IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 5 April 2023, accepted 9 May 2023, date of publication 19 May 2023, date of current version 30 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3277977

RESEARCH ARTICLE

One-Shot Bipedal Robot Dynamics Identification With a Reservoir-Based RNN

MICHELE FOLGHERAITER, (Member, IEEE), ASSET YSKAK, AND SHARAFATDIN YESSIRKEPOV
Department of Robotics, School of Engineering and Digital Sciences, Nazarbayev University, Astana 01000, Kazakhstan

Corresponding author: Michele Folgheraiter (michele.folgheraiter@nu.edu.kz)

This work was supported by Nazarbayev University under the Faculty Development Competitive Research Grants Program through the Project "Development of an Intrinsically Safe Actuation System With Adaptive Neuromorphic Control for Humanoid Robotics Application" under Award 021220FD0551.

ABSTRACT The nonlinear inverted pendulum model of a lightweight bipedal robot is identified in real-time using a reservoir-based Recurrent Neural Network (RNN). The adaptation occurs online, while a disturbance force is repeatedly applied to the robot body. The hyperparameters of the model, such as the number of neurons, connection sparsity, and number of neurons receiving feedback from the readout unit, were initialized to reduce the complexity of the RNN while preserving good performance. The convergence of the adaptation algorithm was numerically proved based on Lyapunov stability criteria. Results demonstrate that, by using a standard Recursive Least Squares (RLS) algorithm to adapt the network parameters, the learning process requires only few examples of the disturbance response. A Mean Squared Error (MSE) of 0.0048, on a normalized validation set, is obtained when 13 instances of the impulse response are used for training the RNN. As a comparison, a linear Auto Regressive eXogenous (ARX) model with the same number of adaptive parameters obtained a MSE of 0.0181, while a more sophisticated Neural Network Auto Regressive eXogenous model (NNARX), having ten time more adaptive parameters, reached a MSE of 0.0079. If only one example, one-shot, is used for identifying the RNN model, the MSE increases to 0.0329 while showing still good prediction capabilities. From a computational point of view, the RNN in combination with the RLS adaptation algorithm, presents a lower complexity compared with the NNARX model that uses the back propagation algorithm, which makes the reservoir-based RNN model more suitable for real-time applications.

INDEX TERMS One-shot nonlinear model identification, bipedal robot inverse pendulum model, reservoir-based RNN, echo state machine, liquid state machine, long short-term memory RNN, RNN real-time implementation.

I. INTRODUCTION

Although current humanoid robots present locomotion and manipulation skills that allow them to carry out successfully well-designed tasks [1], they still lack of flexibility and long-term autonomy. Neurobotics is an interdisciplinary field that aims at fulfilling these gaps by developing and studying robotics systems endowed with control strategies strongly inspired by the knowledge of the natural nervous systems. The methodologies and techniques used in this field will allow developing truly multipurpose machines that can better adapt and cooperate with human beings. Among the different paradigms to provide the robot with learning and adapting capabilities, RNN [2] represents a powerful model that is strongly inspired by the morphology and physiology of biological neural networks. In comparison with a classical Feed-forward Neural Network (FNNs) architecture, where the signals propagate only from the input layer toward the output layer, in a RNN the presence of directed cycles endows the system with a dynamic nonlinear temporal behavior. Although, it is also possible to model nonlinear dynamic systems, such as a humanoid robot, using a FNN receiving

The associate editor coordinating the review of this manuscript and approving it for publication was Shuihua Wang.

as input the state of the system, in many situations the full knowledge of the instantaneous state of the system may not be available. In comparison, a RNN can model the dynamics of the target system without the aforementioned constraints. Thus, RNNs exhibit attractor dynamics and short- and long-term memory. In [3] it was demonstrated that a RNN with an architecture like the one proposed by Elman [4], [5] could forecast the joints torque of a manipulator receiving as input only the desired position of the joints. Another possible architecture is the one based on the combination of a FNN and a set of tapped delay lines [6]. This belongs to the class of Wiener models, where a dynamic linear system is followed by a memory-less non-linear one. Depending on whether the model uses the real process output or its prediction, we have two possible configurations: the Neural Network Autoregressive with eXogeneous input architecture (NNARX) and Neural Network Output Error (NNOE) architecture [7]. The main advantage of the first configuration is that its behavior is always stable since the model output is a pure algebraic function of the process inputs, while in the second case the model itself resemble a dynamic system and therefore its stability needs to be assessed.

Among the possible RNN architectures, the Long Short-Term Memory (LSTM) recurrent neural networks have been proven to have remarkable learning capabilities especially when used within deep neural structures. Due to the presence of different layers capable to extract features with increasing complexity, they are suitable to elaborate and abstract the information at different levels, avoiding the gradient vanishing problem that affects the back propagation algorithm [8] and implementing both short and long term memory mechanisms [9].

Although there are many applications of LSTM RNNs in the fields of speech, image and video processing, for a review see [10], their suitability for systems identification still needs to be fully explored [11]. In [12] a LSTM RNN is combined with a FNN in order to identify in real-time the power fluctuations caused by renewable energy. In [11] a combination of n LSTM models is realized using a convex coefficient that is updated using the back propagation algorithm. By adding a tapped delay input line, a LSTM can be used to model nonlinear electronic components [13]. Nonlinear rolling friction can be predicted from the joint position by a RNN using LSTM [14], [15], where a mini-batch training scheme is used. In [16] a LSTM-based analysis method is used to rapidly identify low frequency modal features in a power system, while in [17] a thermal field model is obtained using a LSTM to simulate the crystal growth process. In this case, the past measured outputs of the process are provided together with the inputs to the LSTM model.

Although these applications demonstrated a remarkable accuracy of the LSTM RNN, the main limitation of using such architecture for system identification is represented by the fact that the RNN is trained offline using a batch learning setup. Indeed the algorithms necessary to train the RNNs, such as Back-Propagation Through Time (BPTT) [18], Real-Time Recurrent Learning (RTRL) [19], Extended Kalman Filter (EKF) [20] and LSTM [21], are computationally expensive. BPTT and LSTM are $O(T * N^2)$, where T represents the dimension of the training set, RTRL is $O(N^4)$ and EKF is $O(N^2)$ in the case the RNN has only one output.

The computational complexity and the need for extensive theoretical background have in the past limited the practical application of RNNs for online identification of nonlinear dynamic systems [11]. However, in more recent years, a different category of RNNs gained more attention in the scientific community, which can remove partially this issue. A reservoir-based RNN is a type of reservoir computing framework [22], also referred as Echo State Networks (ESN) [23], [24], [25] or Liquid State Machine (LSM) [26], [27], which consists of a reservoir of hundreds of neurons that are randomly and sparsely connected by static synapses. When the RNN is excited with input signals it starts to oscillate in a more or less chaotic way according to the initialization conditions. In comparison with classical RNN architectures, like the Elman's Network, Jordan's Network, State-Space Model, or the Recurrent MultiLayer Perceptron (RMLP) [5], not all the synapses of a reservoir-based RNN are trained by the learning algorithm. The adaptation, and therefore the capability to reproduce target period signals [28], delayed temporal sequences [21], nonlinear dynamic system behaviors, associative memory, etc., it is possible thanks to an output layer of readout units that performs a linear combination of the signals generated by the RNN's reservoir. Due to this simple paradigm no back propagation of the error signal is necessary, avoiding this way the known problem of gradient vanishing [8] and reducing the computational complexity of the learning algorithm. Having only one adapting layer to deal with, and a linear activation function, it is possible to employ standard linear regression methods such as the Least Square (LS) and Recursive Least Square (RLS) algorithms, which have been proven performing well in case of linear and nonlinear systems identification [29]. More bio-inspired learning techniques, namely the Exploratory Hebbian (EH) learning [30] and the p-delta rule [31], can also be used to train the RNN.

This work presents a reservoir-based RNN applied for identifying the nonlinear inverted pendulum dynamic model of a bipedal robot. The learning process requires only few examples of the input-output signals to identify the target system. Since the input-output pairs are provided to the RLS algorithm only one time (one epoch), the proposed adaptation scheme is suitable for real-time model identification applications. In comparison with other RNN architectures that require computationally expensive offline training, such as the LSTM RNN [11], [12], [13], [14], [15], [17], [32], the proposed identification scheme uses much less computational resources and can be employed in an online model-based adaptive control scheme. In this work, we further assume that it is not possible to access the full state of the system

(position and velocity in our case) and, therefore, we adopted a single input-output representation that makes the identification problem more difficult. As input variable, we consider the impulsive force that acts to destabilize the robot posture, and which information content is mainly coded in the amplitude and the time of application. From this information, the model must predict the behavior of the system, namely the instantaneous angular position of the robot's upper body. This task results particularly difficult for other model architectures such as the NNARX and LSTM RNN, since an impulsive input is not able to excite enough the Neural Network (NN). Thus, these frameworks are more suitable to represent long-term dependencies in sequential data.

The main contributions of this research work can be summarized as:

- The proposed model, consisting of a reservoir-based RNN, was applied to identify the nonlinear inverted pendulum dynamics of a bipedal robot under realistic operating conditions.
- Compared to other models, which are based on FNNs and RNNs and use computationally expensive training algorithms like BPTT, our adaptation scheme is based on the RLS method and was proven to be more accurate, and to require fewer computational resources.
- A prove for the convergence of the adaptation scheme is given by means of numerical simulations.
- Finally, it is demonstrated that a one-shot (one example) online system identification can be achieved and that the so obtained model can be executed on small and low-power consumption computational units.

The organization of this paper is as follows: Next section presents the experimental setup and the input-output data acquisition process. Section III introduces the reservoir-based RNN architecture, assesses its stability and proposes a procedure to initialize its parameters. Section IV reports the experimental results and the performances comparison with other architectures. Finally, last section concludes this work and points to possible future developments.

II. EXPERIMENTAL SETUP AND DATA ACQUISITION

The experimental setup consists of a humanoid bipedal robot developed in our laboratory, a tutor that can support the robot in case of a failure of the control system, and a Linux based PC where all the data are monitored and logged.

The robot, see Fig. 1, includes 12 Degrees of Freedoms (DOFs) actuated by Dynamixel PRO H-42-20-S300 servomotors with a nominal power and torque of 36 W and 6.3 Nm, respectively. Each leg consists of six joints, three in the hip, one in the knee and two in the ankle. The robot's structure was realized with a combination of lightweight materials such as PLA, aluminum and carbon fibers. In particular, the links and the different mechanical parts were designed and 3D printed in a way to facilitate the integration of reinforcement plates made of metal.

This allowed to build a full-size bipedal robot which is 1.1 m height and weights only 10.8 kg. The robot is

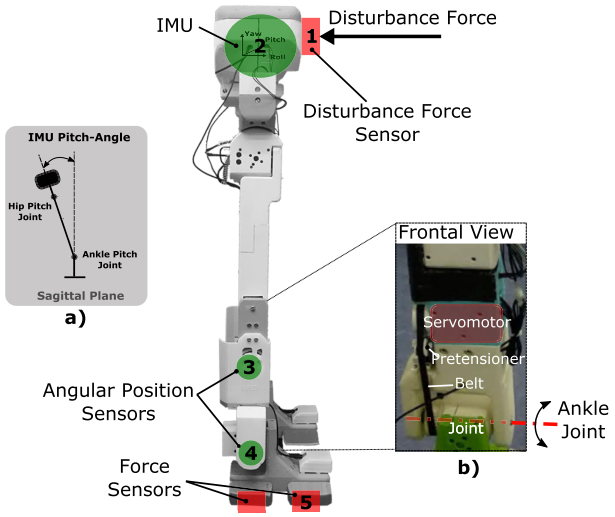

FIGURE 1. Displacement of the sensors used for the experiment in the NU-Biped robot and details of the elastic pretensioner mechanism (Frontal View).

equipped with an onboard computational unit represented by a Raspberry Pi 3 and different sensors that are interfaced with the main CPU through several micro-controllers. For more details about the robot's design and hardware please refer to [33].

Particular attention in this study should be paid to the mechanism that actuates the ankle-pitch joint (see frontal view in Fig.1-b). In this case, the joint is not directly driven by the servomotor, but through a belt equipped with a pretensioner elastic mechanism. This introduces elasticity in the joint as well as a nonlinear component due to the viscoelastic rolling friction of the synchronous belt [14].

During the experiment the robot control system was activated in order to keep the standing position. The robot was pushed rapidly and repeatedly with the hand, applying a force orthogonal to the frontal plane and ranging from 6 N to 14 N. Between two perturbations, a delay of 15 seconds was applied to allow the oscillations to subside. While performing the experiment different quantities were measured. Fig. 1 illustrates the displacement of the contact force sensor (1) that measures the orthogonal component of the disturbance force, the Inertial Measurement Unit (IMU) sensor (2) that measures the orientation of the robot, the magnetic encoder (3) that measures the angular position of the servomotor that controls the ankle pitch joint, the magnetic encoder AS5048A (4) that measures the angular position of the ankle pitch joint, and the force sensors (5), four under each foot, that are used to estimate the Center of Pressure (CoP) of the feet while in contact with the floor. It is to notice that the position of the ankle joint is measured independently from the position of the motor that actuate it. This because they are connected through the synchronous belt. The elastic component presents in the transmission mechanism is useful to absorb part of the impact forces that have a torque component about the pitch-ankle joint. Through a differential measurement of the

two magnetic encoders, it is also possible to estimate the extension of the belt and therefore the tension applied to it.

Although all the available sensory quantities were acquired during the experiment, for this study, we used only the measurement of the disturbance force and the pitch-angle (measured through the IMU), see Fig. 1-a. In addition, since different communication protocols (see Fig. 2) were used to allow the sensors and the computational units to communicate, it was not possible to sample all the signals with the same frequency. Thus, a re-sampling and interpolation of the signals was necessary.

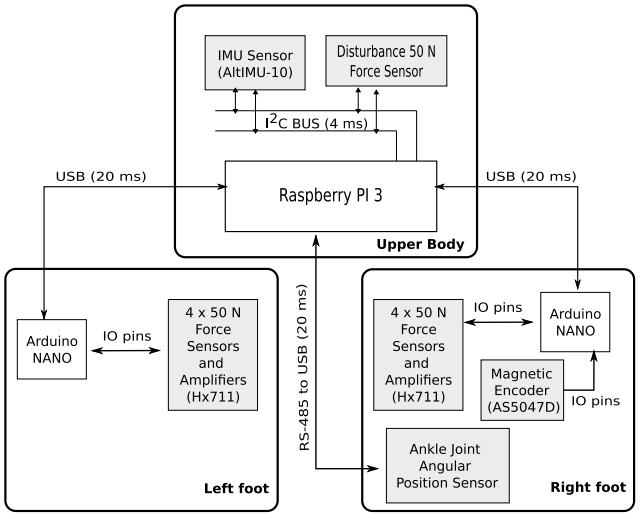

FIGURE 2. Communication protocols used to acquire the sensory data.

We conducted an experiment that lasted 267 seconds, during which we applied a total of 21 perturbations to the robot. A data set was recorded that includes the orientation of the robot, thus the pitch-angle obtained from the IMU, and the disturbance force measurement. These two signals were acquired with a sample time of 4 ms and, to allow a better model identification, they were scaled in the range [−1,+1] using Eq. 1.

$$\mathbf{x}_N = \frac{2(\mathbf{x} - min(\mathbf{x}))}{max(\mathbf{x}) - min(\mathbf{x})} - 1 \qquad (1)$$

In Fig. 3 the pitch-angle and the disturbance force signals are reported for the entire data set respectively in the upper and lower plots.

As it is possible to observe, the force signal consists of a sequence of impulses with different amplitudes, while the position consists of a sequence of oscillatory signals characteristic of the response of an under-damped second order dynamic system. This is more visible in Fig.4, where a single perturbation is reported (not normalized data).

It is important to point out that if we compare the amplitudes of the first peak of the oscillation and the amplitude of the corresponding impulse force (see Fig. 3), we can observe that they are non linearly related. As an example, the last impulse force has a maximum amplitude increased

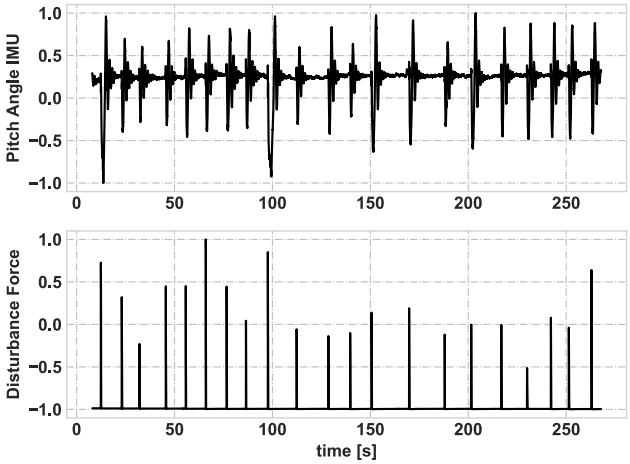

FIGURE 3. Entire data set consisting of a sequence of force impulses [N] and correspondent pitch-angle position [deg]. Signals are scaled in the range [−1, +1].

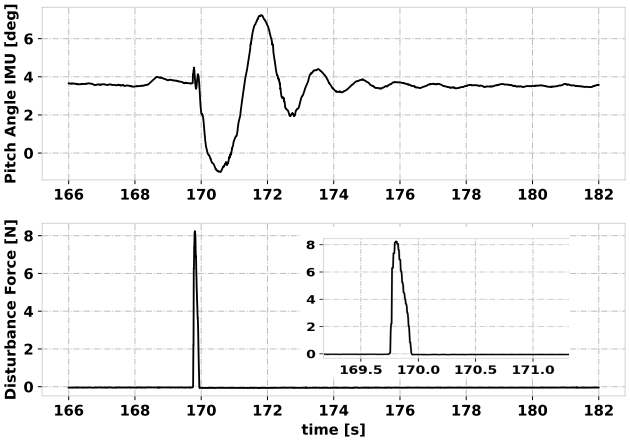

FIGURE 4. Single perturbation experiment. The response is typical of a second order nonlinear dynamic system. The force impulse last 200 *ms*, in the sub-plot at the bottom it is possible to observe its shape.

of 72 % with respect to the maximum amplitude of previous impulse. However, if we look at the pitch-angle signal, the amplitudes of the last two oscillations are comparable. This is also confirmed by considering the energy, in this case the last impulse has a correspondent energy that is 17 % higher than the energy of the previous one.

III. THE RESERVOIR-BASED RNN ARCHITECTURE

A reservoir-based RNN architecture [23], see Fig.5, consists of a set of N neurons which are randomly and sparsely connected to each other by positive and negative synapses. The neurons that form the reservoir receive additional signals from the input nodes and from the RNN outputs. In turns, the RNN outputs are computed by linear combinations of the outputs of the neurons present in the reservoir.

More specifically, the reservoir's internal connections are represented by the elements of a $[N \times N]$ *Connection Matrix* $\mathbf{W^r}$ which are initialized in the range [−1,+1] according to a normal distribution having mean $\mu = 0$ and

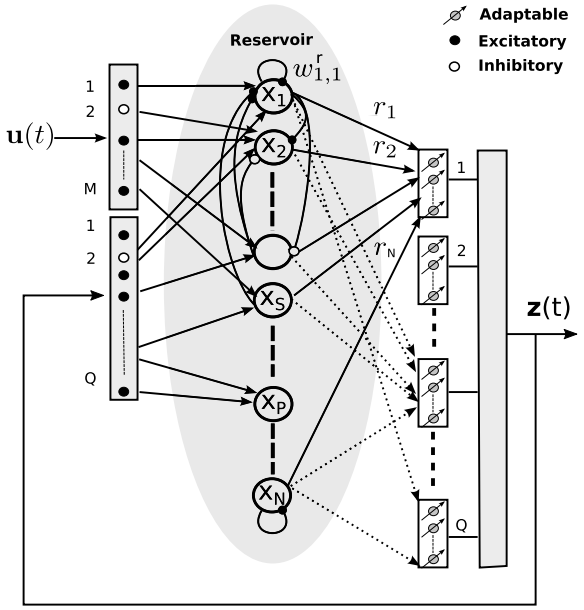

FIGURE 5. The RNN architecture consists of a reservoir of N neurons sparsely and randomly connected by excitatory and inhibitory synapses, black and white dots respectively.

variance $\sigma = \frac{C_h}{\sqrt{S_p * N}}$, where Ch is a parameter that regulates the level of chaos in the RNN and S_p represents the sparsity of the *Connection Matrix* $\mathbf{W^r}$.

The inputs of the RNN are represented by the vector $\mathbf{u}$ which dimension is $[M \times 1]$ (in our case the system has only one input, therefore $M = 1$). These signals are weighted by a constant $[N \times M]$ matrix $\mathbf{W^{in}}$ which elements are randomly initialized and kept constant during the RNN training and operation. The function of these synapses is to realize a sparse encoding of the inputs signals to better recognize patterns in the data. Only the first S neurons of the RNN are influenced by the inputs, therefore the last $N - S$ rows of the matrix $\mathbf{W^{in}}$ have all the elements equal to zero. The outputs of the neurons are represented by the elements of the vector $\mathbf{r}$ which dimension is $[N \times 1]$. The outputs of the readout units, which correspond also to the outputs of the RNN, are represented by the vector $\mathbf{z}$ which dimension is $[Q \times 1]$ (in our case only one output is required, thus $Q = 1$).

As shown in the RNN architecture of Fig. 5, the outputs of the readout units $\mathbf{z}$ are feedback to the inputs of the neurons through the matrix $\mathbf{W^{fb}}$ which dimension is $[N \times Q]$, where only the first P elements are different from zero, thus only the first P neurons of the RNN receive a feedback signal. This feedback is important "to force" the RNN to produce the proper periodic signals that are necessary to build the target RNN's outputs [34], [35].

Fundamental for a correct initialization of the *Connection Matrix* $\mathbf{W^r}$ is the Echo State Property (ESP) [23], which implies that the initial conditions will not have effect on the RNN's state after a certain amount of time. A restrictive constraint that ensures the ESP, in the case of no feedback connection from the readout units, is that $|1 - \frac{s}{\tau}(1 - \bar{\rho}(C_h\mathbf{W}^r))|$ is less than one, where $\bar{\rho}(C_h\mathbf{W}^r)$ represents the maximum singular value of the matrix $C_h\mathbf{W}^r$.

Depending on the initialization of the matrix $\mathbf{W}^r$, the RNN will have different kind of dynamic behaviors. When all the eigenvalues have real part less than one, and the input and/or the noise are suddenly removed, the output of the neurons will converge to zero. However, if the real part of some of the eigenvalues of the matrix $\mathbf{W}^r$ will overcome the value one, oscillations will occur and persist also when the input and/or the noise are removed. In general, the bigger is the real part of the eigenvalues and the more chaotic the behavior of the RNN will be. In other terms, the richer is the dynamics of the RNN the more capable to identify complex behaviors it will be. However, on the other hand, if the movement of the RNN are too chaotic the learning algorithm will not converge toward a stable solution. Fig. 6 represents a typical displacement of the eigenvalues in the complex plane, in red is represented the unit circle and in blue the circle that encloses all the eigenvalues and which radius is referred as the spectral radius of $\mathbf{W}^r$.

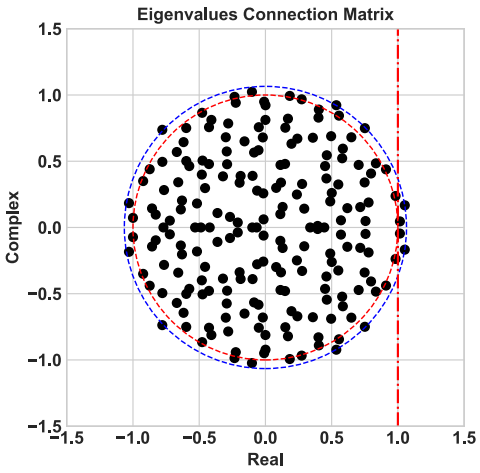

FIGURE 6. Eigenvalues displacements of the *Connection Matrix* $\mathbf{W^r}$. The eigenvalues with real part greater than one bring chaos in the RNN [34].

For the sake of completeness, Fig. 7 reports an example of the elements of the matrix $\mathbf{W^r}$, in red are reported the positive (excitatory) synapses and in blue the negative (inhibitory) synapses.

The dynamics of the RNN, see Eq. 2, is modeled by a first order differential equation representing a set of coupled leaky integrator neurons. In Eq. 2, $\mathbf{x}$ is a vector of neurons activation potentials and τ a time constant. The smaller is τ the higher will be the frequency of the activation potentials of the neurons. This translates in a better capability of the RNN to reproduce signals with high bandwidth.

$$\tau\dot{\mathbf{x}}(t) = -\mathbf{x}(t) + C_h\mathbf{W}^r\mathbf{r}(t) + \mathbf{W}^{fb}\mathbf{z}(t) + \mathbf{W}^{in}\mathbf{u}(t) \qquad (2)$$

To limit the outputs of the neurons in the range $[-1, +1]$ and to introduce a distributed nonlinear component in the RNN model, a hyperbolic tangent activation function is

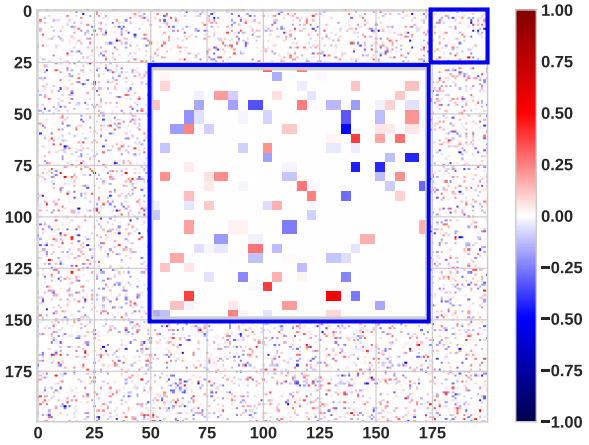

FIGURE 7. The RNN's connection matrix has dimension 200 × 200, is sparsely connected and presents a total of 8000 nonzero synapses, 4000 inhibitory and 4000 excitatory.

introduced as in Eq. 3.

$$\Phi(\mathbf{x}) = [tanh(x_1), tanh(x_2), \ldots, tanh(x_N)]^T \tag{3}$$

The outputs of the neurons are computed from the activation potentials using equation Eq. 4, where σ represents a Gaussian exploration noise [30], which serves to improve the learning process and bootstrap the activity of the RNN when no input is present.

$$\mathbf{r}(t) = \Phi(\mathbf{x}(t)) + \mathbf{G_{Noise}}(\mu, \sigma) \tag{4}$$

The outputs of the readout units are calculated via a linear combination of the outputs of the neurons as in Eq. 5, where $\mathbf{W^{Ad}}$ is a matrix of adaptive synapses whit dimension is $[Q \times N]$.

$$\mathbf{z}(t) = \mathbf{W^{Ad}}\mathbf{r}(t) \tag{5}$$

Due to the fact that only the synapses of the RNN associated with the inputs of the readout units are processed by the learning algorithm, everything boils down to a standard linear regression problem [36] for which different solutions already exist. In particular, to allow online identification and fast convergence, the elements of $\mathbf{W^{Ad}}$ are adapted at discrete time intervals using the Recursive Least Squares (RLS) algorithm as in Eq. 7. Where the error signal is calculated as the difference between the target and the predicted angular position of the robot, Eq. 6.

$$Err(t) = \overline{Pitch}(t) - \mathbf{W^{Ad}}(t)\mathbf{r}(t) \tag{6}$$

$$\mathbf{W^{Ad}}(t+1) = \mathbf{W^{Ad}}(t) + (\mathbf{F}(t+1)\mathbf{r}(t)Err(t)) \tag{7}$$

In Eq. 7, $\mathbf{F}$ represents an adaptive gain matrix which dimension is $[N \times N]$ and is calculated as Eq. 8.

$$\mathbf{F}(t+1) = \mathbf{F}(t) - \frac{\mathbf{F}(t)\mathbf{r}(t)\mathbf{r}(t)^T\mathbf{F}(t)}{1 + \mathbf{r}(t)^T\mathbf{F}(t)\mathbf{r}(t)} \tag{8}$$

It is important to emphasize that the RNN model expressed by Eq. 2-5, seen as a closed box, is not using any past input or output of the target system, but only the actual input.

A. STABILITY OF THE RESERVOIR-BASED RNN

To assess the stability of the RNN [37], by substituting Eq. 4 and 5, we can rewrite Eq. 2 as

$$\tau\dot{\mathbf{x}}(t) = -\mathbf{x}(t) + \bar{\mathbf{W}}\Phi(\mathbf{x}(t)) + \mathbf{W^{in}}\mathbf{u}(t), \tag{9}$$

where the matrix $\bar{\mathbf{W}}$ is defined as

$$\bar{\mathbf{W}} = C_h\mathbf{W}^r + \mathbf{W^{fb}}\mathbf{W^{Ad}}. \tag{10}$$

In obtaining Eq. 9 we assumed that, in first approximation, the noise can be neglected with respect to the output signal of the neurons.

By further approximate the differential equation as a difference equation, thus obtaining a discrete-time representation with step size Δt, and linearizing the Lipschitz continuous activation function $\Phi(\cdot)$ about the origin, we obtain Eq. 11 for the null input case, where $\mathbf{I}$ represents the $[N \times N]$ identity matrix.

$$\mathbf{x}(t+1) = [(1 - \frac{\Delta t}{\tau})\mathbf{I} + \frac{\Delta t}{\tau}\bar{\mathbf{W}}]\mathbf{x}(t) \tag{11}$$

According to linear systems theory, asymptotic stability is guarantied if the spectral radius of the matrix $(1 - \frac{\Delta t}{\tau})\mathbf{I} + \frac{\Delta t}{\tau}\bar{\mathbf{W}})$ is less than one. To notice that the matrix $\bar{\mathbf{W}}$ is directly dependent on the learning process that adapts the matrix $\mathbf{W^{Ad}}$.

B. CONSIDERATIONS ABOUT THE CONVERGENCE OF THE ADAPTATION ALGORITHM

A necessary condition for the learning algorithm to converge to an optimal solution for the matrix $\mathbf{W^{Ad}}$, thus $\mathbf{W^{Ad*}}$, is the availability of a rich dynamics in the reservoir. If we can demonstrate that the fixed point $\mathbf{W^{Ad*}}$ is asymptotically stable, then the convergence of the learning algorithm is guarantied.

Let us define $\tilde{\mathbf{W}^{Ad}} = \mathbf{W^{Ad*}} - \mathbf{W^{Ad}}$, to demonstrate the stability we can use a Lyapunov candidate function similar to the one proposed in [38] as for Eq. 12.

$$V(\mathbf{W^{Ad}}(t)) := \tilde{\mathbf{W}^{Ad}}(t))^T\mathbf{F}(t)\tilde{\mathbf{W}^{Ad}}(t) \tag{12}$$

where the matrix $\mathbf{F}(t)$ is symmetric and positive definite. According to Lyapunov stability criteria for a discrete system, the fixed point $\mathbf{W^{Ad*}}$ is asymptotically stable if

$$\begin{aligned} &V(\mathbf{W^{Ad}}) = 0 \ \ for \ \ \mathbf{W^{Ad}} = \mathbf{W^{Ad*}}, \\ &V(\mathbf{W^{Ad}}) > 0 \ \ \forall \ \ \mathbf{W^{Ad}} \neq \mathbf{W^{Ad*}}, \end{aligned} \tag{13}$$

and

$$\Delta V(\mathbf{W^{Ad}}) < 0 \ \ \forall \ \ \mathbf{W^{Ad}} \neq \mathbf{W^{Ad*}}, \tag{14}$$

where

$$\Delta V(\mathbf{W^{Ad}}(t)) = V(\mathbf{W^{Ad}}(t)) - V(\mathbf{W^{Ad}}(t-1)) \tag{15}$$

The first condition in Eq. 13 is true since when the optimal point is reached, $\tilde{\mathbf{W}^{Ad}} = \mathbf{O_{N,1}}$. Furthermore, due to the fact that $\mathbf{F}(t)$ is positive definite, Eq. 12 represents a quadratic form, therefore also the second condition in Eq. 13 is verified.

Finally, in order to demonstrate the condition in Eq. 14 we can substitute the Eq. 12 in Eq. 15 obtaining Eq. 16.

$$\Delta V(\mathbf{W^{Ad}}(t)) = (\tilde{\mathbf{W}^{Ad}}(t))^T \mathbf{F}(t) \tilde{\mathbf{W}^{Ad}}(t) + - (\tilde{\mathbf{W}^{Ad}}(t-1))^T \mathbf{F}(t-1) \tilde{\mathbf{W}^{Ad}}(t-1) \quad (16)$$

In comparison with the demonstration proposed in [38], here we have the complication that the gain matrix $\mathbf{F}(t)$ is changing with time, therefore a rigorous demonstration of the condition in Eq. 14 is difficult to obtain. We therefore resorted to demonstrate it numerically. The upper plot in Fig. 8 presents the numerical values of the $tr(\mathbf{F})$ during the adaptation process, as it is shown the $tr(\mathbf{F})$ is decreasing with time and converges to zero.

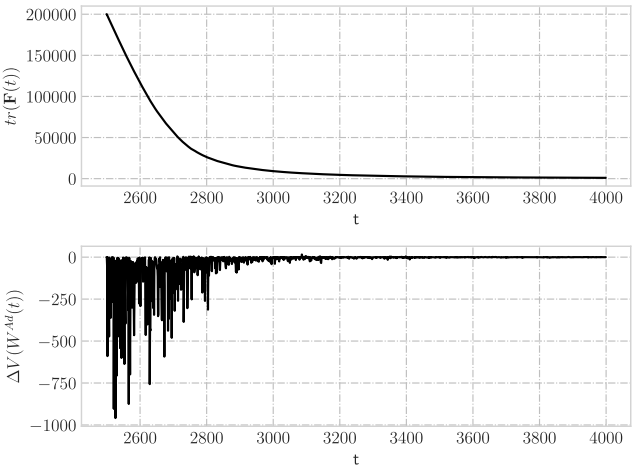

FIGURE 8. Plots showing the convergence of the gain matrix's trace $tr(\mathbf{F})$ (top) and the difference of the Lyapunov Function of two consecutive samples (bottom).

From the lower plot of Fig. 8, we can also observe that the numerical derivative of the Lyapunov function is negative or equal to zero, thus the algorithm is Lyapunov stable and the convergence is numerically guarantied.

IV. RESULTS

After the data set was elaborated according to the procedure described in section II, it was divided in two parts, the training set and the validation set. The training set represents the first 75% of the data and the validation set the last 25%. The RNN model and learning algorithm, represented by Eq. 2-8, were implemented as Python-3 code to have full control of all the variables. The RNN, consisting of a reservoir of 200 neurons and one input and one output, was initialized according to the parameters reported in the Table 1.

A. HYPER-PARAMETERS INITIALIZATION

The behavior of the Reservoir-Based RNN depends on the initialization of its hyperparameters. In the previous section, we provided some guidance on how to initialize the synapses of the RNN by fulfilling the constraint on the spectral radius of $(1-\frac{\Delta t}{\tau})\mathbf{I}+\frac{\Delta t}{\tau}\bar{\mathbf{W}}$). This section outlines how various factors such as the number of neurons, sparsity of the RNN, and the

TABLE 1. Reservoir-based RNN parameters.

Parameter	Description	Value
M	Number of external inputs	1
Q	Number of outputs (readout units)	1
Δt	Sample time	4 ms
Hyper-Parameter		
N	Number of neurons	200
Sp	Sparsity [34]	20%
S	Number of neurons receiving the inputs	100
P	Number of neurons receiving feedback	2
T	Neuron time constant	0.14
μ, σ	Readouts noise mean, variance	0, 0.01
C_h	Chaos-modulation constant	1.1

number of neurons involved in receiving input and output signals, impact both the training phase and overall performance of the RNN. The first hyperparameter we considered is the number of neurons present in the RNN.

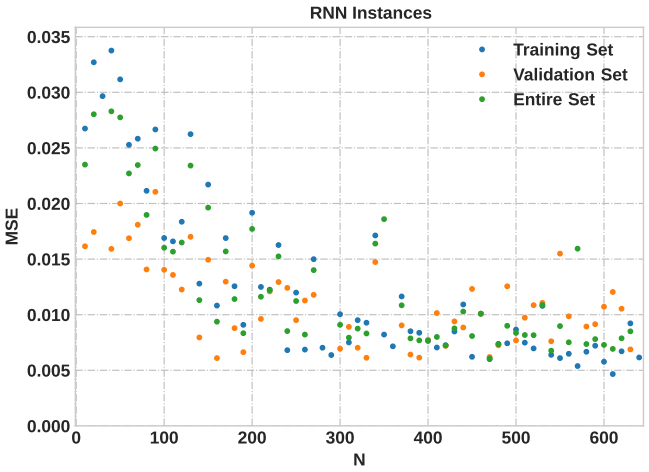

FIGURE 9. MSE for 64 instances of the RNN with a number of neurons ranging from a minimum of 10 to a maximum of 640 with step 10.

A low number of neurons reduces the capability of the network to remember the past samples of the input signal as well as its aptitude to approximate nonlinear dynamic systems. On the other hand, a large number of neurons requires more computation and may lead to over-fitting of the data. To investigate this, we trained a set of 64 RNNs with a number of neurons ranging from 10 to 640 with step 10, and measured the MSE on both the training set, validation set and entire data set. How it is possible to see from Fig. 9, the MSE decreases rapidly till we reach a number of 200 neurons, after the performance of the RNN increases more slowly with the number of neurons. Therefore, when it comes to find a balance between the RNN complexity and performance, for our model, it's recommended to use a neuron count between 200 and 400.

The second hyperparameter we considered is the level of sparsity of the RNN. According to [30], the number of recursive connections among the neurons should be kept in a range between 10-20 % of the total connections. This is necessary to promotes the generation of rich and diverse

temporal dynamics and at the same time to reduce the amount of computation, e.g., by using sparse matrix operations.

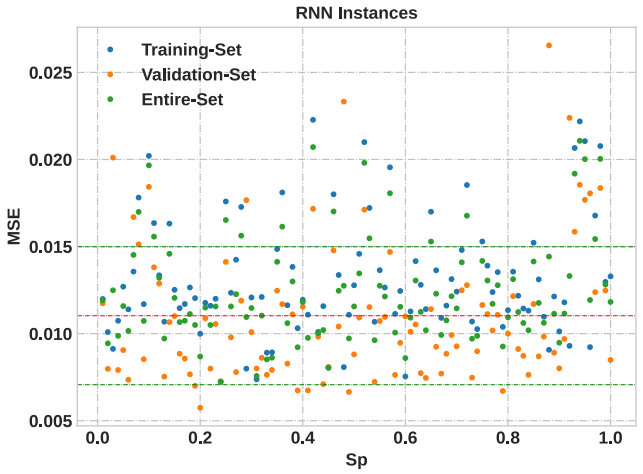

FIGURE 10. MSE for 100 RNNs instances with a sparsity (*Sp*) ranging from a minimum of 1 % to a maximum of 100 % with step 1.

To investigate this parameter, we trained 100 RNNs with sparsity values ranging from 1% to 100% and step 1%. From Fig. 10 it is possible to evince that the sparsity, in our case, does not effect significantly the performance of the RNN. For this reason, and to reduce the amount of computation, we opted for a $Sp = 20\%$.

As a third hyperparameter (see Table 1) we considered the number of the RNN's neurons that receive the input signal. As for the previous parameter, we can observe that from the simulations results of Fig. 11, ranging S from 2 to 200 with step 2 does not improve the behavior of the RNN. Thus we opted for the value $S = 2$.

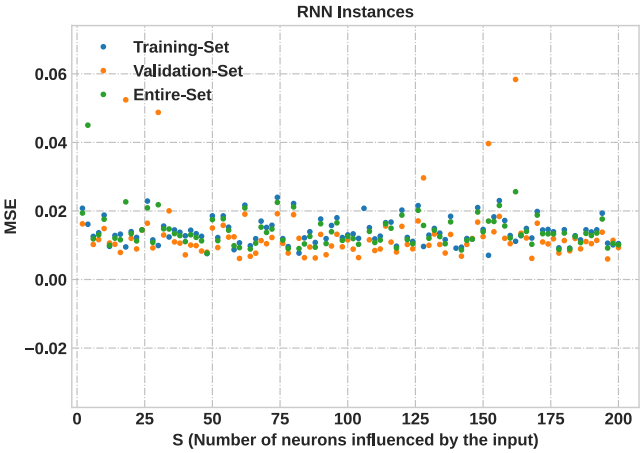

FIGURE 11. MSE for 100 RNNs instances with a number of neurons (*S*) receiving the input signal ranging from 2 to 200 with step 2.

Finally, we took into consideration the number of neurons influenced by the output of the read unit (P hyperparameter). This feedback has the main purpose to force the RNN to faster learn the target output signal [35], [38]. We considered 100 instanced of the RNN ranging P from 0 till 200 with step 2. How it is possible to notice in the plot of Fig. 12, after a value of $P = 36$ the performance of the RNN starts to degrade by presenting more instances of the RNN having unacceptable value for the MSE. If we zoom-in to the first 6 values of the plot, with a P ranging from 0 to 6, we notice that the performance of the RNN sets to a MSE value around 0.007. Thus an optimal range for P is between 2 to 5.

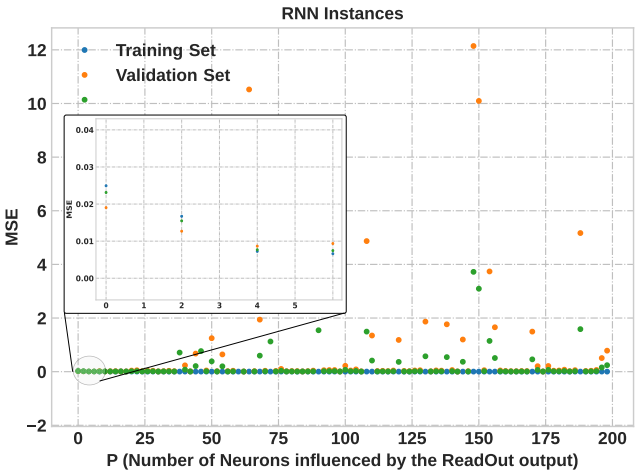

FIGURE 12. MSE for 100 RNNs instances with a number of neurons (*P*) receiving the readout unit signal ranging from 0 to 200 with step 2.

B. INVERTED-PENDULUM MODEL IDENTIFICATION

While training the model, we employed a series-parallel mode of identification, thus the actual input u(t) was provided to both the plan (our robot) and the reservoir-based RNN.

The performance of the RNN was evaluated calculating the MSE on the training set, validation set and entire data set. In order to increase the statistical significance of our results we trained 20 RNN models with different initialization of the *Connection Matrix* $\mathbf{W^r}$. As we mentioned before, the initialization of $\mathbf{W^r}$ is very important and influences the dynamic properties of the network. In Fig. 13 the MSEs for the different instances of the model are represented respectively for the training set, validation set and entire data set. The total mean and standard deviation of the model instances on the validation set are $mean(MSE_i)_{i=1}^{20} = 0.0081$ and $SD(MSE_i)_{i=1}^{20} = 0.0017$ respectively, where MSE_i represents the MSE of the i-th model instance. These figures are represented in the plot of Fig. 13 by the red and green horizontal dashed lines.

Among all the model instances we selected the one with lower MSE on the validation set. Fig. 14 shows for the entire data set, as dashed red line, the output of the single readout unit of the RNN that models the IMU's Pitch-Angle, while the black line represents the target signal. The vertical dashed line indicates when the synapses adaptation stops, at this point the RNN can replicate the target signal.

For the best RNN model instance, the MSE value is 0.0189 for the training set, 0.0048 for the test set and 0.0152 for the total data set. If we look to the last three motions (see Fig. 15), when the training phase is finished, the RNN can reproduce the movement of the robot with good

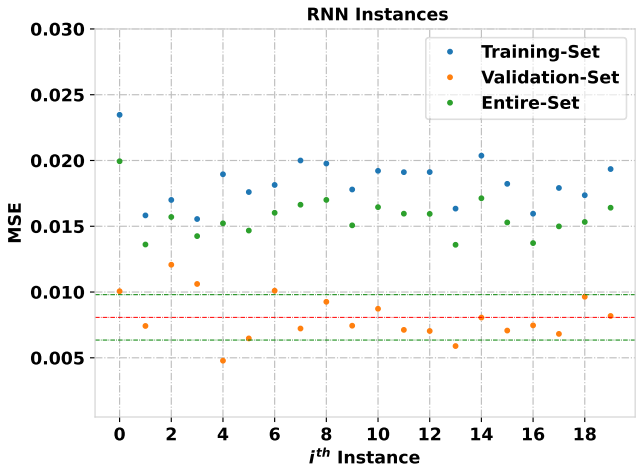

FIGURE 13. MSE calculated for 20 instances of the RNN model for the training set, validation set and all data set.

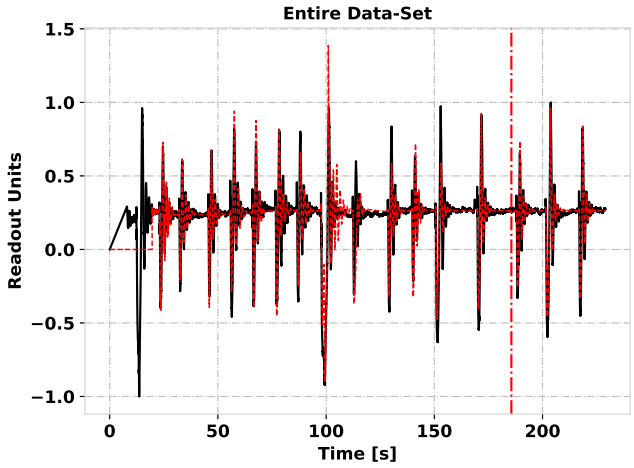

FIGURE 14. The output of the readout unit of the RNN is shown in red color, while in black is reported the target IMU's Pitch-Angle.

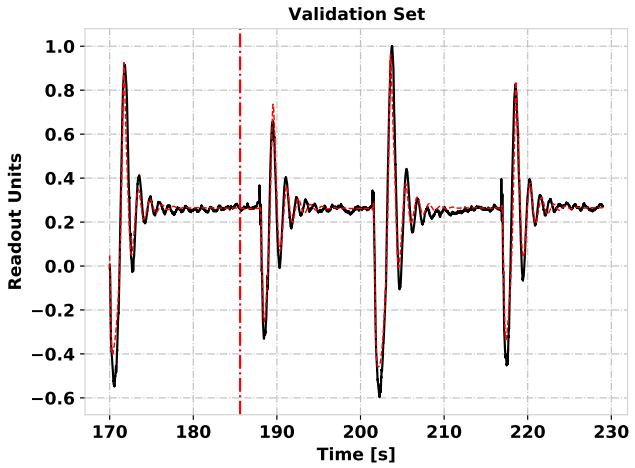

FIGURE 15. The output of the readout unit of the RNN, in red color, for the test set, at this point the learning phase is completed.

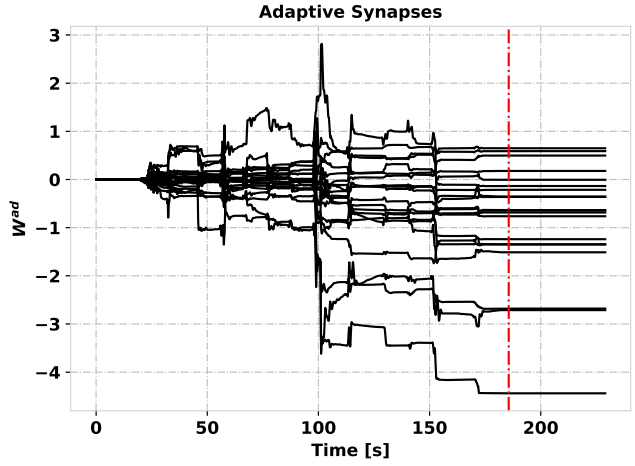

FIGURE 16. In the plot are represented 20 out of 200 of the synapses that are adapted by the RLS algorithm.

accuracy. In particular, the first three peaks of the oscillation are tracked correctly with some error in the amplitude. However, the following peaks are not replicated, or they are not synchronized with the target signal. From a pragmatic point of view, this has a limited impact on the control action due the fact that the amplitudes of these peaks are much smaller than the amplitudes of the previous one. It is also worth to mention that the RNN can predict the motion of the robot only using the information about the timing and amplitude of the impulse signal that represents the input force.

For completeness, Fig. 16 shows how the synapses $\mathbf{W}^{\mathbf{Ad}}$ are adapted by the RLS algorithm.

If the learning time is reduced, such as only one motion (one-shot) is used to adapt the synapses of the readout unit, the performance of the RNN decreases. The MSE value for the training set is now 0.0075, for the test set is 0.0329 and for the total data set is 0.0555. Overall, as shown in Fig. 17, the output of the RNN is still able to track the target Pitch-Angle.

In particular, if we consider the last four motions (see Fig. 18), we can observe a phase shift of the signal produced by the RNN in comparison with the target signal representing the Pitch-Angle. In addition, we can notice the presence of a positive offset with respect the reference signal. However, regarding the amplitudes of the oscillations, the performances are still comparable with the previous case. The first peak of the oscillation has also a correct timing, this is important for a model-based control system because it will allow to counteract the disturbance force in a more effective way.

Finally, Fig. 19 illustrates the adaptation of the synapses $\mathbf{W}^{\mathbf{Ad}}$ that in this case lasts 11 s.

C. COMPARISON WITH ARX, NNARX AND LSTM

When the dynamic system to be identified is linear, it is possible to approximate its dynamics using the well-known Auto Regressive Moving Average eXogenous input (ARMAX) model. In the case the system is not subject to dominating

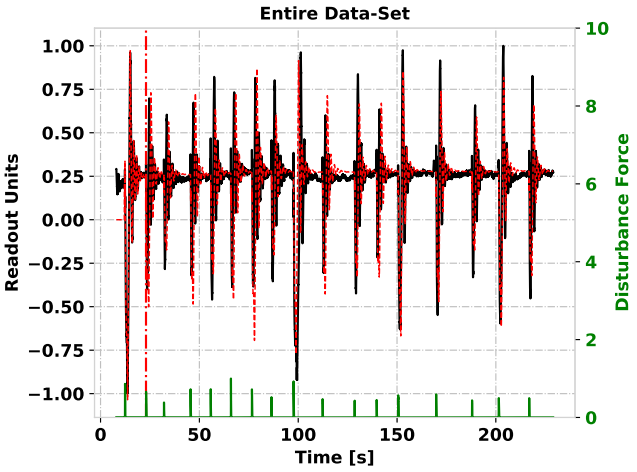

FIGURE 17. Training performed on one-shot (11 seconds). The dashed vertical red line indicates the instant of time where the training phase ends. The output of the readout unit of the RNN is shown in red color, in black is reported the target Pitch-Angle and in green color the normalized disturbance force.

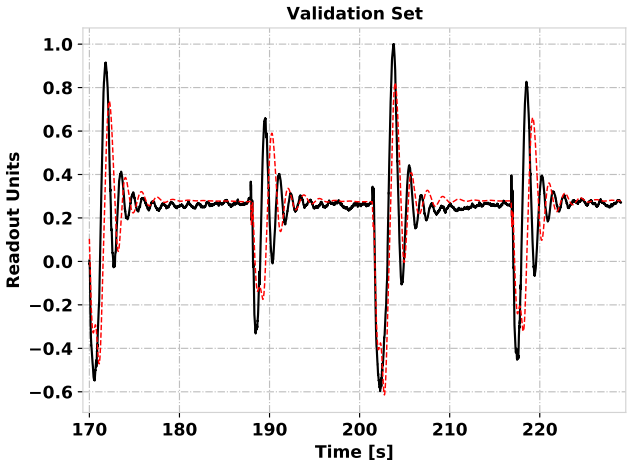

FIGURE 18. The output of the readout unit of the RNN for the validation set in the case of one-shot learning.

disturbance effects, we can remove the Moving Averaging part obtaining the simpler ARX model that computes its output from the past input and output values.

However, when the dynamic system to be identified is not linear it is necessary to add a nonlinear component to the model. As natural extension of the ARX model the Nonlinear ARX model (NARX) can be used to make one-step ahead predictions and can be written as Eq. 17.

$$y(t) = F[\mathbf{x}(t)] + \epsilon(t) \tag{17}$$

where $y(t)$ is the time series to predict (output of the process), $F(\cdot)$ is a general scalar function, $\mathbf{x}$ is the regression vector expressed by Eq. 18, and $\epsilon(t)$ is the prediction error of the model defined as $\hat{y}(t) - y(t)$ and assumed to be an

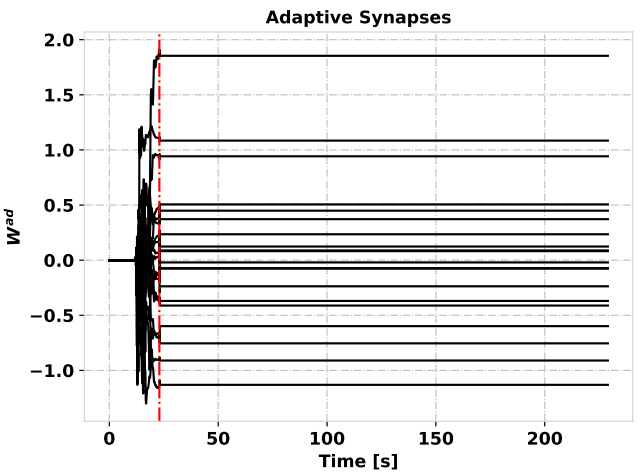

FIGURE 19. In the plot are represented 20 out of 200 of the synapses that are adapted by the RLS algorithm, the adaptation phase lasts 11 *s*.

independently distributed noise.

$$\mathbf{x}(t) = [y(t-1), \ldots, y(t-n), u(t-d), \ldots, u(t-d-m)] \tag{18}$$

In Eq. 18, $u(t)$ is the eXogeneous time series that represents the input of the process, d is a pure time delay and n and m the input and output regressions orders respectively. According to the chosen function $F(\cdot)$, Eq. 17 can represent a wide range of models. If $F(\cdot)$ is a linear combination of its arguments, than a simple and less computationally expensive ARX model is realized, i.e., $\hat{y}(t) = a_1 y(t-1), \ldots, a_n y(t-n) + b_1 u(t-1) + \ldots + b_m u(t-m)$. If $F(\cdot)$ is a power-form polynomial function with degree greater than one, than a nonlinear model is realized, i.e. NARX. As an example, if the degree is two and $n = m = 1$ and $d = 0$ then $\hat{y}(t) = a_0 + a_1 y(t-1) + a_2 y^2(t-1) + b_1 u(t-1) + b_2 u^2(t-1) + c_1 y(t-1)u(t-1)$. Other possibilities for $F(\cdot)$ are wavelet expansions, fuzzy logic-based models, neural networks, etc.

For the case where a multi-layer perceptron is used as $F(\cdot)$, a very powerful model is realized that can reproduce the dynamics of complex nonlinear dynamic systems. Depending on whether we feed the trained model with the past outputs of the real process or the predicted one, we have two possible configurations (see Fig. 20),which are indicated as NNARX type-A and NNARX type-B (also referred as NNOE) [7].

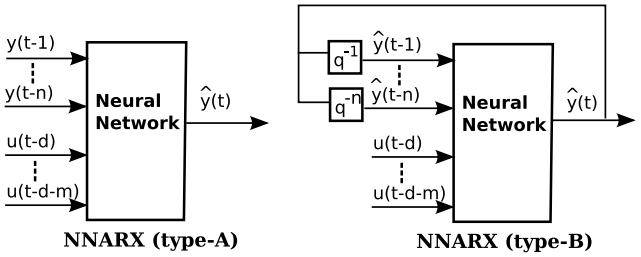

FIGURE 20. Two possible prediction schemes for the NNARX architecture.

Since the RNN model does not make usage of the previous process outputs to make the predictions, in this study, for an objective comparison, we considered only the NNARX type-B architecture.

In the simple case of a single hidden layer network (see Fig. 21) with p neurons and linear activation function for the output neuron, the NNARX type-B model can be expressed as Eq. 19

$$\hat{y}(t) = w_{o0} + \sum_{i=1}^{p} w_{oi} \Phi((\mathbf{W}(t)\mathbf{x}(t))[i]), \tag{19}$$

where $\mathbf{W}(t)$ is a $[p \times (n+m)]$ matrix, w_{oi} is the output synapse that connects the i-th neuron of the hidden layer with the output neuron and $\Phi(\cdot)$ represents a nonlinear activation function, e.g., a logistic function, applied to all activation potentials of the neurons of the hidden layer.

All the aforementioned models have a network architecture which differs if compared with the reference reservoir-based RNN. Therefore, to have a fair comparison, we started considering the same number of adapting parameters. In the case the performances of the compared model were not satisfactory, the number of adapting parameters were increased accordingly. To notice also that the same data sets and sampling time were used for all the considered models, see Table 2. A first comparison of the reservoir-based RNN was done with the ARX model implemented by using the Python's *statsmodels* library [39]. Our aim was to show that it is not possible to fully represent the dynamics of the robot with a linear system. As for the order of the regression vector of the past outputs and inputs, n=100 and m=100 were chosen respectively. In this way we have the same number of adapting parameters as in the RNN, i.e. the 200 synapses that connect the output of the neurons to the readout unit. The parameters of the ARX model were identified using a LSM (Least Squared Method).

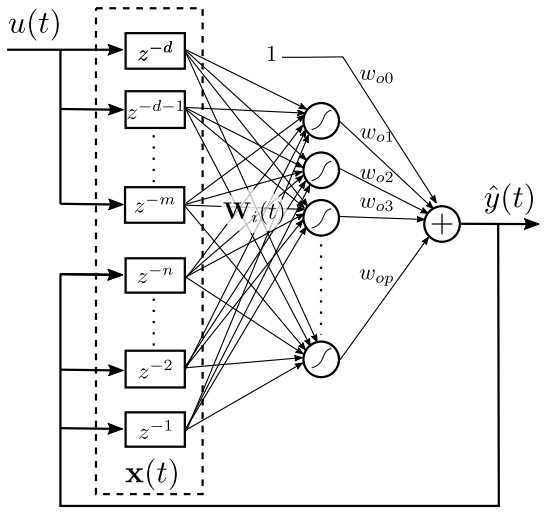

FIGURE 21. NNARX model consisting of a single hidden layer of p neurons.

TABLE 2. ARX and NNARX model parameters.

Parameter	Description	
M	Number of inputs	1
Q	Number of outputs	1
Δt	Sample time	4 ms
Hyper-Parameter ARX		
m	Input Regression Vector Dimension	100
n	Output Regression Vector Dimension	100
Hyper-Parameter NNARX		
m	Input Regression Vector Dimension	100
n	Output Regression Vector Dimension	100
l	Number of hidden layers	2
N	Number of neurons per layer	10
η	Learning Constant	0.001

Notice that, once trained, the ARX model works in a multi-step prediction modality, therefore it predicts the outputs of the system receiving as input the n past steps of predicted output and the m past steps of the system input, thus the regression vector is $\mathbf{x}(t) = [\hat{y}(t-1), \ldots, \hat{y}(t-n), u(t-1), \ldots, u(t-1-m)]$.

Using the ARX model, with the validation set we obtained a MSE equal to 0.0181, with the training set equal to 0.0439 and with the entire data set equal to 0.0357. As it is possible to see from the plot in Fig. 22 the first pick is overshoot by the ARX model while the second pick undershoot. It is also possible to notice that after the third pick, the prediction goes flat.

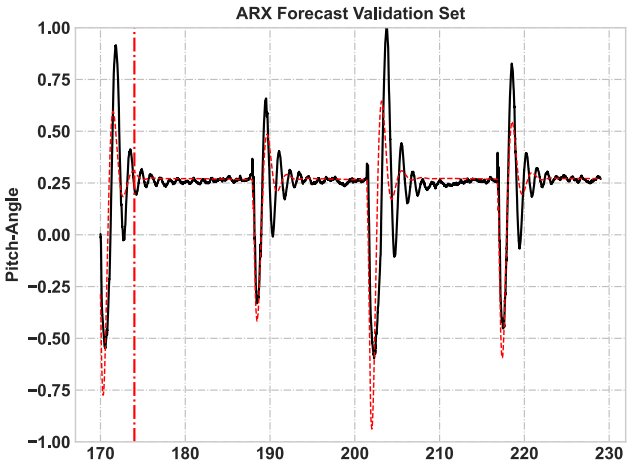

FIGURE 22. Prediction on the validation set for an ARX model with a regression vector of dimensions n=m=100.

The second comparison was done with a NNARX model. In this case we used a FNN with two hidden layers of 10 neurons each, a constant learning rate of 0.001 and the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) learning algorithm with a maximum of 500 iterations. Considering a fully connected NN, and including the bias, the number of adaptive synapses is equal to $(n+m)*10+10$ for the first hidden layer, $10*10+10$ for the second layer, and $10+1$ for the output layer that consists of only one neuron. In total there are 2131 adaptive parameters in a NNARX model in comparison with only 200 of the RNN model. We also tested NNARX models with

a smaller number of neurons in the hidden layers, however they obtained very low performances.

Since the NNARX learning process depends on a random component, to increase the statistical significance of our results we trained 100 NNARX models. For certain instances of the training process the learning algorithm did not converge to a minimum, in other cases the learning algorithm converged to a minimum of the error function. However, the MSE on the test set was far too high and therefore the model was not significant. That is why, after eliminating the outliers by considering the Q1 and Q3 quantiles of the MSE calculated on the training data set, we selected 62 models out of the 100 trained. Fig. 23 shows the MSEs for the different model instances on the training set, validation set, and entire set. In particular, the $mean(MSE_i)_{i=1}^{62}$ (red dashed line) and standard deviation $SD(MSE_i)_{i=1}^{62}$ (green dashed lines) calculated on the validation set are 0.0328 and 0.2305 respectively.

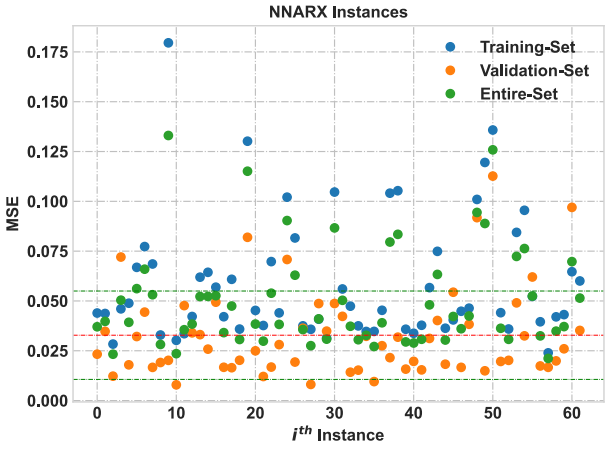

FIGURE 23. MSEs for the 62 instances of the 10-10 NNARX model with n=100 and m=100. The red horizontal dashed line represents the mean of the MSEs for the validation set on all the instances, while the green dashed line the standard deviation.

Finally, out of the 62 NNARX models we selected the best with a MSE for the validation set equal to 0.0079, for the training set 0.0302, and for the entire set 0.0235. Fig. 24 illustrates the output of the NNARX model for the validation set. The model can track correctly the first pick of the response and undershoots the second pick. As main difference in comparison with the pure ARX model, NNARX presents all the picks. However, in comparison with the reservoir-based RNN model (see Fig. 15), NNARX is not accurate in tracking the smaller picks, which are generally overshot.

The last comparison was done with a LSTM RNN based model [21] developed using Keras [40]. We implemented a NN having one neuron in the input layer, N neurons in the recurrent hidden layer, 10 neurons in the feed forward hidden layer and one neuron in the output layer. Thus, according to the LSTM cell structure, the number of adaptive synapses (bias included) of the overall network is $N_p = 4 * (2 * N + N^2) + (N * 10 + 10) + 10 + 1 = 4N^2 + 18N + 21$, e.g.

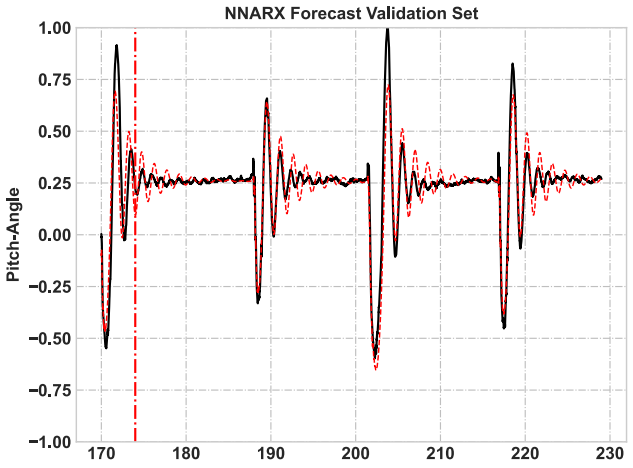

FIGURE 24. Prediction on the validation set of a NNARX model with a regression vector of dimension n=m=100 and two hidden layers of 10 neurons each.

for N=5 we have a NN with a total of 211 adaptive synapses. We tested different RNN's with the dimension from few to hundreds of units and trained them with different number of epochs. However, after training, the NN could not predict the target output. A possible explanation is that the impulses do not last enough to excite the RNN. What we obtained is in line with what is stated in [11], [12], and [32], that is, a LSTM in its original structure is not suitable for model identification problems and therefore more complex architectures must be considered. In [17] a LSTM is used to identify a nonlinear crystal growth process. However, the NN not only receives the process input but also its delayed output, which is not the case for our reference model identification setup.

We conclude this section by presenting a comparison of the three architectures that were considered in this study and resumed by Table 3. From the table it is possible to notice that the lowest MSE on the validation set is obtained by the reservoir-based RNN and NNARX models, while the ARX, as expected, has the worst performances. If we compare directly the RNN with NNARX, we can observe that not only the RNN performs better but it also has less variability when considering different training instances. In addition, even with only one example the reservoir-based RNN architecture can show acceptable performance. This is particularly important if the model is used as feed-forward module in a control scheme that has the goal to keep the robot in equilibrium while affected by disturbance forces [41], [42], [43].

D. CONSIDERATIONS ON COMPUTATIONAL COMPLEXITY AND REAL-TIME EXECUTION

The N synapses of the reservoir-based RNN are trained with the RLS algorithm which temporal complexity is $O(N^2)$ for each input-output pair presented. If T is the number of samples used for training, the complexity of the RNN is therefore $O(TN^2)$.

TABLE 3. Comparison between the different models in terms of mean and standard deviation of the MSE computed for the RNN on 20 model instances and for the NNARX on 62 model instances. Data are reported for both the training and validation sets.

MODEL	TRAINING SET	$\{MSE_i\}_{i=1}^{M}$	
	Best	mean	SD
ARX	0.0439	-	-
NNARX	0.0302	0.0586	0.0306
RNN	0.0189	0.0183	0.0018
RNN One-Shot	0.0075	-	-
	VALIDATION SET	$\{MSE_i\}_{i=1}^{M}$	
	Best	mean	SD
ARX	0.0181	-	-
NNARX	0.0079	0.0328	0.0221
RNN	0.0048	0.0081	0.0017
RNN One-Shot	0.0329	-	-

The LSM, used to train an ARX model, has complexity $O(N^3)$ for the Cholesky factorization and $O(TN^2)$ for the matrices multiplication, however if $T \gg N$, like in our case, the term TN^2 will dominate on the term N^3 and the overall complexity is again $O(TN^2)$.

To train the NNARX model it is necessary to use the BP (Back Propagation) algorithm. The BP algorithm has a computational complexity linear with the number of synapses present in the layers and depends on the number of samples T and epochs E. For the NNARX model that we considered in this study with one input and one output, a regression vector $\mathbf{x}$ of dimension $n + m$ and two hidden layers of N neuron each, we have therefore a time complexity $O(T * E * [(n + m) * N + N^2 + N])$. Considering that BP requires more epochs to converge, among the three learning paradigms it represents the one with the highest temporal complexity and therefore the less suitable to be applied in online adaptive controllers [44].

After a reservoir-based RNN is trained, it has to be saved for future usages, e.g., within a model-based control system architecture, or re-implemented in hardware to allow a fast computation [45], [46]. We refer here to this procedure as ''cloning the trained network''. If the proper conditions are not recreated in the cloned RNN it will run in a different way and the performances will be lower than the original neural network.

To guaranty a proper ''cloning'', we had to modify the training procedure of the original RNN. The RNN starts to learn and receives the input only after a certain amount of time, the *Start Learning Time*. Thus, at the beginning the RNN starts to oscillate due to the noise, then the learning phase takes place together with the effect of the input(s). Due to the fact that the inputs influence the dynamics of the RNN, it is necessary that the RNN reaches a certain regime before applying learning. In Fig. 25 we can observe some of the RNN neurons' outputs. Initially, when noise is injected, there are small oscillations for about 4 *s*, then the oscillations increase till reaching the maximum amplitude. Finally, when the adaptation phase starts, periodic signals are generated.

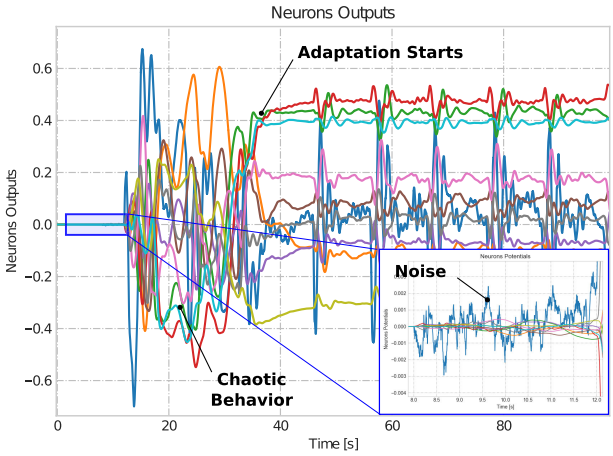

FIGURE 25. The outputs of the RNN's neurons start to oscillate when noise is injected in the network. At the beginning the RNN has a quasi-chaotic behavior. However, when the adaptation phase commences, the neurons start to generate periodic signals.

To clone the RNN, at the end of the training phase the Connection Matrix $\mathbf{W^r}$ is saved together with the matrix of adapted synapses $\mathbf{W^{Ad}}$, and the matrices $\mathbf{W^{in}}$ and $\mathbf{W^{fb}}$.

The so cloned RNN was tested on the Orange Pi 3 and Jetson Nano minicomputers. From our profiling tests, we could obtain a running frequency of 500 Hz on the Orange Pi 3, thus the RNN can be computed with a sample time of 2 ms which is a half of what was used to sample the original data (4 ms). On the more powerful Jetson Nano we could reach a frequency of 1 KHz, therefore with a sample time four time smaller than the one used to acquire the data. In both setups, the behavior and performance of the cloned RNN was equivalent to the original one.

V. CONCLUSION

In this paper, a nonlinear input-output dynamic model based on a reservoir-based RNN was presented. The particular architecture of the RNN and the applied learning algorithm allow for fast system identification based on a few input-output examples of the system response. In details, the RNN consists of a reservoir of N neurons sparsely and randomly connected by positive and negative synapses that are initialized in a way to guaranty stable oscillations. Gaussian noise is added to the output of the neurons in order to obtain a chaotic behavior during the start-up of the RNN, also without the presence of input signals. The output of the RNN is obtained by a linear combination of the outputs of the neurons, where the synapses are adapted using the RLS algorithm, which is suitable to implement real-time identification of the target dynamic system.

The RNN was used to model the input-output characteristic of a bipedal robot which posture was perturbed by impulsive forces applied to the upper body. The robot under study, represented as an inverted pendulum, behaves as a nonlinear dynamic system also due to the presence of elastic and friction components in the actuation system.

Results show that the adaptive model can reproduce the behavior of the target system with good accuracy. Thus, once trained, the RNN can predict the output of the dynamic system using only its actual input, in our case the impulsive force. Furthermore, the RNN has acceptable performances also when only one instance of the experiment is used for training.

Computationally, the RNN in combination with a RLS adaptation algorithm is more suitable for real-time system identification. In comparison, the BP algorithm, applied to identify an NNARX model, requires multiple epochs of training before converging to a minimum of the error function. Compared with other state-of-the-art RNN architectures, e.g. a LSTM RNNs trained with the BP Through Time (BPTT) algorithm [12], [13], [14], [15], [17], [32], a reservoir-based RNN does not requires off-line training and can adapt very rapidly to changes in the dynamic behavior of the target system.

To allow the deployment of the trained reservoir-based RNN, a procedure was developed to clone its data structure and architecture, thereby enabling its execution on different machines. From our tests, on a Jetson Nano running a Linux based OS, it is possible to execute the Python code with frequencies up to 1 KHz.

As a future work we intend to integrate the RNN in a model-based control system architecture in order to stabilize the posture of the robot under the presence of perturbation forces. In particular, the control system will consist of two main parts: the Feed-Forward module represented by the reservoir-based RNN, and the Feedback module represented by a classical PID controller that uses the error feedback in order to force the process to follow the target trajectory without causing the system instability.

REFERENCES

[1] K. Yamazaki, R. Ueda, S. Nozawa, M. Kojima, K. Okada, K. Matsumoto, M. Ishikawa, I. Shimoyama, and M. Inaba, "Home-assistant robot for an aging society," *Proc. IEEE*, vol. 100, no. 8, pp. 2429–2441, Aug. 2012.

[2] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, Jun. 1989.

[3] S. Yildirim, "Design of adaptive robot control system using recurrent neural network," *J. Intell. Robotic Syst.*, vol. 44, no. 3, pp. 247–261, Nov. 2005.

[4] J. L. Elman, "Finding structure in time," *Cognit. Sci.*, vol. 14, no. 2, pp. 179–211, Mar. 1990.

[5] S. S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Upper Saddle River, NJ, USA: Pearson Education, 2009.

[6] H. Uchida Frausto and J. G. Pieters, "Modelling greenhouse temperature using system identification by means of neural networks," *Neurocomputing*, vol. 56, pp. 423–428, Jan. 2004.

[7] M. Nørgaard, O. Ravn, N. Poulsen, and L. Hansen, *Neural Networks for Modelling and Control of Dynamic Systems*. London, U.K.: Springer, 2000.

[8] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 6, no. 2, pp. 107–116, Apr. 1998.

[9] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," in *Proc. 9th Int. Conf. Artif. Neural Netw. (ICANN)*, vol. 2. Sep. 1999, pp. 850–855.

[10] V. Sharma, M. Gupta, A. Kumar, and D. Mishra, "Video processing using deep learning techniques: A systematic literature review," *IEEE Access*, vol. 9, pp. 139489–139507, 2021.

[11] Y. Wang, "A new concept using LSTM neural networks for dynamic system identification," in *Proc. Amer. Control Conf. (ACC)*, May 2017, pp. 5324–5329.

[12] S. Wen, Y. Wang, Y. Tang, Y. Xu, P. Li, and T. Zhao, "Real-time identification of power fluctuations based on LSTM recurrent neural network: A case study on Singapore power system," *IEEE Trans. Ind. Informat.*, vol. 15, no. 9, pp. 5266–5275, Sep. 2019.

[13] M. Moradi A., S. A. Sadrossadat, and V. Derhami, "Long short-term memory neural networks for modeling nonlinear electronic components," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 11, no. 5, pp. 840–847, May 2021.

[14] N. Hirose and R. Tajima, "Modeling of rolling friction by recurrent neural network using LSTM," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 6471–6478.

[15] M. Qiao, S. Yan, X. Tang, and C. Xu, "Deep convolutional and LSTM recurrent neural networks for rolling bearing fault diagnosis under strong noises and variable loads," *IEEE Access*, vol. 8, pp. 66257–66269, 2020.

[16] S. Ye, J. Wei, Z. Xu, W. Zhang, X. Liu, and C. Zhang, "LSTM-based rapid identification of dominant low frequency oscillation modal features in power system," in *Proc. IEEE 4th Conf. Energy Internet Energy Syst. Integr. (EI)*, Oct. 2020, pp. 2455–2460.

[17] J. Zhang, Q. Tang, and D. Liu, "Research into the LSTM neural network-based crystal growth process model identification," *IEEE Trans. Semicond. Manuf.*, vol. 32, no. 2, pp. 220–225, May 2019.

[18] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Netw.*, vol. 1, no. 4, pp. 339–356, Jan. 1988.

[19] R. J. Williams and D. Zipser, "Experimental analysis of the real-time recurrent learning algorithm," *Connection Sci.*, vol. 1, no. 1, pp. 87–111, Jan. 1989.

[20] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960.

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

[22] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, "Recent advances in physical reservoir computing: A review," *Neural Netw.*, vol. 115, pp. 100–123, Jul. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608019300784

[23] H. Jaeger, "The 'echo state' approach to analysing and training recurrent neural networks—With an Erratum note," Fraunhofer Inst. Auton. Intell. Syst., Germany, Tech. Rep. 148, 2010.

[24] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, Apr. 2004.

[25] G. Manjunath and H. Jaeger, "Echo state property linked to an input: Exploring a fundamental characteristic of recurrent neural networks," *Neural Comput.*, vol. 25, no. 3, pp. 671–696, Mar. 2013.

[26] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, Nov. 2002.

[27] W. Maass, P. Joshi, and E. D. Sontag, "Computational aspects of feedback in neural circuits," *PLOS Comput. Biol.*, vol. 3, no. 1, pp. 1–20, 2007.

[28] M. Bana, A. M. Franchi, G. Gini, A. Keldibek, and M. Folgheraiter, "Learning and executing rhythmic movements through chaotic neural networks: A new method for walking humanoid robots," in *Proc. 47th Int. Symp. Robot.*, Munich, Jun. 2016, pp. 528–533.

[29] M. Folgheraiter, "A combined B-spline-neural-network and ARX model for online identification of nonlinear dynamic actuation systems," *Neurocomputing*, vol. 175, pp. 433–442, Jan. 2016.

[30] G. M. Hoerzer, R. Legenstein, and W. Maass, "Emergence of complex computational structures from chaotic neural networks through reward-modulated Hebbian learning," *Cerebral Cortex*, vol. 24, no. 3, pp. 677–690, Mar. 2014.

[31] P. Auer, H. Burgsteiner, and W. Maass, "A learning rule for very simple universal approximators consisting of a single layer of perceptrons," *Neural Netw.*, vol. 21, no. 5, pp. 786–795, Jun. 2008.

[32] J. Gonzalez and W. Yu, "Non-linear system modeling using LSTM neural networks," *IFAC-PapersOnLine*, vol. 51, no. 13, pp. 485–489, 2018.

[33] M. Folgheraiter and B. Aubakir, "Design and modeling of a lightweight and low power consumption full-scale biped robot," *Int. J. Humanoid Robot.*, vol. 15, no. 5, Oct. 2018, Art. no. 1850022.

[34] D. Sussillo and L. F. Abbott, "Generating coherent patterns of activity from chaotic neural networks," *Neuron*, vol. 63, no. 4, pp. 544–557, Aug. 2009.
[35] B. DePasquale, C. J. Cueva, K. Rajan, G. S. Escola, and L. F. Abbott, "Full-FORCE: A target-based method for training recurrent networks," *PLoS ONE*, vol. 13, no. 2, Feb. 2018, Art. no. e0191527.
[36] H. Jaeger, *Foreword to the Book Reservoir Computing: Theory, Physical Implementations, and Applications* (Natural Computing Series). Berlin, Germany: Springer Nature, 2021, pp. 5–10.
[37] H. Jaeger, "The 'echo state' approach to analysing and training recurrent neural networks," German Nat. Res. Center Inf. Technol., Hanover, Germany, Tech. Rep. 148, 2001.
[38] H. Tamura and G. Tanaka, "Transfer-RLS method and transfer-FORCE learning for simple and fast training of reservoir computing models," *Neural Netw.*, vol. 143, pp. 550–563, Nov. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S089360802100263X
[39] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with Python," in *Proc. Python Sci. Conf.*, 2010, pp. 1–9.
[40] F. Chollet. (2015). *Keras*. [Online]. Available: https://github.com/fchollet/keras
[41] P. Sardain and G. Bessonnet, "Forces acting on a biped robot. Center of pressure-zero moment point," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 34, no. 5, pp. 630–637, Sep. 2004.
[42] F. Sygulla, R. Wittmann, P. Seiwald, A. Hildebrandt, D. Wahrmann, and D. Rixen, "Hybrid position/force control for biped robot stabilization with integrated center of mass dynamics," in *Proc. IEEE-RAS 17th Int. Conf. Humanoid Robot. (Humanoids)*, Nov. 2017, pp. 742–748.
[43] M. Folgheraiter, A. Yessaly, G. Kaliyev, A. Yskak, S. Yessirkepov, A. Oleinikov, and G. Gini, "Computational efficient balance control for a lightweight biped robot with sensor based ZMP estimation," in *Proc. IEEE-RAS 18th Int. Conf. Humanoid Robots (Humanoids)*, Nov. 2018, pp. 232–237.
[44] I. D. Landau and G. Zito, *Digital Control Systems: Design, Identification and Implementation*. Cham, Switzerland: Springer, 2006.
[45] A. Rao, P. Plank, A. Wild, and W. Maass, "A long short-term memory for AI applications in spike-based neuromorphic hardware," *Nature Mach. Intell.*, vol. 4, no. 5, pp. 467–479, May 2022.
[46] L. Wang, Z. Yang, S. Guo, L. Qu, X. Zhang, Z. Kang, and W. Xu, "LSM-Core: A 69k-Synapse/mm^2 single-core digital neuromorphic processor for liquid state machine," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 5, pp. 1976–1989, May 2022.

MICHELE FOLGHERAITER (Member, IEEE) received the M.S. degree in computer engineering with major in automation systems and the Ph.D. degree in information technology from Politecnico di Milano, in 2000 and 2004, respectively. From 2002 to 2007, he was an Adjunct Researcher and Teaching Assistant in robotics with Politecnico di Milano. From 2008 to 2013, he was a Senior Researcher with the German Research Center for Artificial Intelligence (DFKI). Since 2013, he has been an Associate Professor with the Department of Robotics, Nazarbayev University. His research interests include humanoid robotics, haptic interfaces, and bio-inspired control systems.

ASSET YSKAK received the M.Eng. degree in sciences from Al-Farabi Kazakh University, in 2016. From 2016 to 2017, he was a Junior Researcher with Al-Farabi University. From 2018 to 2020, he was a Research Assistant in robotics with Nazarbayev University. His research interests include computer science and humanoid robotics.

SHARAFATDIN YESSIRKEPOV received the B.Sc. degree from Kazakh National Research Technical University, in 2010, and the M.Eng. degree from the University of Alberta, in 2014. He is currently pursuing the Ph.D. degree in robotics with Nazarbayev University. His research interests include teleoperation systems, humanoid robotics, and parallel manipulators.