Autoencoder-Inspired Identification of LTI Systems

Tobias Nagel[1] and Marco F. Huber[1,2]

Abstract— **Identifying the state space representation of a dynamical system during usage enables a controller to adapt quickly in a changing environment. We propose a new method for identifying linear time-invariant (LTI) systems online based on the measurement of input-output data. Therefore, we implement the calculation of a system response in a machine learning framework and use an autoencoder-related approach to find a neural network which performs a system identification by one single forward pass. This is computationally efficient and can be performed online during usage. We validate the approach by identifying the wear of a robot leg.**

I. INTRODUCTION

Identifying a state space representation of a system is the first step in designing a controller for gaining a desired system behavior. Setting up the corresponding differential equations manually, usually needs a lot of insight in the system. Besides identifying all energy storing components, the engineer has to know the relevant physical dependencies and terms, so a Lagrangian equation can be defined, which is more difficult with a growing system complexity.

For linear state space representations, data-driven approaches allow a simplified identification. If input-output data is given, so-called subspace identification methods are well established. The Subspace-based State Space Identification (or 4SID) calculates a state space representation by performing a singular value decomposition on measured data in order to get an extended observability matrix [18]. Afterwards a leastsquares algorithm leads to the parameters of a discrete-time differential equation system. If the identified system has to be stable, the algorithm needs to be extended by a constrained optimization [11]. However, as stated in [10], the listed methods are not convenient for identifying a system online, especially if the available computing resources are limited. This for instance is necessary if we want to implement a model-based reinforcement learning algorithm, where a frequent update of the learned dynamics model is required. More recently, Hardt et. al. showed that a system identification is possible by using gradient descent methods with polynomial complexity [7]. Additionally Sarkar et. al. proposed a solution that makes an identification of a lower state space order possible, which enables a decreasing computational effort [16].

Algorithms in machine learning tend to be easier adaptable than regular control approaches. A popular method, called PILCO, is described in [5], where a Gaussian process is trained for representing simple dynamics and is used afterwards to obtain a policy that leads to the control target online. Gaussian processes can be trained or updated efficiently, as has been shown in [8], which makes them feasible for environments with little computational resources. Systems with a large dimension require models with increasing degrees of freedom, such as neural networks [14, 21], which are computationally expensive, since a change in the system's dynamics requires a new training iteration. However, the usage of machine learning algorithms for modeling the dynamics of a system generally prevents a mathematical proof of guaranteed stability the same way state space systems can. Wang et. al. [19] avoid this issue by training a neural network with a special architecture in such a way that the weights are equivalent to a state space model of a defined order. Nevertheless the problem of being computationally efficient, while having insight in the state spaces, persists.

In this paper a new approach is introduced that makes an online state space identification possible by limited computational resources. We do so by creating a variant of an autoencoder: The measurements are fed into an encoding neural network which outputs an estimate of a suitable state space representation. The decoder utilizes this estimate to calculate the system's response given the original input. Like standard autoencoders, training is conducted in an unsupervised fashion and requires significant computing resources, depending on the application. During runtime, a single forward pass through the encoder network is sufficient for system identification, which is computationally cheap. As opposed to the 4SID-methods, we identify a continuous-time state space model directly. The approach, which is yet limited to linear state spaces, can also be seen as a baseline for future research towards nonlinear systems.

The rest of the paper is organized as follows: In Section II we give an introduction in system theory and neural networks. Afterwards the implementation and training of the proposed method is shown in Section III. The results of a simulation-based validation are discussed in Section IV. The paper closes with conclusions and an outlook on future work.

II. PRELIMINARIES

The state space representation of a linear time-invariant single input single output (SISO-) system is defined by means of

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A} \cdot \mathbf{x}(t) + \mathbf{b} \cdot u(t) \\ y(t) &= \mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}(t) + d \cdot u(t) \ , \end{aligned} \tag{1}$$

[1]Tobias Nagel and Marco F. Huber are with the Fraunhofer Institute for Manufacturing, Engineering and Automation IPA, Center for Cyber Cognitive Intelligence (CCI), 70569 Stuttgart, Germany {tobias.nagel, marco.huber}@ipa.fraunhofer.de

[2]Marco F. Huber is with the Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, 70569 Stuttgart, Germany marco.huber@ieee.org

where $\mathbf{x}$ denotes the state vector and u, y denote the input and output, respectively. $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}^{\mathrm{T}}$ and d perform linear transformations at inputs and states. The solution of this differential equation is

$$\mathbf{x}(t) = e^{\mathbf{A}t} \cdot \mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)} \cdot \mathbf{b} \cdot u(\tau) \mathrm{d}\tau \tag{2}$$

according to [13], where $e^{\mathbf{A}t}$ is a matrix exponential. By substituting (2) into (1), the output signal can be calculated by means of

$$y(t) = \mathbf{c}^{\mathrm{T}} \left(e^{\mathbf{A}t} \cdot \mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)} \cdot \mathbf{b} \cdot u(\tau) \mathrm{d}\tau \right) + du \ . \tag{3}$$

The stationary value y_∞ of a system denotes its output when stimulated by a step signal and waited for an infinite amount of time. It can be calculated by transforming the continuous state space equation to the Laplacian transfer function at the frequency $s = 0$ which leads to

$$y_\infty = d - \mathbf{c}^{\mathrm{T}} \mathbf{A}^{-1} \mathbf{b} \ . \tag{4}$$

A. Canonical State Space Representations

The choice of a system's state space representation is not unique. There are multiple linearly dependent combinations to build a suitable state space. To avoid redundancy, there exist several canonical forms. In the SISO case, the controllable canonical form (CCF) uses only variables in the bottom row of the system matrix $\mathbf{A}$ and in the output vector $\mathbf{c}^{\mathrm{T}}$, leading to

$$\begin{aligned} \dot{\mathbf{x}} &= \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & \cdots & -a_{n-1} \end{pmatrix} \cdot \mathbf{x} + \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} u \ , \\ y &= \begin{pmatrix} c_0 & c_1 & \cdots & c_n \end{pmatrix} \mathbf{x} + du \ , \end{aligned} \tag{5}$$

where $a_0, \ldots, a_{n-1}$ denote the coefficients of the corresponding transfer function denominator. Another special state space representation is the modal canonical form (MCF), where the eigenvalues of the dynamic matrix $\mathbf{A}$ are listed on the diagonal, thus the system states become decoupled. If there are complex conjugate eigenvalues, the complex part is listed on the adjacent diagonal. This leads to

$$\begin{aligned} \dot{\mathbf{x}} &= \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \delta_2 & \omega_2 & 0 \\ 0 & -\omega_2 & \delta_2 & 0 \\ 0 & 0 & 0 & \lambda_3 \end{pmatrix} \cdot \mathbf{x} + \tilde{\mathbf{b}} u \ , \\ y &= \tilde{\mathbf{c}}^{\mathrm{T}} \mathbf{x} + du \end{aligned} \tag{6}$$

for the case of a 4[th]-order system, with eigenvalues $\lambda_i = \delta_i + \mathrm{j}\omega_i$ and j being the imaginary unit. The input and output vectors $\mathbf{b}$ and $\mathbf{c}^{\mathrm{T}}$ are transformed by the matrix of eigenvectors $\mathbf{V}$ to $\tilde{\mathbf{b}} = \mathbf{V}^{-1}\mathbf{b}$ and $\tilde{\mathbf{c}}^{\mathrm{T}} = \mathbf{c}^{\mathrm{T}}\mathbf{V}$.

B. Neural Networks

This paper utilizes feed-forward artificial neural networks that consist of multiple neuron layers which transform an input signal into an output signal, depending on a chosen architecture in combination with a weighting matrix. The input layer distributes the incoming signal to every neuron of the first hidden layer. In all hidden layers and in the output layer every neuron calculates an output $o \in \mathbb{R}$ based on

$$o = \sigma \left(\sum_{k=1}^{q} i_k w_k + w_0 \right) = \sigma \left(\mathbf{i}^{\mathrm{T}} \cdot \mathbf{w} \right) \ , \tag{7}$$

where $\mathbf{i} = [i_0, i_1, \ldots, i_q]^{\mathrm{T}} \in \mathbb{R}^{q+1}$ is the incoming signal from the preceding layer with q neurons and a constant $i_0 = 1$. $\mathbf{w} = [w_0, w_1, \ldots, w_q]^{\mathrm{T}} \in \mathbb{R}^{q+1}$ is the weighting vector of the considered neuron, with w_0 being the bias. $\sigma(\cdot)$ is a nonlinear transformation called activation function.

Besides the presented neuron, there are various more types, such as the 1-dimensional convolutional neuron [6]. The weights in a convolutional neuron are packed in a vector $\mathbf{w} \in \mathbb{R}^r$ of fixed size with $r < q$. In a single layer each neuron reacts only to a fraction of its input, which leads to better adaptions of local patterns. Additionally the weights are shared across a single layer, so the same transformation is applied to different parts of the input vector. Mathematically this corresponds to a convolution of the weight vector

$$(\mathbf{w} * \mathbf{i})(x) = \sum_k \mathbf{w}(k) \mathbf{i}(x - k) \tag{8}$$

with $\mathbf{i} \in \mathbb{R}^{q+1}$ being the input [1]. Afterwards a nonlinear activation function is applied.

The training of a neural network corresponds to adapting the weights in such a way that a specific loss function is minimized. Usually, this training is performed by means of a gradient descent-based optimization in a supervised fashion with labeled data.

III. Autoencoder-related Identification

We want to train a neural network which calculates the inverse way of Equation (3) in order to estimate the continuous SISO state space representation. Therefore, we measure the output of the unknown system, stimulated by an arbitrary input $\mathbf{u} = [u_1, \ldots, u_m]$ at m discrete time steps, leading to a time series $\mathbf{y} = [y_1, \ldots, y_m]$. The neural network maps $\mathbf{y}$ to a state space representation by a projection $f(\cdot)$ according to

$$f(\mathbf{y}) = \begin{pmatrix} \hat{\mathbf{A}} & \hat{\mathbf{b}} \\ \hat{\mathbf{c}}^{\mathrm{T}} & \hat{d} \end{pmatrix} \ . \tag{9}$$

The hat symbol denotes estimated parameters.

Neural networks are usually trained by a supervised approach which means that both, input and output data are necessary to acquire the correct projection. The system matrices, which are supposed to be the output of the neural network, are unknown a priori, so we use an approach based on the principle of an autoencoder. Autoencoders are often used to encode and compress data. They consist of

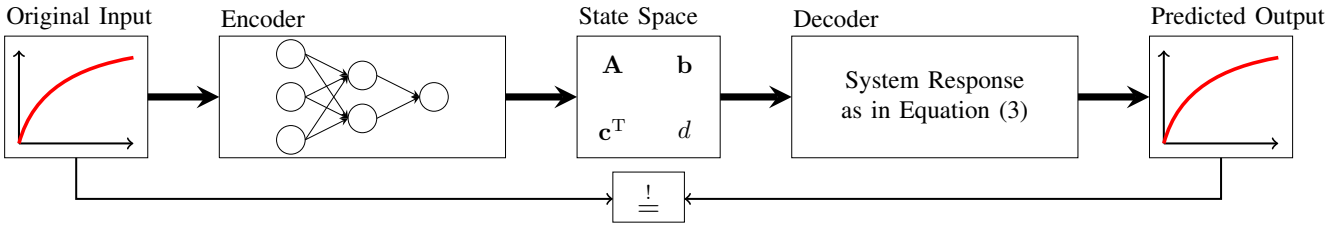


Fig. 1: Sketch of the approach: Measured system data is fed into the neural network (encoder), which transforms it to a state space representation. The system response function generates an output signal, based on the estimated parameters (decoder). The network's weights are adapted, so the output signal of the system response is as similar as possible to the input signal.

a symmetric, sandglas-like neural network architecture with a smaller layer in its center, which is an embedding of a lower dimension compared to the input. The first half of the network, named encoder, reduces the dimension of the input, until it meets the dimension of the embedding. Afterwards the second half, named decoder, processes the encoder-output and increases its dimension in order to find a representation that is as close as possible to the original input.

Instead of implementing a full autoencoder, we propose a variant: The encoder is a common neural network with decreasing layer-size. Subsequently, the decoder is represented by Equation (3) that can be implemented in a machine learning framework. We feed the measured data into the neural network which outputs an estimate of a corresponding state space. Afterwards, we calculate an output signal from this system and compare it to the original input. Hence, we train the neural network in such a way that its input and the "decoder's" output are equal. The full principle is shown in Fig. 1.

Once the neural network is trained, it only needs to perform a single forward pass for identifying a system, which is computationally efficient. This is especially useful when performing system identification online without having huge computational resources. We want to point out two more features of our approach. Firstly, we can identify continuous-time state space representations directly. If we use a 4SID-method, we need to transform the identified discrete state space to a continuous one separately, which increases the execution time. Secondly, we can handle partly identified systems, since we can always specify elements that we want to estimate. If, for example, single elements of the system matrix $\mathbf{A}$ are already known, we can exclude them from the estimation.

A. Neural Network Architecture

Since we want to create an encoder neural network, the input dimension is much larger than the output dimension. This issue is related to a time series classification problem, where neural networks are trained to classify different time series. Residual neural networks prove to be successful in this domain [9]. They feature a special architecture where certain connections skip a subsequent layer. Our network for system identification is built by three sequential 1-D-convolution layers. Additionally, we implement another 1-D-convolution layer which shortcuts the three sequential ones. The resulting outputs are concatenated and fed to a standard dense layer. The numbers of the different neurons can be adapted according to the complexity of the problem and the available computing resources. By having neurons that process time series as input layers and standard dense neurons as output layers, we obtain the sandglas-like architecture, which is needed for an autoencoder.

B. Usage of a Canonical State Space Representation

As already stated in Section II-A, state space representations are not unique. A different dynamic matrix $\mathbf{A}$ can lead to an identical behavior. Because of this fact and to prevent the necessity of predicting all values of $\mathbf{A}$, we use canonical state space representations. We evaluate our observations regarding the different system representations.

The CCF leads to fewer estimated parameters. The dynamic matrix $\mathbf{A}$ is built from the coefficients of the associated Laplacian transfer function. Additionally, the input vector $\mathbf{b}$ is constant, so only the output vector $\mathbf{c}^{\mathrm{T}}$ and the last row of $\mathbf{A}$ have to be estimated. For a system of order n, this leads to $2n$ values that have to be estimated. However it is not possible to determine the system's stability just by looking at the values of $\mathbf{A}$, in general. This leads to the possibility of estimating a system with exponentially growing outputs. After performing some tests, we observed an unstable learning behavior as well, which is reflected by exploding gradients during training. Nevertheless it is possible to avoid systems with real positive eigenvalues by using Descarte's rule:

> "An algebraic equation $f(s) = a_0 + a_1 s + \ldots + a_n s^n$ with real coefficients cannot have more positive real roots than the sequence $a_0, \ldots, a_n$ has variations of sign." [3]

The coefficients of $\mathbf{A}$'s characteristic polynomial are equal to the negative last row of the CCF. It can be shown that if the last row of $\mathbf{A}$ includes no sign changes, then there are no real positive eigenvalues. It is still possible that complex conjugate eigenvalues will have a positive real part leading to unstable systems, but excluding an unstable subspace from scratch simplifies the optimization problem significantly.

In the case of the MCF, the eigenvalues of $\mathbf{A}$ as well as the input and output vectors $\mathbf{b}$ and $\mathbf{c}^{\mathrm{T}}$, respectively, have to be estimated. Since the dynamic behavior of the system depends mostly on the eigenvalues, there is a direct

correlation between the measured behavior and the estimated eigenvalues. Preventing the estimation of unstable systems is easily achieved by constraining the space of the predicted eigenvalue's real parts to the negative half. A drawback of this choice is the large number of parameters which have to be estimated. For a system of rank n, we have to estimate n eigenvalues, as well as n values of the input and output vectors $\mathbf{b}$ and $\mathbf{c}^{\mathrm{T}}$, respectively, leading to $3n$ values.

C. Implementation

We aim at directly identifying continuous-time state space representations. However, measurements are discrete in time. The forced response of a continuous-time system can be calculated by performing a step-wise discretization in order to get the state $\mathbf{x}(dt)$, with dt being the constant time elapsed between two subsequent measurements, by means of

$$\begin{bmatrix} \hat{\mathbf{x}}(dt) \\ u(dt) \\ \Delta u(dt) \end{bmatrix} = \operatorname{expm}\left(\begin{bmatrix} \hat{\mathbf{A}} \cdot dt & \hat{\mathbf{b}} \cdot dt & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \right) \begin{bmatrix} \hat{\mathbf{x}}(0) \\ u(0) \\ \Delta u(dt) \end{bmatrix} , \tag{10}$$

according to [4, 15]. $\mathbf{0}$ and $\mathbf{I}$ denote the zero and the identity matrix, respectively, of appropiate dimension and $\operatorname{expm}(\cdot)$ represents a matrix exponential. Additionally

$$\Delta u(dt) = u(k \cdot dt) - u((k-1) \cdot dt) \tag{11}$$

is the difference between the input signals of two subsequent time steps with $k \in \mathbb{N}$. This is performed m times, so an estimated state matrix is obtained according to

$$\hat{\mathbf{X}} = \begin{bmatrix} \hat{\mathbf{x}}(0) & \hat{\mathbf{x}}(dt) & \cdots & \hat{\mathbf{x}}(m \cdot dt) \end{bmatrix} . \tag{12}$$

The output signal $\hat{\mathbf{y}}$ can then be calculated by means of

$$\hat{\mathbf{y}} = \hat{\mathbf{c}}^{\mathrm{T}} \cdot \hat{\mathbf{X}} + \hat{d}\mathbf{u} . \tag{13}$$

We implement Equations (10)-(13) in the well-known machine learning framework TensorFlow.

Besides an identical output sequence, we also want to enforce that the identified system has the same stationary value as the original one. For this purpose, we define the training loss function

$$J = \frac{1}{m+1} \left(\sum_{i=1}^{m} \left(\mathbf{y}_i - \hat{\mathbf{y}}_i \right)^2 + \gamma \left(y_\infty - \hat{y}_\infty \right)^2 \right) \tag{14}$$

that comprises the mean squared error of the predicted system output and an additional penalty term quantifying the squared deviation between the true stationary value and the predicted one. Here, $\gamma \in \mathbb{R}^+$ denotes a hyper-parameter for weighting the influence of the stationary value's deviation.

Normalizing the data is crucial when we want to train a machine learning algorithm [12]. In the case of the initial state being $\mathbf{x}_0 = \mathbf{0}$, we choose

$$\tilde{\mathbf{y}} = \frac{2}{\overline{y}} \mathbf{y} - 1 \tag{15}$$

as a normalization scheme, where $\overline{y} = \max |\mathbf{y}|$. Doubling and shifting the whole measurement signals leads to the creation of both, positive and negative values. It is important to apply this normalization not only to the data, but also to the output of the implemented TensorFlow function in Equation (13). Common normalization techniques include a varying vertical shift instead of the selected fixed one. We chose to not follow this approach, since the denormalization of the identified state space representation would become nontrivial. This is also the reason behind the restriction of $\mathbf{x}_0 = \mathbf{0}$, since a non-zero initial state leads to the necessity of a varying shift over the training batch.

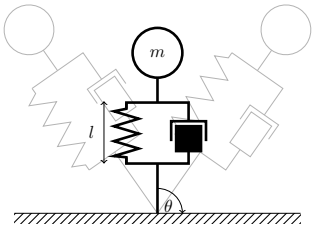


Fig. 2: Spring-loaded, damped inverted pendulum as a simplification of a robot leg [2].

IV. Online System Identification of a Robot Leg

For validating our approach, we present an example use case by identifying the wear of a robot leg online. It consists of an inverted, damped spring-mass pendulum, with the control target of keeping it stable in the upright position. The system is depicted in Figure 2. While the system has already been identified and controlled in various publications [2], it is not clear whether the constructed controllers will still work after a long time, when the parameters of the system might change due to wear. This applies especially to the stiffness parameter k_l of the spring as well as to the damping coefficient d_l. A third important parameter that might vary, is the mass m_l of the robot, e.g., if it carries a load. Measuring these values during usage of the system is not practical. Thus we use our system identification method for identifying a change in the state space online.

We implement the system in Python and simulate the step response for different parameters d_l, k_l and m_l. If the system dynamics are unknown in general, the proposed method also works by measuring one input-output-relation and training the neural network only on this data instance. Another way could be to perform a standard subspace-based state space identification (command `n4sid` in MATLAB). Both require the estimation of the system rank.

In the present case we already know the system dynamics, so we can use them to generate training data for an online identification. They are described by a system of differential equations, linearised at $l = l_0$, $\dot{l} = 0$, $\theta = \frac{\pi}{2}$ and $\dot{\theta} = 0$:

$$\begin{pmatrix} \dot{l} \\ \ddot{l} \\ \dot{\theta} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\frac{k_l}{m_l} & -\frac{d_l}{m_l} & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{g}{L} & 0 \end{pmatrix} \begin{pmatrix} l \\ \dot{l} \\ \theta \\ \dot{\theta} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ \frac{1}{m_l} & 0 \\ 0 & 0 \\ 0 & \frac{1}{m_l L^2} \end{pmatrix} \mathbf{u} \tag{16}$$

TABLE I: Error values of the performed validations for the stable and unstable system. The step signal errors are calculated by the mean absolute difference after a normalization between $[0, 1]$, while magnitude and phase are mean absolute differences without a normalization. Parameter errors are given in percentage.

	Stable		**Unstable**	
Signal Type	Scenario	Overall	Scenario	Overall
Step Signal ($\cdot 10^{-3}$)	9.01	14.82	15.70	31.50
Magnitude (dB)	0.73	0.89	4.03	3.37
Phase ($\cdot 10^{-3}$ rad)	39.72	28.46	0	0
Δk_l (in %)	1.97	11.68	24.19	26.36
Δd_l (in %)	3.49	8.67	-	-
Δm_l (in %)	3.89	14.77	50.13	48.32

according to [17], with $L = l_0 - \frac{m_l g}{k_l}$ and g being the gravitational constant. This linearisation is reasonable, since the control target is to keep the inverted pendulum upright. We can see, that we have to identify two decoupled SISO systems which comprise the two degrees of freedom: the translatory forces, located in the damped spring-mass-pendulum and the rotatory forces, located in the rotating inverted pendulum. The latter one is unstable, since stimulating it in its idle state will make it fall down (with a cap at π, since it cannot fall further). The damped spring-mass-pendulum on the other hand is a stable, oscillating system.

For the online identification we build two neural networks, which correspond to the CCF for both systems. We simulate 10^4 different step responses by varying the parameters between $k_l \in [20, 25]$, $d_l \in [2, 5]$, $m_l \in [0.5, 1.5]$ and setting the length of the uncompressed spring to $l_0 = 0.8\,\mathrm{m}$. We choose a residual neural network architecture with three 1-D-convolutional layers of eight neurons, followed by two dense layers of ten neurons and train the neural networks by keeping $20\,\%$ of the data for testing the found weight combination online which prevents overfitting.

For a validation we construct another data set which includes 500 new parameter combinations. To demonstrate the results, we pick one parameter scenario out of this data set and calculate the associated step responses. It has a setting of $k_{l,1} = 24.23\,\frac{\mathrm{N}}{\mathrm{m}}$, $d_{l,1} = 2.24\,\frac{\mathrm{kg}}{\mathrm{s}}$ and $m_{l,1} = 1.01\,\mathrm{kg}$. Afterwards, we feed our neural networks with this simulated data, which gives us an estimated state space representation of

$$\begin{pmatrix} \dot{l} \\ \ddot{l} \\ \dot{\theta} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -23.7 & -2.2 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 18.4 & 0 \end{pmatrix} \begin{pmatrix} l \\ \dot{l} \\ \theta \\ \dot{\theta} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1.0 & 0 \\ 0 & 0 \\ 0 & 7.0 \end{pmatrix} \mathbf{u} \; .$$

Figure 3 shows the step response and bode plots of the identified systems, compared to the step responses of the estimated ones.

We can see that identifying a stable and oscillating system leads to a reasonable estimation, while the unstable system is less precise. This is reasoned by slight changes in the state space of an unstable system which lead to heavy deviations in the step response. The bode plots show that the identified stable system behaves very similar to the corresponding original one, while the unstable system shows a diverge, especially in the low frequency areas. The phase of the unstable system is constant at π, since there is no imaginary part in this subsystem's transfer function. Table I lists the corresponding error values, including the errors over all 500 random validation systems which are in the same region as the presented scenario. Additionally, the table shows the parameter errors, based on the identified system in a mean absolute percentage deviation. We can see that our introduced identification method enables systems that behave equally over all frequencies but deviate in the identified parameters. This is caused by fractions in (16) which compensate for false parameters. In general, neural networks are especially useful for interpolation tasks like the presented one. Extrapolation is only achieved under special conditions [20] which is why the identification method will fail as soon as the training data does not represent the area of application anymore. Usually neural networks perform better with more training data. Figure 4 shows a plot, where we depict the step signal error of the same architecture but with different data sizes (ranging from 100 to 10^5). Although the error decreases with more training samples, we already reach a reasonable estimation by using only 100 systems for training.

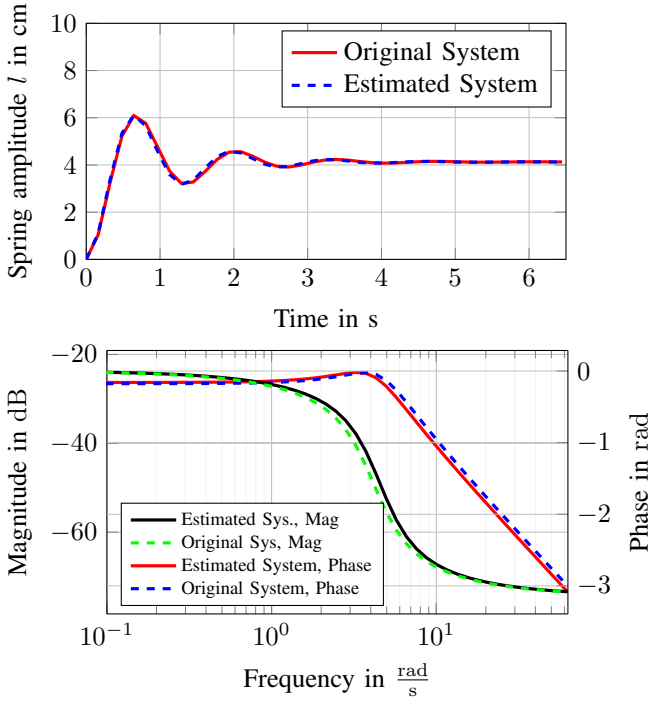
We measured the execution time in Python on a weak processor by limiting our device to $1 \times 400\,\mathrm{MHz}$. This leads to an elapsed time of $1.194\,\mathrm{s}$ for a single system identification. Since we want to compare the method to the `n4sid`-function in MATLAB, we ran an equivalent network in a MATLAB environment and reached roughly the same execution time ($1.329\,\mathrm{s}$). The execution time of the `n4sid`-function, including the transformation to a continuous state space model, is $3.682\,\mathrm{s}$. We want to point out though that the presented identification method requires an offline training session which is computationally expensive, while the `n4sid`-function works online as a whole. Additionally the latter technique is far more precise (the normalized mean absolute error of the step responses is smaller than 10^{-15}).

V. CONCLUSIONS

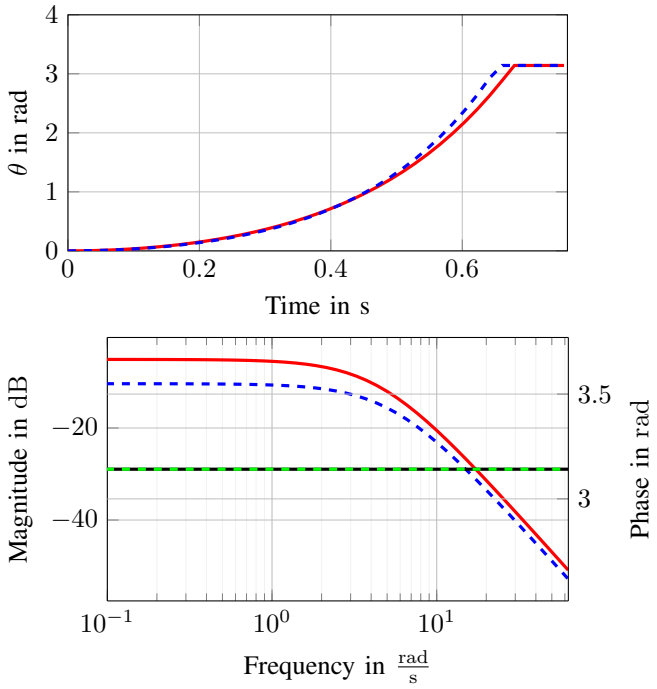
We introduced a new method to identify the state space representation of a system online. While the training is computationally expensive, only a single forward pass is necessary to identify the system, which leads to a runtime efficient code. Nevertheless, the main contribution of this method can be seen as a base line for identifying nonlinear systems with machine learning methods.

ACKNOWLEDGMENT

This work was partially supported by the State Ministry of Baden-Wuerttemberg for Economic Affairs, Labour and Housing Construction (KI-Fortschrittszentrum "Lernende Systeme", Grant No. 036-170017).



(a) Step response and bode plot of the stable system.



(b) Step response and bode plot of the unstable system.

Fig. 3: Step response and bode plots of the a parameter scenario with $k_{l,1} = 24.23\,\frac{\text{N}}{\text{m}}$, $d_{l,1} = 2.24\,\frac{\text{kg}}{\text{s}}$ and $m_{l,1} = 1.01\,\text{kg}$.

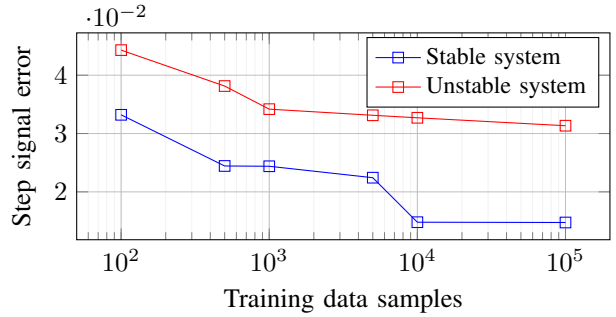


Fig. 4: Step signal error over different training data sizes.

References

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. IEEE, 21.08.2017 - 23.08.2017, pp. 1–6.

[2] Hua Chen, Patrick M. Wensing, and Wei Zhang. *Optimal Control of a Differentially Flat 2D Spring-Loaded Inverted Pendulum Model*. 2019. arXiv: 1911.07168.

[3] D. R. Curtiss. "Recent Extentions of Descartes' Rule of Signs". In: *The Annals of Mathematics* 19.4 (1918), p. 251.

[4] R. A. DeCarlo. *Linear Systems: A State Variable Approach with Numerical Implementation*. USA: Prentice-Hall, Inc., 1989.

[5] MP. Deisenroth and CE. Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*. Omnipress, 2011, pp. 465–472.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[7] Moritz Hardt, Tengyu Ma, and Benjamin Recht. *Gradient Descent Learns Linear Dynamical Systems*. 2019. arXiv: 1609.05191.

[8] Marco F. Huber. "Recursive Gaussian process: Online regression and learning". In: *Pattern Recognition Letters* 45 (2014), pp. 85–91.

[9] Hassan Ismail Fawaz et al. "Deep learning for time series classification: a review". In: *Data Mining and Knowledge Discovery* 33.4 (2019), pp. 917–963.

[10] I. W. Jamaludin et al. "N4SID and MOESP subspace identification methods". In: *2013 IEEE 9th International Colloquium on Signal Processing and its Applications*. IEEE.

[11] S. L. Lacy and D. S. Bernstein. "Subspace identification with guaranteed stability using constrained optimization". In: *IEEE Transactions on Automatic Control* 48.7 (2003).

[12] Yann A. LeCun et al. "Efficient BackProp". In: *Neural Networks: Tricks of the Trade*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Vol. 7700. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[13] Concepción A. Monje et al. *Fractional-order Systems and Controls: Fundamentals and Applications*. Advances in Industrial Control. London: Springer-Verlag London, 2010.

[14] A. Nagabandi et al. *Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning*. 2018.

[15] *Python Control Systems Library*. 2020. URL: https://github.com/python-control/python-control/blob/master/control/timeresp.py.

[16] Tuhin Sarkar et al. *Nonparametric Finite Time LTI System Identification*. arXiv: 1902.01848.

[17] T. Turner Topping et al. "Towards bipedal behavior on a quadrupedal platform using optimal control". In: *Unmanned Systems Technology XVIII*. 2016.

[18] Peter van Overschee and Bart de Moor. *Subspace Identification for Linear Systems*. Boston, MA: Springer US, 1996.

[19] Jeen-Shing Wang and Yen-Ping Chen. "A fully automated recurrent neural network for unknown dynamic system identification and control". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 53.6 (2006), pp. 1363–1372.

[20] Keyulu Xu et al. *How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks*. 2021. arXiv: 2009.11848.

[21] Wenhao Yu et al. *Preparing for the Unknown: Learning a Universal Policy with Online System Identification*. 2017. arXiv: 1702.02453.