\boldsymbol{\theta}$ and carry out the learning process with respect to the augmented weight vector [52]. Although this approach may address the state initialization problem during the training phase, it does not provide a mechanism to generate the initial state values in general, after the training is finished. For special cases (for instance, in [52]), it may be possible to obtain the initial hidden state values by running an architecture or problem specific optimization process. Since a general approach that is applicable to any RNN regardless of the architecture is more appealing, this method is not desirable for our purposes.

C. RNN State Initialization: NN-Based

Consider the ideal RNN, RNN*, where the network output differs from the desired output infinitesimally. We can write

$$\mathbf{o}_k = \mathbf{y}^*_{k-1} \approx \mathbf{y}_{k-1} \tag{21a}$$
$$\mathbf{s}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k). \tag{21b}$$

Equation (21) governs the dynamics of the RNN* *states*. To approximate this mapping, it is possible to employ NNs. In [19], we have proposed an auxiliary FFNN to produce the RNN *initial state values*, receiving a short history of the system input and output. To avoid confusion, the auxiliary network will be referred to as the *initializer* and the RNN which performs the prediction as the *predictor*.

The idea is to divide the data samples into two segments; the first segment is used as the input to the initializer, which

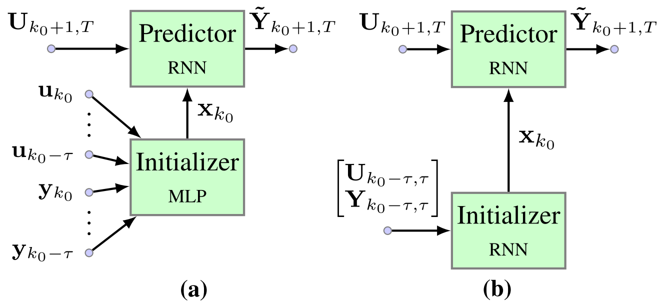

Fig. 3. Two proposed initializer-predictor pairs for multistep prediction. (a) MLP initializer. (b) RNN initializer.

initializes the predictor states, and the second one is used to train the whole network, i.e., the initializer-predictor pair. The number of steps in the prediction and initialization segment will be referred to as the prediction and initialization length and denoted by T and τ, respectively, as shown in Fig. 2. The total length of the training sample is, therefore, $T_{\text{tot}} = \tau + T$.

The desired values for the output of the initializer network, i.e., the initial RNN state values, are unknown. However, (20) proposes a penalty on the initializer network output. Therefore, the initializer-predictor pair will be trained on the following cost:

$$L_{\text{tot}} = \alpha L_{\text{pred}} + \beta L_{\text{si}} \tag{22}$$

where the prediction errors, L_{pred} and L_{si}, are defined in (10) and (20), respectively, and the coefficients α and β can be used to balance between the two costs. Without loss of generality, α and β are set to one in this paper. Nevertheless, they can be treated as hyperparameters and tuned to achieve desired performance, if necessary.

1) MLP Initializer Network: An MLP can be employed as the initializer network, which receives a history of the measurements from the system and produces the predictor initial states

$$\mathbf{x}_{k_0} = \boldsymbol{\zeta}(\mathbf{u}_{k_0-\tau}, \mathbf{u}_{k_0-\tau+1}, \ldots, \mathbf{u}_{k_0}, \mathbf{y}_{k_0-\tau}, \mathbf{y}_{k_0-\tau+1}, \ldots, \mathbf{y}_{k_0}). \tag{23}$$

In Fig. 3(a), the block diagram of this type of the initializer-predictor pair is illustrated. The underlying assumption in this approach is that the dynamics of the RNN states, defined in (21), over a fixed period (i.e., the initialization length) can be approximated by a static function. The initializer network approximates that function.

2) Recurrent Initializer Network: Since the RNN states also possess dynamic, it is also viable to employ an RNN to model their dynamic. An RNN for initialization purpose can be a sequence-to-sequence model, $\boldsymbol{\xi}(.)$, which sequentially receives the system measurement history over the initialization length, τ, and produces an output sequence, $\mathbf{X}_{k_0-\tau,\tau}$

$$\mathbf{X}_{k_0-\tau,\tau} = \boldsymbol{\xi}(\mathbf{U}_{k_0-\tau,\tau}, \mathbf{Y}_{k_0-\tau,\tau}). \tag{24}$$

However, only the last element of the output sequence of the initializer network, $\mathbf{x}_{k_0}$, is used

$$\mathbf{X}_{k_0-\tau,\tau} = [\mathbf{x}_{k_0-\tau} \quad \mathbf{x}_{k_0-\tau+1} \quad \ldots \quad \mathbf{x}_{k_0}]. \tag{25}$$

Fig. 3(b) shows the RNN–RNN initializer-predictor pair. The initializer RNN states are set to zero. Clearly, the length of the initialization segment should be long enough to capture the dynamics of the predictor states.

In the training process, the two networks are trained together, meaning that their weights are augmented and the gradient of the total cost in (22) is calculated with respect to the augmented weight vector.

V. Model Architectures

Two classes of models are considered. The first class implements RNN-based black-box models. The high-level architectures of the black-box models are shown in Fig. 3. The predictors in this figure can either be a multilayer-fully-connected (MLFC) RNN, which is essentially an RMLP with skip connections across the network, or LSTM [51] cells arranged in layers. The initializer networks, however, are either MLP or LSTM.

The second class implements a hybrid of the black-box model and a simple physics-based dynamic model of the system in a single end-to-end network. The physics-based model represents *a priori* knowledge about the system behavior derived from first principles, while the black-box model learns the unmodeled dynamics. The hybrid model is trained for the quadrotor vehicle in this paper.

A. Multilayer-Fully-Connected RNN

This architecture consists of sigmoid layers, which are connected in series and equipped with interlayer (skip) connections in feedforward and feedback directions. The presence of skip connections helps to attenuate the effect of the structural vanishing gradient problem. This network is previously presented and studied for multistep prediction of a quadrotor vehicle in [18], [53], and [54]. For an MLFC with L layers, each layer G^l, where $l = 1, \ldots, L$, has m^l inputs and n^l outputs (neurons). The equations governing the dynamics of G^l are

$$\begin{cases} \mathbf{x}_k^l = \mathbf{A}^l \mathbf{y}_{k-1}^l + \mathbf{B}^l \mathbf{u}_k^l + \mathbf{b}^l \\ \mathbf{y}_k^l = \mathbf{f}^l(\mathbf{x}_k^l) \end{cases} \tag{26}$$

where $\mathbf{x}_k^l \in \mathbb{R}^{n^l}$ is the layer activation level, $\mathbf{y}_k^l \in \mathbb{R}^{n^l}$ is the layer output, and $\mathbf{u}_k^l \in \mathbb{R}^{m^l}$ is the layer input, all at time step k. The matrix $\mathbf{A}^l \in \mathbb{R}^{n^l} \times \mathbb{R}^{n^l}$ is the feedback weight, $\mathbf{B}^l \in \mathbb{R}^{n^l} \times \mathbb{R}^{m^l}$ is the input weight matrix, $\mathbf{b}^l \in \mathbb{R}^{n^l}$ is a bias weight vector, and $\mathbf{f}^l(.) : \mathbb{R}^{n^l} \to \mathcal{R}_f^{n^l}$ is the layer activation function ($\mathbf{f}^l \in \mathcal{C}^\infty$). For an MLFC with L layers, the input to G^l at time k, i.e., $\mathbf{u}_k^l$ is constructed as

$$\mathbf{u}_k^l = [\mathbf{u}_k, \mathbf{y}_k^1, \ldots, \mathbf{y}_k^{l-1}, \mathbf{y}_{k-1}^{l+1}, \ldots, \mathbf{y}_{k-1}^L]. \tag{27}$$

B. Long-Short-Term-Memory RNN

LSTMs, first introduced in [51], consist of *cells* which are equipped with *gates*, and are referred to as *gated* RNNs. Gates in LSTMs are sigmoid layers which are trained to let information pass throughout the network in such a way that the gradient of the information is preserved across time. There are many versions of LSTMs [55]. The version described in [56], equipped with *peephole* connections, is used in this paper. Peephole connections are connection from the cell, $\mathbf{c}(.)$, to the gates. The equations of the LSTM we use in this paper are given by

$$\begin{aligned}\mathbf{g}^i k &= \sigma\left(\mathbf{W}_i^i \mathbf{u}_k + \mathbf{W}_i^m \mathbf{m}_{k-1} + \mathbf{W}_i^c \mathbf{c}_{k-1} + \mathbf{b}_i\right)\\ \mathbf{g}_k^f &= \sigma\left(\mathbf{W}_f^i \mathbf{u}_k + \mathbf{W}_f^m \mathbf{m}_{k-1} + \mathbf{W}_f^c \mathbf{c}_{k-1} + \mathbf{b}_f\right)\\ \mathbf{g}_k^o &= \sigma\left(\mathbf{W}_o^i \mathbf{u}_k + \mathbf{W}_o^m \mathbf{m}_{k-1} + \mathbf{W}_o^c \mathbf{c}_k + \mathbf{b}_o\right)\\ \mathbf{c}_k &= \mathbf{g}_k^i \odot \mathbf{f}\left(\mathbf{W}_c^i \mathbf{u}_k + \mathbf{W}_c^m \mathbf{m}_{k-1} + \mathbf{b}_c\right) + \mathbf{g}_k^f \odot \mathbf{c}_{k-1}\\ \mathbf{m}_k &= \mathbf{g}(\mathbf{c}_k) \odot \mathbf{g}_k^o\\ \mathbf{y}_k &= \mathbf{h}(\mathbf{W}_y \mathbf{m}_k + \mathbf{b}_y) = \mathbf{W}_y \mathbf{m}_k + \mathbf{b}_y. \end{aligned} \qquad (28)$$

In this set of equations, indices i, f, o, and c correspond to the *input gate*, *forget gate*, *output gate*, and *cell*, respectively. Gate activation functions are logistic sigmoid [$\sigma(.)$], while the cell activation function $\mathbf{g}(.)$ and the output activation function $\mathbf{h}(.)$ are chosen by the designer. Since the problem at hand is regression, the activation functions $\mathbf{h}(.)$ and $\mathbf{g}(.)$ are set to identity and tangent-hyperbolic function, respectively.

Note that in LSTMs, there are two types of states: cell states, $\mathbf{c}_k$, and hidden states, $\mathbf{m}_k$. In our initialization scheme, they both are treated similar to the hidden states of MLFCs. That is, they both are initialized by the initializer network.

C. Hybrid Model

It is possible to incorporate prior knowledge in modeling a dynamic system with RNNs to enhance both the training convergence speed and the accuracy of the predictions. In the context of system identification, white-box models are one of the most convenient ways to represent the prior knowledge. Depending on the simplifying assumptions taken in white-box modeling, as well as the intended usage for the final model, the definition of the states and input to the white-box model may vary. Therefore, the input to the white-box model is not necessarily the measured input collected in the data set. For instance, it is difficult to measure thrust acting on a quadrotor vehicle, and the dependence of the thrust on the motor speeds is complex in general. However, it is possible to approximate this dependence throughout the training process and employ a white-box model which receives thrust as input. Therefore, the first level of learning is to approximate the required input to the white-box model from the measured input, which is unsupervised in nature since it is already assumed that the required input to the white box is not measured during data set collection.

The desired values for the white-box model output, on the other hand, are partially or entirely measured during data set collection. However, since the white-box model does not capture many of the complex nonlinearities acting on the system, its output may be too inaccurate to be useful in generating predictions of the measured output. Therefore, a second level of training is deemed necessary for compensating for the error between the white-box model prediction and the actual output measurements. Fig. 4 shows our suggested hybrid model based on the two-level training discussed above. It comprises three

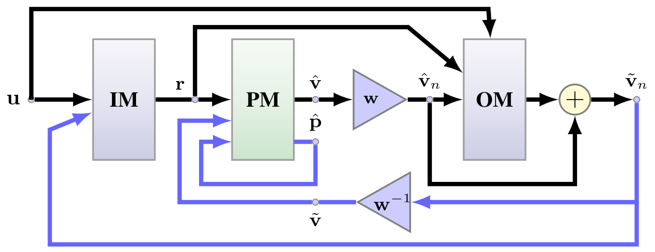

Fig. 4. Suggested method to incorporate white-box models with RNNs as black-box models (hybrid or gray box).

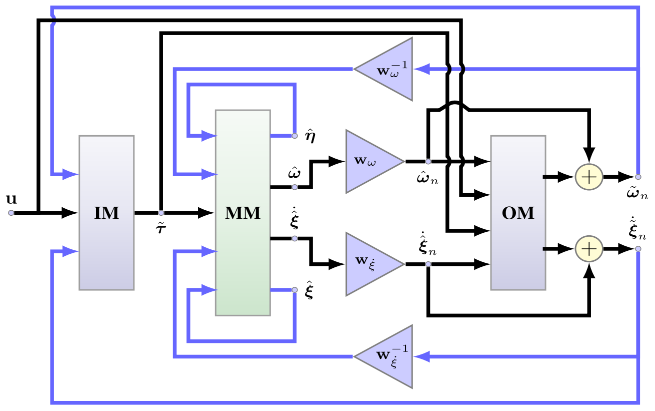

Fig. 5. Hybrid model of a quadrotor.

modules: an input model (IM), a physics model (PM), and an output model (OM). The PM module represents the white-box model, while the IM and OM modules are RNNs (with initialization networks). The PM module receives $\mathbf{r}$, generated by the IM module, as input and updates the state vector, $\mathbf{x} = [\hat{\mathbf{p}}, \hat{\mathbf{v}}]$. The vector $\hat{\mathbf{p}}$ represents the part of the states that can be updated using integrators only, e.g., position. The vector $\hat{\mathbf{v}}$ is additionally dependent on the input to the PM module. The main responsibility of the OM module is to compensate for the deviation of the $\hat{\mathbf{v}}$ vector from the output measurements. The weights denoted by $\mathbf{w}$ normalize the input to the OM module for numerical stability. The OM module generates correction to the white-box prediction, assuming that the white-box prediction error can be compensated by an additive term. The inclusion of the PM module output in the hybrid model output ensures that the PM module is actively engaged in the prediction. During the supervised training of the hybrid model, if the PM module only receives the error information through the OM module, it is possible, and highly likely, that the PM module is not effectively employed and the OM module, due to its many DOFs, learns a mapping which is too far from being a reliable and accurate model for the system [19].

For modeling the quadrotor, the PM module implements the vehicle Motion Model (MM) and, therefore, is referred to as the MM module. The MM module receives torques and thrust and updates the vehicle states, $\mathbf{x}$, which is

$$\begin{aligned}\mathbf{x}^T &= [\boldsymbol{\eta}^T \ \boldsymbol{\omega}^T \ \boldsymbol{\xi}^T \ \dot{\boldsymbol{\xi}}^T]\\ &= [\phi \ \theta \ \psi \ p \ q \ r \ x \ y \ z \ \dot{x}_I \ \dot{y}_I \ \dot{z}_I] \end{aligned} \qquad (29)$$

TABLE I
MAXIMUM VALUES FOR THE PELICAN MEASUREMENTS

$\dot{x}\ (m/s)$	$\dot{y}\ (m/s)$	$\dot{z}\ (m/s)$	$p\ (rad/s)$	$q\ (rad/s)$	$r\ (rad/s)$
3.9268	3.9721	5.8526	3.9116	3.8506	3.7902

where $\boldsymbol{\eta}$ is the Euler angles and $\boldsymbol{\omega}$ is the body angular rates in the body frame (or body rates), and $\boldsymbol{\xi}$ is the position and $\dot{\boldsymbol{\xi}}$ is the velocity both in the inertial frame. The details of the model can be found in [26], [57], and [58]. This configuration is shown in Fig. 5. Blue links represent feedback connections and black links represent feedforward connections. The gains $\mathbf{w}_\omega$ and $\mathbf{w}_{\dot{\xi}}$ are set to the maximum values of the vehicle body rates and velocity (see Table I).

VI. RESULTS AND DISCUSSION

Two data sets are used for training RNNs in the described multistep prediction problem. The first is the Stanford Helicopter data set[3] which is intended for apprenticeship learning [1]. The second data set, which is collected for this paper, encompasses various flight regimes of a quadrotor flying indoors. The Quadrotor data set is publicly available at https://github.com/wavelab/pelican_data set.

A. Quadrotor Data Set

The Quadrotor data set consists of time-series samples which are recovered from postprocessing measurements of the states of a flying quadrotor in a cubic indoor space. The total recorded flight time is approximately 3 h and 50 min, which in total corresponds to about 1.4 million time steps for each time series. The vehicle states are measured using onboard sensors as well as a precise motion capture system. The vehicle is operated by a human pilot in various flight regimes, such as hover, slight, moderate, and aggressive maneuvers. The maximum values for the six DOFs of the vehicle are listed in Table I. For more details about the data set, refer to [59].

B. Helicopter Data Set

The Helicopter data set was collected in August 2008 as a part of research for apprenticeship learning [1]. It has also been used for a single-step prediction system identification problem [2]. The flights are carried out in an outdoor environment; however, the data set does not provide a wind measurement. The flight time is approximately 55 min and there are 300k samples for each quantity. For more information about the data set, see http://heli.stanford.edu/index.html.

C. Architectures and Implementation

The predictor network can be either an MLFC, LSTM, or LSTM equipped with buffers. The buffers are called tapped delay lines (TDLs) [9]. The TDL size is 10, throughout the experiments in this paper. Each of the predictors may be initialized in one of the three fashions: washout, with an MLP initializer, or with an RNN initializer. In the case of an RNN initializer, an LSTM with one layer of LSTM cells is employed. In order to refer to each configuration, the following notation is used:

[predictor]: [number of layers] $\times$ [size of each layer] - [initializer type]: [hidden layer size] $\times$ [initialization length].

For example, an LSTM predictor with three layers, each having 200 LSTM cells initialized by an MLP with 1000 neurons in the hidden layer and an initialization length of 10, is referred to by LSTM: 3 $\times$ 200-MLP:1000 $\times$ 10.

The initialization length is 0.1 s, or 10 steps, for all of the networks trained in this paper. Two prediction lengths are considered, $T_{\text{pred}} = 0.4$ s and $T_{\text{pred}} = 1.9$ s, which correspond to 40 steps and 190 steps of prediction, respectively.

D. Evaluation Metrics

To evaluate the network prediction accuracy, the mean and distribution of the prediction error at each time step are studied. The distributions will be illustrated using box plots over the two aforementioned prediction lengths. Note that for the evaluation, the test data set is used, i.e., each sample for evaluation has not been used to train the network. The following norms are used:

$$||\bar{e}_{\dot{\xi},k}|| = \frac{1}{3}(|e_{\dot{x}_I,k}| + |e_{\dot{y}_I,k}| + |e_{\dot{z}_I,k}|)$$
$$||\bar{e}_{\omega,k}|| = \frac{1}{3}(|e_{p,k}| + |e_{q,k}| + |e_{r,k}|)$$
$$||\bar{e}_{\dot{\eta},k}|| = \frac{1}{3}(|e_{\dot{\phi},k}| + |e_{\dot{\theta},k}| + |e_{\dot{\psi},k}|) \quad (30)$$

where $\bar{e}_{\dot{\xi},k}$ is measured in meters per second (m/s) and $\bar{e}_{\omega,k}$ and $\bar{e}_{\dot{\eta},k}$ are measured in degrees per second (deg/s). The subscripts, $\dot{\xi}$, ω, and $\dot{\eta}$, correspond to the velocity, body rates, and Euler rates, respectively. Note that each error is averaged over the three perpendicular axes to give the average errors on each component of each vector. All the three prediction errors are calculated at each prediction step, k. Note that using absolute values weighs all errors equally, regardless of sign. Using other norms does not significantly alter the comparative results.

E. NN-Based Initialization Versus Washout

To compare the effect of our NN-based initialization method versus washout, simple configurations of MLFCs and LSTMs, initialized by either our method or washout, are trained and studied. To save training time, the networks are trained on three subsets of the Helicopter data set. Each data set belongs to a multi-input-single-output subsystem of the helicopter, which maps the pilot commands to the Euler rates. The following architectures are trained and compared in Fig. 6.

1) MLFC: 1 $\times$ 50 - MLP: 60 $\times$ 10.
2) MLFC: 1 $\times$ 50 - Washout: 10.
3) LSTM: 1 $\times$ 50 - MLP: 60 $\times$ 10.
4) LSTM: 1 $\times$ 50 - Washout: 10.

From Fig. 6, it is observed that the NN-based initialization method has improved both the immediate and the overall prediction accuracy. When the prediction error is long, however,

[3]Available at http://heli.stanford.edu/data set/.

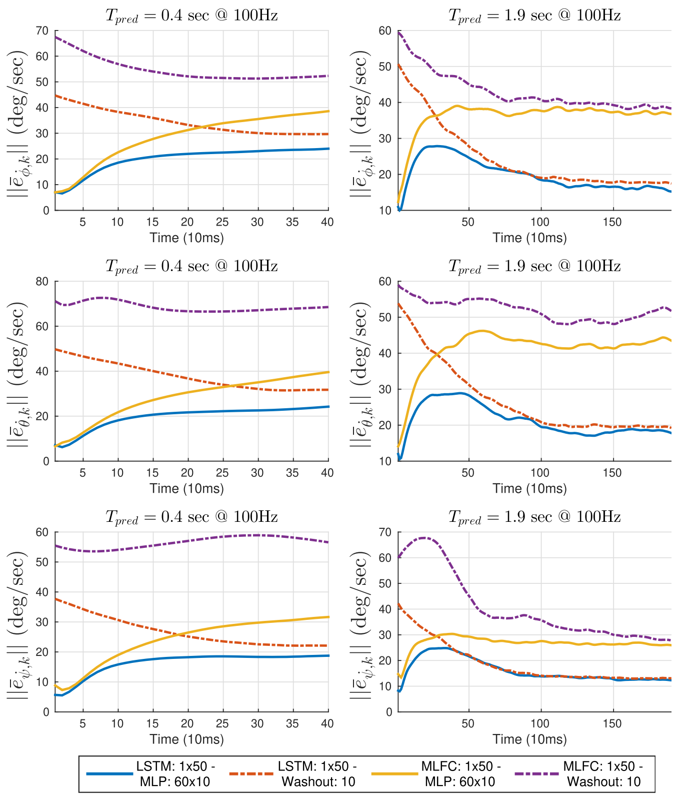

Fig. 6. Comparison between NN-based (using MLP) and washout initialization on the simplified Helicopter data set. From top to bottom: roll rate, pitch rate, and yaw rate.

the NN-based and washout-initialized RNN predictors converge to almost the same error eventually. It is also evident that an efficient washout period is difficult to determine, whereas in our NN-based initialization methods, such a problem does not exist. The results clearly show the superiority of our NN-based initialization scheme over the washout method, which is the most common method used to initialize RNNs.

F. MLFC Versus LSTM

Having established the superiority of the NN-based initialization scheme, the predictor architecture is evaluated. The following architectures are trained on the same subsets of the Helicopter data set.

1) MLFC: 1×50 - MLP: 60×10.
2) MLFC: 2×50 - MLP: 100×10.
3) MLFC: 2×100 - MLP: 200×10.
4) LSTM: 1×50 - MLP: 60×10.
5) LSTM: 2×50 - MLP: 100×10.

As a comparative measure, the following root mean sum-of-squared error (RMSSE) measure is calculated on the test data set:

$$\text{RMSSE} = \sqrt{\frac{1}{T_{\text{pred}} n_{\mathcal{G}}} \sum_{i=1}^{n_{\mathcal{G}}} \sum_{k=\tau+1}^{T_{\text{pred}}} \mathbf{e}_i^\top(k)\mathbf{e}_i(k)} \tag{31}$$

where $n_{\mathcal{G}}$ is the size of the test data set, $\mathcal{G}$. Note that other metrics, such as SSE, relative error, or relative magnitude

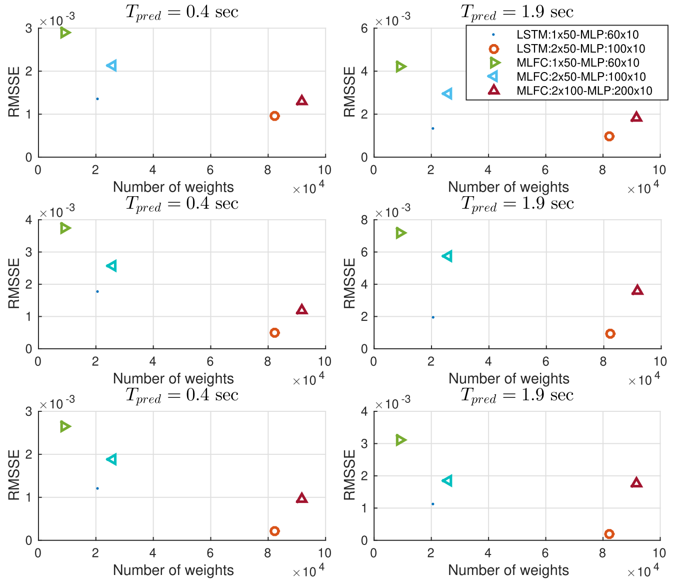

Fig. 7. Network size versus RMSSE and LSTMs versus MLFCs, helicopter-reduced data set.

error, will yield the same comparative results, however, with different values. The RMSSE is used to choose a model architecture, among those considered in this paper, with the best overall performance to further expand, train, and study in this paper.

In Fig. 7, the RMSSE measure versus the size of the networks (the number of weights) is plotted for the aforementioned architectures. In these graphs, it is observed that the LSTMs with MLP initialization outperform other methods. In fact, LSTMs with fewer weights perform better than MLFCs. As a part of hyperparameter optimization, it is a reasonable choice to conduct the remaining experiments with LSTMs initialized by the NN-based scheme.

G. MLP Versus RNN Initializers

In this section, variants of LSTM networks initialized with the two initializer networks are examined. The LSTM networks are comprised of the layers of LSTM cells connected in series. The last layer output is fed back to the first layer. For some experiments, to provide the predictor with a truncated history of the signals, TDLs are placed at the input and output of the networks. The networks map the pilot commands to the Euler rates (four inputs and three outputs). The following architectures are trained and assessed.

1) LSTM: 7×200 - MLP: $15\,000 \times 10$.
2) LSTM: 7×200 - RNN: 2500×10.
3) LSTM TDL: 7×200 - MLP: $15\,000 \times 10$.
4) LSTM TDL: 7×200 - RNN: 2500×10.

Fig. 8 shows the total RMSSE cost on the test data set for the helicopter angular rates over the previously mentioned prediction lengths. The networks are quite large; therefore, a few strategies were chosen to prevent overfitting: the network weights were initialized to tiny numbers, and a weight decay regularizer and a dropout method [60] were also employed.

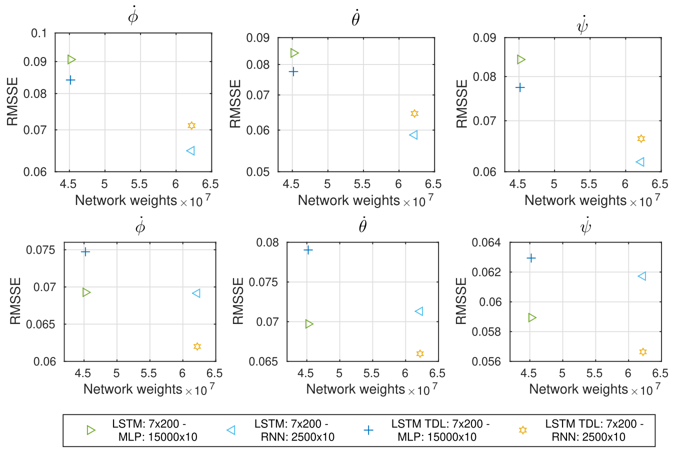

Fig. 8. Comparisons of network sizes and types on learning the helicopter angular rates from pilot commands. Top row: $T_{\text{pred}} = 0.4$ s. Bottom row: $T_{\text{pred}} = 1.9$ s.

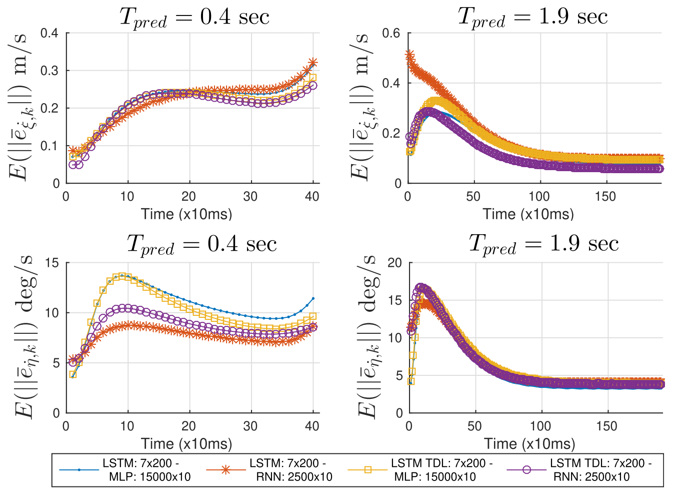

Fig. 9. Mean of the error distributions for the four black-box models of the helicopter. Top row: velocity. Bottom row: angular rates.

Since the measures are calculated over the test data set, overfitting did not occur. Based on the results illustrated in Fig. 8, it can be seen that the trained models behave similarly; therefore, we mainly focus on these four architectures.

H. Black-Box Modeling of the Helicopter

To study the reliability and accuracy of the predictions, it is best to look at the *distribution* of the prediction error, across the test data sets, throughout the prediction length.

Fig. 9 compares the mean of the error distributions over the two prediction lengths. It would have been expected that the prediction error increases monotonically throughout the prediction length. This is more or less the case for $T_{\text{pred}} = 0.4$ s. However, for longer prediction lengths, the monotonic increasing behavior is no longer observed. Instead, a peak appears at the early stages of the prediction and it is attenuated as we go forward in time. This is contrary to our expectation.

As the LSTMs are efficient in learning long-term dependences [61], the decrease in the error over late predictions might be due to noise attenuation and accumulating more information about the process by the LSTMs. Also, the peaks may be reduced if the initialization length increases. However, increasing the initialization length decreases the lengths to be used for training the predictor networks. It is also observed that the LSTMs, equipped by TDLs and initialized by RNNs, generally perform better for longer horizons, which reinforces this hypothesis. However, for the Euler rate predictions, the behavior of the mean error is not consistent. This inconsistency may be due to the size and quality of the Helicopter data set. Some of the drawbacks in using the Helicopter data set for multistep prediction are as follows.

1) The input to the networks is the pilot command and there are many levels of transformation which take place before the commands affect the helicopter motion. Time synchronization can also become very difficult to manage in such situations. To circumvent this, using actual motor speeds as the inputs is likely to mitigate the effects of delay and command transformation.
2) Considering the complex dynamics of a helicopter, the data set is relatively small. Better prediction performance is expected if more data are collected in a variety of flight regimes.
3) The helicopter is flown outdoors and is very likely affected by wind. However, no information about the wind is provided in the Helicopter data set. To obtain a predictor for the vehicle dynamic, a controlled environment is more desirable.

As we have considered the above drawbacks in collecting the Quadrotor data set, it is expected that the Quadrotor data set better suits the multistep prediction problem at hand.

I. Black-Box Modeling of the Quadrotor

The quadrotor models in this paper map a trajectory of the motor speeds and a truncated history of the system states (to initialize the RNNs) to the vehicle velocity and body angular rate vectors (or body rates). Learning velocity directly from motor speeds is difficult because of the dependence of the velocity on the vehicle attitude. A network whose output comprises of both the velocity and the body rates is difficult to train, as the network output associated with the velocity diverges during the early stages of the training and prohibits the training to converge. Therefore, as proposed in [54], the velocity and body rates are decoupled and learned separately: one network learns to predict body rates from motor speeds and the other learns to predict the velocity from motor speeds and *body rates*. As the training is offline, the actual body rates are used to train the second network, which resembles the *teacher forcing* method [33]. However, to use the two networks for multistep prediction, the first network provides the second network with the *predicted* body rates, in a cascaded architecture. We call this mode *practical*.

Fig. 10 shows the mean of the error norms, measured at each prediction step, for the aforementioned architectures, over the two prediction lengths. For the body rate prediction (the top row), the prediction accuracy on average remains better than

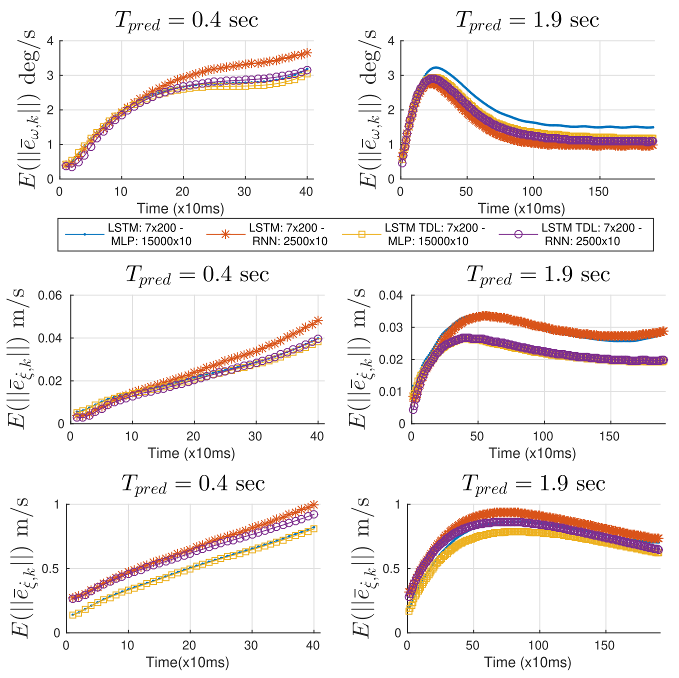

Fig. 10. Mean of the error norms for the black-box model of the quadrotor over the two prediction lengths. From top to bottom, the y-axis corresponds to body rates, velocity in the teacher force, and velocity in the practical mode.

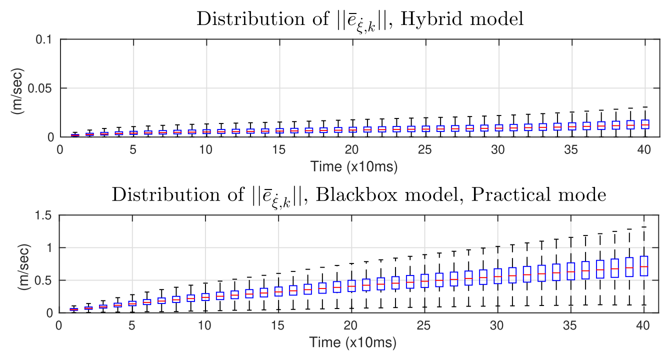

Fig. 11. Velocity error distribution of the hybrid (top) and the black-box (bottom) models ($T_{\text{pred}} = 0.4$ s). Note that the prediction by the hybrid model is over an order of magnitude better than the black box.

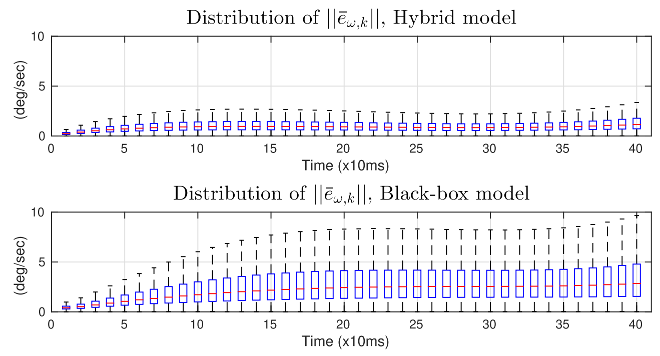

Fig. 12. Comparison of the body rate error distribution between the hybrid (top) and the black-box (bottom) models ($T_{\text{pred}} = 0.4$ s).

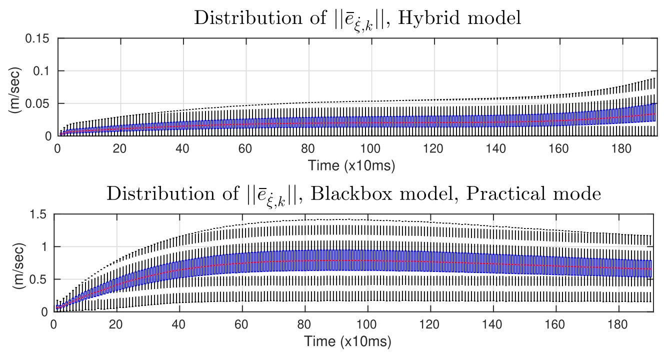

Fig. 13. Comparison of the velocity error distribution between the hybrid (top) and the black-box (bottom) models ($T_{\text{pred}} = 1.9$ s).

3.5 (deg/s) over 1.9-s prediction length ($T_{\text{pred}} = 1.9$ s). For this length, similar to the helicopter black-box model, an initial increase in the prediction error is observed. Although the error improves as the prediction proceeds, this initial increase is not favorable in a control application. The later improvement of the prediction error is due to the properties of the LSTM network as discussed for the helicopter model.

The plots in the middle row illustrate the average of the norm of the velocity prediction errors in the teacher force mode, i.e., the samples in the test data set include the measured body rates as inputs. The accuracy of the predictions on average remains better than 4 cm/s over almost 2 s. From the plots on the first and second rows, it is also observed that TDLs improve the prediction accuracy, which is consistent with our observation from the Helicopter data set. It is also observed that the LSTMs initialized with RNNs (RNN–RNN pairs) have better prediction accuracy over the longer prediction lengths, a reinforcing observation on the argument that the LSTMs efficiently employ information spread across time.

The bottom row corresponds to the velocity prediction errors in the practical mode. Comparing the teacher force mode, the velocity prediction accuracy is degraded by a factor of approximately 25. Additionally, it can be observed that the networks with the RNN initializer suffer more from the error in body rate prediction. In conclusion, the black-box model provides a reliable and accurate body rate prediction; however, the velocity prediction is far from desirable.

J. Hybrid Model of the Quadrotor

As opposed to the black-box model, the hybrid model (see Fig. 5) provides the velocity and the body rate prediction, simultaneously. The IM and OM modules in the hybrid model are initialized RNNs. The proceeding results correspond to the IM and OM modules configured as LSTM: 4×200 - MLP: 5000×10. In this section, the error distributions are studied to further assess the model prediction accuracy.

In Figs. 10–14, we compare the error distributions of the black-box model (in the practical mode) and the hybrid model for the two prediction lengths. For the shortest prediction length (0.4 s), the hybrid model reduces the body rate error by 50% and the velocity error by about 20× compared to the black-box model. For the body rate prediction over $T_{\text{pred}} = 1.9$ s, the black box initially performs worse; however, in the long run, it performs better than the hybrid model. One possible explanation for this behavior is that the MM

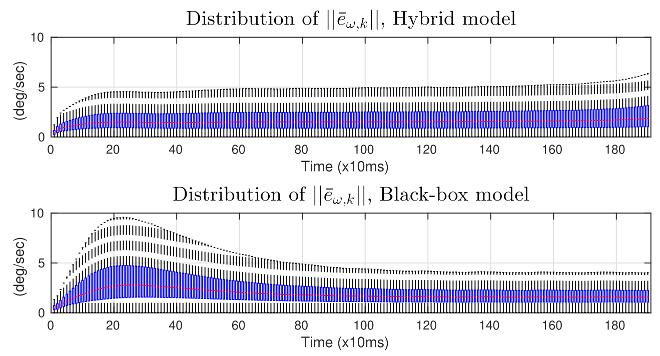

Fig. 14. Comparison of the body rate error distribution between the hybrid (top) and the black-box (bottom) models ($T_{\text{pred}} = 1.9$ s).

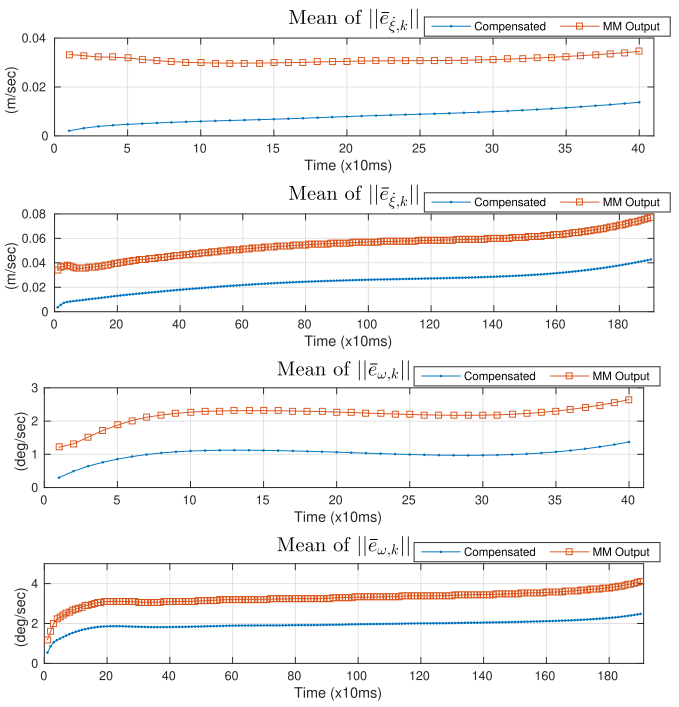

Fig. 15. Means of uncompensated output of the MM module versus the compensated output for the velocity and body rate predictions.

module in the hybrid model limits the exploration capacity of the OM module. Although the inclusion of the MM contributes significantly to the accuracy of the early prediction steps, it may also impose restrictions on the search for the optimal weights. However, the LSTMs in the black-box model are free to explore the entire weight space and they can accumulate relevant information over longer time to perform better. For the velocity, we see that the black box still performs worse; however, a slight decrease is seen on the later predictions, which is similar to the behavior in the body rate prediction.

In Fig. 15, the mean of the prediction errors is illustrated before compensation (the MM output) and after, for each prediction length. This figure shows the importance of RNN network initialization. The plots show that the MM module prediction error starts from a nonzero value, which requires the OM module to immediately compensate for that. Therefore, the output of the OM module should start from a nonzero value, which requires a proper state initialization for the RNNs. Note that at each prediction step, the compensated states are fed back into the MM module.

VII. Conclusion

In this paper, the importance of the RNN state initialization is explained and the methods to address the state initialization problem are discussed. A cascaded architecture is proposed and studied to initialize RNNs for multistep prediction. It is shown that the cascaded architecture significantly improves the prediction performance comparing the RNNs initialized with the commonly used washout method. A variety of RNN architectures, initialized with our proposed state initialization method, are trained as black-box models on learning the dynamics of the two aerial vehicles, and their performances are studied. To overcome drawbacks from a pure black-box model, a simplified model of the quadrotor motion is embedded with RNNs in a hybrid model. As a result, the accuracy of the trained hybrid model for multistep prediction is significantly improved. The distribution of the multistep prediction errors from the trained models is illustrated, which show that reliable and accurate predictions are possible with the proposed methods in this paper. The prediction provided by the hybrid model can be effectively employed in a variety of control schemes. We hope that the provided study and experiments in this paper illustrate a number of efficient ways to benefit from the ever rising power of ML methods in modeling and control of robotic systems.

References

[1] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *Int. J. Robot. Res.*, vol. 29, no. 13, pp. 1608–1639, 2010.
[2] A. Punjani and P. Abbeel, "Deep learning helicopter dynamics models," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2015, pp. 3223–3230.
[3] A. G. Parlos, O. T. Rais, and A. F. Atiya, "Multi-step-ahead prediction using dynamic recurrent neural networks," *Neural Netw.*, vol. 13, no. 7, pp. 765–786, Sep. 2000.
[4] R. Boné and M. Crucianu, "Multi-step-ahead prediction with neural networks: A review," in *Proc. de l'équipe RFAI, 9èmes Rencontres Internationales: Approches Connexionnistes en Sci. Économiques et en Gestion*, Boulogne-sur-Mer, France, vol. 2, Nov. 2002, pp. 97–106.
[5] J. Maciejowski, *Predictive Control With Constraints*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2002.
[6] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, "Precision flight control for a multi-vehicle quadrotor helicopter testbed," *Control Eng. Pract.*, vol. 19, no. 9, pp. 1023–1036, Sep. 2011.
[7] R. R. Yager and L. A. Zadeh, *An Introduction to Fuzzy Logic Applications in Intelligent Systems*, vol. 165. New York, NY, USA: Springer, 2012.
[8] J. J. E. Oviedo, J. P. Vandewalle, and V. P. L. Wertz, *Fuzzy Logic, Identification and Predictive Control*. London, U.K.: Springer-Verlag, 2006.
[9] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1999.
[10] O. Nelles, *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. Berlin, Germany: Springer-Verlag, 2013.
[11] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.
[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[13] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[14] V. Badrinarayanan, A. Kendall, and R. Cipolla. (2015). "SegNet: A deep convolutional encoder-decoder architecture for image segmentation." [Online]. Available: https://arxiv.org/abs/1511.00561

[15] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proc. AAAI*, 2017, pp. 4278–4284.

[16] H.-G. Zimmermann, C. Tietz, and R. Grothmann, "Forecasting with recurrent neural networks: 12 tricks," in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 2012, pp. 687–707.

[17] H. Jaeger, "A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the 'echo state network' approach," GMD-Forschungszentrum Informationstechnik, Sankt Augustin, Germany, Tech. Rep. 159, 2002, vol. 5.

[18] N. Mohajerin and S. L. Waslander, "State initialization for recurrent neural network modeling of time-series data," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2330–2337.

[19] N. Mohajerin, M. Mozifian, and S. L. Waslander, "Deep learning a quadrotor dynamic model for multi-step prediction," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 2454–2459.

[20] V. A. Akpan and G. D. Hassapis, "Nonlinear model identification and adaptive model predictive control using neural networks," *ISA Trans.*, vol. 50, no. 2, pp. 177–194, Apr. 2011.

[21] O. Nerrand, P. Roussel-Ragot, D. Urbani, L. Personnaz, and G. Dreyfus, "Training recurrent neural networks: Why and how? An illustration in dynamical process modeling," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 178–184, Mar. 1994.

[22] A. Delgado, C. Kambhampati, and K. Warwick, "Dynamic recurrent neural network for system identification and control," *IEE Proc. - Control Theory Appl.*, vol. 142, no. 4, pp. 307–314, Jul. 1995.

[23] I. Baruch and C. R. Mariaca-Gaspar, "A Levenberg–Marquardt learning applied for recurrent neural identification and control of a wastewater treatment bioprocess," *Int. J. Intell. Syst.*, vol. 24, no. 11, pp. 1094–1114, 2009.

[24] J. Atuonwu, Y. Cao, G. P. Rangaiah, and M. O. Tadé, "Identification and predictive control of a multistage evaporator," *Control Eng. Pract.*, vol. 18, no. 12, pp. 1418–1428, Dec. 2010.

[25] R. Al Seyab and Y. Cao, "Nonlinear system identification for predictive control using continuous time recurrent neural networks and automatic differentiation," *J. Process Control*, vol. 18, no. 6, pp. 568–581, Oct. 2007.

[26] T. Madani and A. Benallegue, "Adaptive control via backstepping technique and neural networks of a quadrotor helicopter," in *Proc. 17th World Congr. Int. Fed. Autom. Control*, 2008, pp. 6513–6518.

[27] H. Boudjedir, O. Bouhali, and N. Rizoug, "Neural network control based on adaptive observer for quadrotor helicopter," *Int. J. Inf. Technol., Control Autom.*, vol. 2, no. 3, pp. 39–54, 2012.

[28] T. Dierks and S. Jagannathan, "Output feedback control of a quadrotor UAV using neural networks," *IEEE Trans. Neural Netw.*, vol. 21, no. 1, pp. 50–66, Jan. 2010.

[29] C. Nicol, C. J. B. Macnab, and A. Ramirez-Serrano, "Robust adaptive control of a quadrotor helicopter," *Mechatronics*, vol. 21, no. 6, pp. 927–938, 2011.

[30] K.-I. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural Netw.*, vol. 6, no. 6, pp. 801–806, 1993.

[31] L. Jin, P. N. Nikiforuk, and M. M. Gupta, "Approximation of discrete-time state-space trajectories using dynamic recurrent neural networks," *IEEE Trans. Autom. Control*, vol. 40, no. 7, pp. 1266–1270, Jul. 1995.

[32] A. M. Schäfer and H. G. Zimmermann, "Recurrent neural networks are universal approximators," in *Artificial Neural Networks—ICANN* (Lecture Notes in Computer Science), vol. 4131, S. D. Kollias, A. Stafylopatis, W. Duch, and E. Oja, Eds. Berlin, Germany: Springer, 2006.

[33] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, Mar. 1990.

[34] S. Chen, S. A. Billings, and P. M. Grant, "Non-linear system identification using neural networks," *Int. J. Control*, vol. 51, no. 6, pp. 1191–1214, 1990.

[35] N. Mohajerin, J. Histon, R. Dizaji, and S. L. Waslander, "Feature extraction and radar track classification for detecting UAVs in civilian airspace," in *Proc. IEEE Radar Conf. (RadarCon)*, Cincinnati, OH, USA, May 2014, pp. 674–679.

[36] J. F. Kolen and S. C. Kremer, Eds., *A Field Guide to Dynamical Recurrent Networks*. New York, NY, USA: IEEE Press, 2001.

[37] A. G. Parlos, K. T. Chong, and A. F. Atiya, "Application of the recurrent multilayer perceptron in modeling complex process dynamics," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 255–266, Mar. 1994.

[38] X. Li and W. Yu, "Dynamic system identification via recurrent multilayer perceptrons," *Inf. Sci.*, vol. 147, nos. 1–4, pp. 45–63, 2002.

[39] S. Li, "Wind power prediction using recurrent multilayer perceptron neural networks," in *Proc. Power Eng. Soc. General Meeting*, vol. 4, Jul. 2003, pp. 2325–2330.

[40] M. V. Kumar, S. N. Omkar, R. Ganguli, P. Sampath, and S. Suresh, "Identification of helicopter dynamics using recurrent neural networks and flight data," *J. Amer. Helicopter Soc.*, vol. 51, no. 2, pp. 164–174, 2006.

[41] H. T. Siegelmann, B. G. Horne, and C. L. Giles, "Computational capabilities of recurrent NARX neural networks," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 27, no. 2, pp. 208–215, Apr. 1997.

[42] T. Lin, B. G. Horne, P. Tiňo, and C. L. Giles, "Learning long-term dependencies in NARX recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 7, no. 6, pp. 1329–1338, Nov. 1996.

[43] M. Basso, L. Giarre, S. Groppi, and G. Zappa, "NARX models of an industrial power plant gas turbine," *IEEE Trans. Control Syst. Technol.*, vol. 13, no. 4, pp. 599–604, Jul. 2005.

[44] S. R. Anderson, N. F. Lepora, J. Porrill, and P. Dean, "Nonlinear dynamic modeling of isometric force production in primate eye muscle," *IEEE Trans. Biomed. Eng.*, vol. 57, no. 7, pp. 1554–1567, Jul. 2010.

[45] Z. Li, M. Hayashibe, C. Fattal, and D. Guiraud, "Muscle fatigue tracking with evoked EMG via recurrent neural network: Toward personalized neuroprosthetics," *IEEE Comput. Intell. Mag.*, vol. 9, no. 2, pp. 38–46, May 2014.

[46] J. Wu, H. Peng, Q. Chen, and X. Peng, "Modeling and control approach to a distinctive quadrotor helicopter," *ISA Trans.*, vol. 53, no. 1, pp. 173–185, 2014.

[47] Z. Taha, A. Deboucha, and M. Bin Dahari, "Small-scale helicopter system identification model using recurrent neural networks," in *Proc. TENCON IEEE Region 10 Conf.*, Nov. 2010, pp. 1393–1397.

[48] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 528–535.

[49] O. Ogunmolu, X. Gu, S. Jiang, and N. Gans. (2016). "Nonlinear systems identification using deep dynamic neural networks." [Online]. Available: https://arxiv.org/abs/1610.01439

[50] I. Lenz, R. A. Knepper, and A. Saxena, "DeepMPC: Learning deep latent features for model predictive control," in *Robotics: Science and Systems*. Rome, Italy, 2015. [Online]. Available: http://www.roboticsproceedings.org/rss11/p12.html, doi: 10.15607/RSS.2015.XI.012.

[51] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[52] V. M. Becerra, J. M. F. Calado, P. M. Silva, and F. Garces, "System identification using dynamic neural networks: Training and initialization aspects," *IFAC Proc. Vols.*, vol. 35, no. 1, pp. 235–240, 2002.

[53] N. Mohajerin and S. L. Waslander, "Modular deep recurrent neural network: Application to quadrotors," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Oct. 2014, pp. 1374–1379.

[54] N. Mohajerin and S. L. Waslander, "Modelling a quadrotor vehicle using a modular deep recurrent neural network," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, Oct. 2015, pp. 376–381.

[55] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space Odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[56] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proc. 15th Annu. Conf. Int. Speech Commun. Assoc.*, 2014, pp. 338–342.

[57] H. Voos, "Nonlinear control of a quadrotor micro-UAV using feedback-linearization," in *Proc. IEEE Int. Conf. Mechatronics (ICM)*, Apr. 2009, pp. 1–6.

[58] A. Freddi, A. Lanzon, and S. Longhi, "A feedback linearization approach to fault tolerance in quadrotor vehicles," in *Proc. IFAC World Congr.*, Milan, Italy, 2011, pp. 5413–5418.

[59] N. Mohajerin, "Modeling dynamic systems for multi-step prediction with recurrent neural networks," Ph.D. dissertation, Univ. Waterloo, Waterloo, ON, Canada, 2017. [Online]. Available: http://hdl.handle.net/10012/12766

[60] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[61] S. Hochreiter *et al.*, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Neural Networks*. Piscataway, NJ, USA: IEEE Press, 2001.

Nima Mohajerin (M'08) received the B.Sc.E. degree in electrical engineering from Shahid Beheshti University, Tehran, Iran, in 2003, the M.Sc. degree in control engineering from the Amirkabir University of Technology, Tehran, in 2006, the M.Sc. degree in computer science with a focus on robotics and intelligent systems from Örebro University, Örebro, Sweden, in 2011, and the Ph.D. degree from the Mechatronics and Mechanical Engineering Department, University of Waterloo, Waterloo, ON, Canada, in 2017, under the supervision of Prof. S. L. Waslander.

Since graduation, he has been a full-time Machine Learning Research Scientist with Huawei Technologies Canada Inc., Markham, ON, Canada. He is an Active Researcher in deep learning, robotics and autonomous ground, and aerial vehicles. He contributed significantly to the industrial research and development in diagnostic systems for domestic automobile industry. In parallel, he focused his research on fuzzy logic controllers and reinforcement learning during his M.Sc. degree.

Steven L. Waslander (M'08) received the B.Sc.E. degree from Queen's University, Kingston, ON, Canada, in 1998, and the M.S. and Ph.D. degrees from Stanford University in Aeronautics and Astronautics, Stanford, CA, USA, in 2002 and 2007, respectively.

As a graduate student at Stanford University in Aeronautics and Astronautics, he created the Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control, the world's most capable outdoor multi-vehicle quadrotor platform at the time. He was a Control Systems Analyst for Pratt & Whitney Canada, Longueuil, QC, Canada, from 1998 to 2001. He was recruited to the University of Waterloo, Waterloo, ON, Canada, in 2008, and moved to the University of Toronto, Toronto, ON, Canada, in 2018, where he founded and directs the Toronto Robotics and Artificial Intelligence Laboratory. He is currently an Associate Professor with the University of Toronto Institute for Aerospace Studies, Toronto, where he is a Leading Authority on autonomous aerial and ground vehicles, simultaneous localization and mapping, and multi-vehicle systems. His current research interests include the state of the art in autonomous drones and autonomous driving through advances in localization and mapping, object detection and tracking, integrated planning and control methods, and multirobot coordination.