

Identifying Ordinary Differential Equations for Data-efficient Model-based Reinforcement Learning

Tobias Nagel* and Marco F. Huber*[†]

*Fraunhofer Institute for Manufacturing Engineering and Automation IPA, 70569 Stuttgart, Germany
tobias.nagel@ipa.fraunhofer.de

[†]Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, 70569 Stuttgart, Germany
marco.huber@ieee.org

Abstract—The identification of a mathematical dynamics model is a crucial step in the designing process of a controller. However, it is often very difficult to identify the system’s governing equations, especially in complex environments that combine physical laws of different disciplines. In this paper, we present a new approach that allows identifying an ordinary differential equation by means of a physics-informed machine learning algorithm. Our method introduces a special neural network that allows exploiting prior human knowledge to a certain degree and extends it autonomously, so that the resulting differential equations describe the system as accurately as possible. We validate the method on a Duffing oscillator with simulation data and, additionally, on a cascaded tank example with real-world data. Subsequently, we use the developed algorithm in a model-based reinforcement learning framework by alternately identifying and controlling a system to a target state. We test the performance by swinging-up an inverted pendulum on a cart.

Index Terms—Kalman filtering, Neural nets, Ordinary Differential Equations, Nonlinear approximation

I. INTRODUCTION

Identifying the characteristics and behavior of a system is a vital step for predicting and controlling its future states. In dynamic systems, the governing equation is often represented by means of an ordinary differential equation (ODE) that contains derivatives with respect to time. However, in many cases it is very difficult to derive the equations that describe the underlying physical laws because large systems in particular exhibit complex behavior. In this paper, we present a new method that allows identifying a nonlinear ODE from noisy input-output data, which is applicable in a model-based Reinforcement Learning (RL) framework. Our method allows human knowledge to be incorporated when available, making it a hybrid machine learning approach.

In the following, we give a formal problem statement. The state-space representation of a continuous-time, nonlinear, dynamic, stochastic, and time-variant system of rank $n \in \mathbb{N}$ is given by

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{w}(t), t), \quad \mathbf{x}(t_0) = \mathbf{x}_0, \\ \mathbf{y}(t) &= \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), t), \end{aligned} \quad (1)$$

with the nonlinear ODE $\mathbf{f}(\cdot)$ and the measurement function $\mathbf{g}(\cdot)$. $\mathbf{x}(t) \in \mathbb{R}^n$ denotes the state vector with an initial

value \mathbf{x}_0 . $\mathbf{u}(t) \in \mathbb{R}^p$ and $\mathbf{y}(t) \in \mathbb{R}^q$ denote the input and output signal with dimensions $p, q \in \mathbb{N}$, respectively. The vectors $\mathbf{w}(t)$ and $\mathbf{v}(t)$ denote white process noise and white measurement noise, respectively, which are both assumed to be zero-mean Gaussian with covariance matrices $\mathbf{Q}(t) \in \mathbb{R}^{n \times n}$ and $\mathbf{R}(t) \in \mathbb{R}^{q \times q}$, respectively. We assume that we can acquire noisy measurements $\bar{\mathbf{y}}(t_i)$ of the system at N discrete time steps t_i with $i = 1, \dots, N$.

Depending on the use case, we distinguish three situations: (i) \mathbf{f} and \mathbf{g} are known, apart from some real-valued parameters. (ii) \mathbf{f} and \mathbf{g} are partly known, which means that it is possible to model single equation elements, but not the system as a whole. (iii) \mathbf{f} and \mathbf{g} are unknown, apart from the system states and the system rank. The aim of our method is to identify $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ in such a way that

$$J = \sum_{i=1}^N \|\mathbf{y}(t_i) - \bar{\mathbf{y}}(t_i)\|_{\mathbb{F}} \quad (2)$$

is minimal, with $\|\cdot\|_{\mathbb{F}}$ being the Frobenius norm.

In this paper, we present a new method to identify ODEs that we call *ODE-Learner*. We use an extended Kalman-Bucy-Filter (EKBF) (cf. Sec. III-A), which is a continuous-time and nonlinear implementation of the famous Kalman filter, consisting of two ODEs for estimating the mean value and the covariance matrix of the system state. As $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ are assumed to be (partly) unknown, the EKBF cannot be applied directly. Instead, both ODEs of the EKBF are realized as neural networks in a Physics-Informed Neural Network (PINN)-framework (cf. Sec. III-B). The governing equation of the ODE is learned using two further neural networks in an Equation Learner (EQL)-framework (cf. Sec. III-C).

A field of application is model-based RL, where an agent collects data in an environment and trains a model that allows predicting the future system behavior. The model is then used to calculate a reward-maximizing policy, which is applied to the agent. Thus, more data can be gathered and the model is re-trained to improve its accuracy until, finally, a target state is reached. This procedure is often more data-efficient than model-free alternatives [8]. We show how the ODE-Learner can be used within continuous-time model-based RL to learn the dynamics model of the agent. The key contributions of our method comprise:

This work was supported by the Baden-Wuerttemberg Ministry for Economic Affairs, Labour and Tourism (project KI-Fortschrittszentrum “Lernende Systeme und Kognitive Robotik”, grant no. 036-140100).

- Identifying an ODE from noisy data and not directly accessible system states by combining an EKBF with PINNs and EQLs.
- Extending the EQL framework to create more shallow networks.
- Allowing to incorporate human knowledge if available.
- Using the identified ODE in a model-based RL environment.

Please note that opposed to many existing methods, we do not simplify the identification process by fitting a discrete-time difference equation, but train in a fully continuous-time fashion, which features several advantages. Firstly, model building typically relies on physical equations that describe dynamics in continuous time. The continuous-time identification approach allows the direct integration of existing system knowledge into the training algorithm. Secondly, continuous-time identification eliminates the need for a pre-defined sampling rate. In contrast, discrete-time identification necessitates setting a sampling rate, which must be large enough to accurately reconstruct the system dynamics, yet not so large that the resulting measurement data exceeds available memory limits.

II. RELATED WORK

As described in [1], the system identification process comprises five major steps: acquiring data, choosing a model class, determining its structure, estimating the parameters and, finally, validating the result. In the following literature review, we will focus on the different model classes, since most other identification steps depend on this crucial choice.

Input-Output Models. Popular approaches to identify nonlinear input-output behavior comprise for example radial basis functions [29], Volterra-Series [12], or the usage of an arbitrary nonlinear function in a nonlinear autoregressive model with exogenous inputs (NARX) [15]. Another possibility is the usage of established machine learning techniques, such as a decision tree [27] or a Gaussian process to model a state space representation of the system [26]. However, these methods are not designed for identifying an ODE, but rather create a model for the input-output behavior and, thus, can identify a discrete-time difference equation at the most. Many control algorithms, however, require an ODE instead of a discrete-time difference equation.

Parameter Identification. Identifying single parameters in an otherwise known ODE is mostly solved by using various optimization algorithms. This task is sometimes referred to as *inverse problem* and can be solved by gradient-based [18], [23] or gradient-free [2], [24] optimization algorithms. However, these algorithms often fail, if the ODE exhibits chaotic, stiff, or highly nonlinear behavior, since it is necessary to solve the ODE numerically in every optimization step. Machine learning algorithms, such as PINNs allow a parameter identification without calculating a numerical solution [11], [20].

Identification of a Partly Known System. In many cases the governing ODE of a system is merely known to a certain extent, but it is not sufficiently precise for a real-world scenario. An example for this could be the modeling of a

motor without considering the friction. It is possible to use a genetic algorithm to first estimate the parameters of the known system part and then extend it by a Gaussian process to model behavior, which goes beyond the scope of the governing equation [4]. Another popular system identification method is Sparse Identification of Nonlinear Dynamics (SINDy) [3]. It trains a matrix that comprises the coefficients of many nonlinear operators, which are used to construct an ODE. The algorithm is well-established and has been improved to perform better under noise or in a control environment [9]. A similar approach is given by the so-called EQL, which replaces the standard activation functions of a neural network, like the hyperbolic tangent or a rectified linear unit, with other nonlinear functions that frequently appear in physical laws [25]. Both, the SINDy-method and the EQL can benefit from prior system knowledge, by pre-conditioning the used operators.

Identification of an Unknown System. If only the system rank is known, the so-called subspace-based state-space estimation allows an identification of a linear state-space model by means of a QR-factorization as well as a singular value decomposition [30]. Linear models can even be used in reinforcement learning [5], but will suffer if the process features a strong nonlinear behavior. A more recent approach for identifying nonlinear models is the use of neural ODEs [6], [16], which allow a neural network to approximate an unknown ODE by embedding it into a numerical ODE-Solver and use the resulting solution to adjust the network weights based on training data. In [34], this method is combined with an EQL, in order to allow the inclusion of symbolic elements in a neural ODE-framework. However, this approach depends on the step width of the numerical ODE-Solver and the identification quality degrades severely if the necessary step width becomes very small, hence, if the ODE's solution is stiff or highly oscillating. The inverse way of using integration instead of differentiation to fit an ODE is presented in [17] and is called Integrated Neural Network (INN).

III. FUNDAMENTALS

In this section, we present three fundamental methods, which our approach combines in order to identify an ODE from noisy measurements: The EKBF is a minimum variance estimator that allows estimating a system's states as precisely as possible. A PINN allows approximating an ODE's solution without the usage of numerical solvers. Finally, an EQL creates a special type of neural network, which uses operators that appear frequently in differential equations. Besides these three methods we briefly introduce model-based RL.

A. Extended Kalman-Bucy-Filter

A Kalman filter allows estimating the system's state from noisy measurements and a given state space model. While the commonly known Kalman filter is limited to linear and discrete-time systems, the EKBF performs state estimation for continuous-time and nonlinear systems [14]. It does so by establishing two initial value problems, which describe the

temporal evolution of the state's estimated mean value and covariance matrix. The mean value is given by

$$\begin{aligned} \dot{\hat{\mathbf{x}}}(t) = & \mathbf{f}(\hat{\mathbf{x}}(t), \mathbf{u}(t), \mathbf{0}, t) \\ & + \mathbf{K}(t) \cdot (\bar{\mathbf{y}}(t) - \mathbf{g}(\hat{\mathbf{x}}(t), \mathbf{u}(t), \mathbf{0}, t)) \end{aligned} \quad (3)$$

with a known initial value $\hat{\mathbf{x}}(t_0) = \hat{\mathbf{x}}_0$ and a Kalman gain

$$\mathbf{K}(t) = \hat{\mathbf{P}}(t) \cdot \hat{\mathbf{C}}^T(t) \cdot \hat{\mathbf{R}}^{-1}(t) \quad (4)$$

as well as its covariance matrix

$$\begin{aligned} \dot{\hat{\mathbf{P}}}(t) = & \hat{\mathbf{A}}(t)\hat{\mathbf{P}}(t) + \hat{\mathbf{P}}(t)\hat{\mathbf{A}}(t)^T \\ & - \mathbf{K}(t)\hat{\mathbf{C}}^T(t)\hat{\mathbf{P}}(t) + \hat{\mathbf{Q}}(t) \end{aligned} \quad (5)$$

with a known initial value $\hat{\mathbf{P}}(t_0) = \hat{\mathbf{P}}_0$ and linearized system matrices

$$\begin{aligned} \hat{\mathbf{A}}(t) = & \left. \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}, t)}{\partial \mathbf{x}(t)} \right|_{\wedge}, & \hat{\mathbf{G}}(t) = & \left. \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}, t)}{\partial \mathbf{w}(t)} \right|_{\wedge}, \\ \hat{\mathbf{C}}(t) = & \left. \frac{\partial \mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{v}, t)}{\partial \mathbf{x}(t)} \right|_{\wedge}, & \hat{\mathbf{V}}(t) = & \left. \frac{\partial \mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{v}, t)}{\partial \mathbf{v}(t)} \right|_{\wedge}. \end{aligned} \quad (6)$$

The \wedge -symbol denotes that the linearization is performed repeatedly for each new mean value $\hat{\mathbf{x}}(t)$. The covariance matrices of the process and measurement noise of the linearized system are given by means of

$$\begin{aligned} \hat{\mathbf{Q}}(t) = & \hat{\mathbf{G}}(t) \cdot \mathbf{Q}(t) \cdot \hat{\mathbf{G}}^T(t), \\ \hat{\mathbf{R}}(t) = & \hat{\mathbf{V}}(t) \cdot \mathbf{R}(t) \cdot \hat{\mathbf{V}}^T(t), \end{aligned} \quad (7)$$

respectively.

B. Physics-Informed Neural Network

PINNs are a subclass of neural networks, which constrain the network's output to some known physical dynamics, which is usually given as a differential equation. They have been introduced by Raissi et al. [21] for partial differential equations. In the following however, we define them in accordance to the state space representation of (1). Let $\mathcal{N}_\pi(t, \mathbf{u}|\mathbf{W}) : \mathbb{R}^{p+1} \rightarrow \mathbb{R}^n$ be a neural network, which maps the time t and system input \mathbf{u} to an estimated state vector $\tilde{\mathbf{x}}(t)$ by means of $\tilde{\mathbf{x}}(t) = \mathcal{N}_\pi(t, \mathbf{u}|\mathbf{W})$. Then, the neural network's weights \mathbf{W} are trained by minimizing the loss

$$\begin{aligned} J_{\text{PINN}} = & \sum_{i=1}^N \left(\frac{d}{dt} \tilde{\mathbf{x}}(t_i) - \mathbf{f}(\tilde{\mathbf{x}}(t_i), \mathbf{u}, t_i) \right)^2 \\ & + (\tilde{\mathbf{x}}(t_0) - \mathbf{x}_0)^2. \end{aligned} \quad (8)$$

Due to the time derivative of the neural network's output $\tilde{\mathbf{x}}(t)$ in (8), it is necessary that $\mathcal{N}_\pi(t, \mathbf{u}|\mathbf{W})$ contains only continuously differentiable activation functions, such as a hyperbolic tangent, which leads to the whole network being continuously differentiable. Note, however, that the original PINN formulation does not consider any noise, neither process noise \mathbf{w} , nor measurement noise \mathbf{v} , in the system. Additionally, it assumes all states to be fully accessible and it does not consider a measurement function $\mathbf{g}(\cdot)$.

C. Equation Learner

The EQL, introduced by Sahoo et al. in [25], learns equations that are suitable for extrapolation and control. The authors show that the method allows identifying a discrete-time state space model by creating a fully-connected neural network, which does not contain the commonly used activation functions such as hyperbolic tangent or rectified linear unit. Instead, they use continuously differentiable functions that appear frequently in governing equations of dynamical systems, such as sine, cosine, or multiplication. The last layer is composed of a division function. In order to keep the system continuously differentiable, the authors force the EQL to output zero, if the denominator goes below a pre-defined threshold and add a penalty term forcing the system's output to avoid discontinuities. To obtain a difference equation with as few functional operators as possible, the authors also include L_1 and L_0 regularizations, which are activated in different phases of the training. However, the EQL does not learn a continuous-time ODE, but a discrete-time difference equation. Additionally, all system states are assumed to be directly accessible. To address these limitations, we extend the EQL in Sec. IV.

D. Model-based Reinforcement Learning

RL is a subtopic of machine learning, which addresses the actions an agent has to take in an environment in order to maximize a reward. The interaction between agent and environment is defined by a Markov Decision Process (MDP), which is a tuple $(\mathcal{X}, \mathcal{U}, \rho(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k), R(\mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{u}_k), \mathbf{x}_0)$ with the set of states \mathcal{X} and the set of system inputs \mathcal{U} [31]. $\rho(\mathbf{x}_{k+1}|\mathbf{x}_k, \mathbf{u}_k)$ describes a probability distribution to reach the successor state \mathbf{x}_{k+1} , provided a current state \mathbf{x}_k and input \mathbf{u}_k . $R(\mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{u}_k)$ is the associated reward to this transition. Both functions are not known in the RL context and are learned implicitly.

The target is to find a control strategy (or *policy*) $K(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^p$ with $\mathbf{u}_k = K(\mathbf{x}_k)$, which maximizes the expected reward $V = \mathbb{E}\{\sum_{i=k}^{\tau} \gamma^i \cdot R(\mathbf{x}_i, \mathbf{u}_i)\}$ with a discount factor $\gamma \in (0, 1)$. In model-based RL, the discrete-time transition function is explicitly trained by using data, which is generated by the agent exploring the environment. The routine is described as follows: First, the agent performs random actions in the environment and, thus, gathers data. Afterwards, a dynamic model is trained and becomes utilized to find a control function $K(\cdot)$, which is often realized as model-based predictive controller (MPC) [33]. After applying the control function to the system, new data is acquired, which is used to improve the model's accuracy and, thus, allows updating the control law. In this work, we use an identified ODE to find an optimal control function.

Note that RL typically considers purely discrete-time problems and models. However, our approach allows the identification of a continuous-time model, which avoids the adverse effects of temporal discretization during the design of a controller.

IV. LEARNING ODES

In the following, we introduce our ODE-Learner framework, which is the main contribution of this paper. Therefore, we build a special EQL-architecture and include it in a PINN-framework to identify the correct ODE. To handle noisy data and modeling errors, we combine the approach with an EKBF. Finally, we integrate several regularization terms in order to acquire an ODE, which is as compact as possible.

A. ODE-Network

The overall target of our method is to identify the nonlinear ODE $\dot{x} = f(x, u, w, t)$ of (1). Therefore, we create a special neural network \mathcal{N}_ζ , which we call *ODE-Network*, to fit the unknown ground truth $f(\cdot)$ as correctly as possible. The main difference between a conventional multi-layer perceptron (MLP) and the ODE-Network is the usage of activation functions, which appear frequently in dynamic systems, such as trigonometric functions or polynomial elements. This idea has first been introduced as EQL in [25] (cf. Sec. III-C). In many systems, the true dynamics arises from the multiplication of several nonlinear operators. To achieve this behavior in the original implementation of the EQL, a very deep network has to be trained. Instead, we create a special EQL-Neuron, which we call *operator-neuron*. It allows building a rather shallow network, at the expense of an increased width, i.e., an increasing neuron count. Afterwards, we combine several operator-neurons to an ODE-Network. This aims at creating an ODE, which is as compact as possible.

Let the operator-neuron's input be denoted by $z = [1, z_1, \dots, z_O]^T \in \mathbb{R}^{O+1}$ with a bias in the beginning and its output by $o \in \mathbb{R}$. Additionally, we define an operator set $\mathcal{O} = \{\text{op}_1(\cdot), \dots, \text{op}_K(\cdot)\}$, which lists K different operators. Note that every operator is required to be continuously differentiable. Finally, the operator-neuron exhibits two weight matrices $\mathbf{W}^{(1)} \in \mathbb{R}^{I \times K \times (O+1)}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{I \times (K+1)}$. One operator-neuron then performs a forward pass by calculating

$$o = \prod_{i=1}^I \left(\mathbf{W}_{i,K+1}^{(2)} + \sum_{k=1}^K \mathbf{W}_{i,k}^{(2)} \cdot \text{op}_k \left(\sum_{l=1}^{O+1} \mathbf{W}_{i,k,l}^{(1)} \cdot z_l \right) \right). \quad (9)$$

The input to the neuron is weighted by the first weighting matrix $\mathbf{W}^{(1)}$ and summed. Afterwards, it is transformed by the set of K operators. The result is then again weighted by the second weighting matrix $\mathbf{W}^{(2)}$ and a bias is added in order to allow a constant activation, if necessary. The operator-neuron performs these calculations I times in parallel and multiplies the scalar results, respectively. Fig. 1a shows a sketch of the calculations that are performed in (9). The final ODE-Network is created by combining multiple operator-neurons in a similar way to a fully-connected neural network with M layers and N_j , $j = 1, \dots, M$, neurons per layer. By using operator-neurons of the proposed structure, a complicated ODE can be built even with a shallow ODE-Network, because of the possibility to directly include multiplication elements. The last layer is composed of two weight vectors $w^{(3)} =$

$[w_0^{(3)}, \dots, w_{N_M}^{(3)}]^T \in \mathbb{R}^{N_m+1}$, $w^{(4)} = [w_0^{(4)}, \dots, w_{N_M}^{(4)}]^T \in \mathbb{R}^{N_m+1}$, which we use to calculate the ratio

$$\hat{x} = \begin{cases} \frac{o^T \cdot w^{(3)}}{o^T \cdot w^{(4)}} & \text{if } o^T \cdot w^{(4)} > \delta \\ 0 & \text{otherwise} \end{cases}, \quad (10)$$

with the output of the preceding layer $o = [1, o_1, \dots, o_{N_M}]^T$ and a hyperparameter $\delta > 0$. This ensures that there is no division by zero. Additionally, we include a penalty term to the loss function, which leads to a higher loss if the denominator is below δ (cf. Sec. IV-B). This idea has first been introduced in [25]. Fig. 1b shows a sketch of the ODE-Network's architecture.

The usage of the ODE-Network features multiple benefits over an MLP. The most important one is the possibility to include prior existing system knowledge by the choice of an operator set. E.g., if we want to model an inverse pendulum system, it makes sense to include trigonometric operators, such as $\mathcal{O} = \{\sin(\cdot), \cos(\cdot)\}$. Using an MLP usually discards this prior knowledge. Depending on our system understanding, it is possible to go even further. In case of situation (i), we do not use an ODE-Network at all, but simply use the known ODE and only train the unknown parameters. Situation (ii) is solved by pre-conditioning the weight matrices of the ODE-Network, based on the existing system knowledge. During the following training, the pre-conditioned ODE-Network is adapted so it suits the provided data. "Adapted" means that the given ODE can be extended by new equation elements and that the initial ODE's parameters can become readjusted. Hence, we use the initial ODE as baseline and improve it continuously during the training. Situation (iii) demands the most training efforts, since, similar to solving a machine learning problem, the architecture needs to be determined by trial-and-error and the operators are set to the best of the user's knowledge. Nevertheless, even without knowing the system dynamics, it is still often possible to make good guesses on the operator set. Another benefit of the ODE-Network over an MLP is the improved extrapolation behavior, which has already been investigated in [25] for the EQL. This is especially important for using the ODE-Network as a model in a model-based RL framework in order to acquire a high reward after as few iterations as possible.

The ODE-Network cannot be trained in a supervised fashion, because it maps the state x , the input u , and the process noise w on the state's temporal derivative \dot{x} . However, \dot{x} is not available in the measurement data and it also cannot be precisely estimated by calculating the difference quotient, especially if only noisy measurements are provided. To circumvent this problem, we use an ODE-Learner framework, which is introduced in the following subsection.

B. The ODE-Learner Framework

The ODE-Learner framework allows fitting an ODE by providing noisy data without simplifying it to a discrete-time difference equation by combining a PINN-approach with

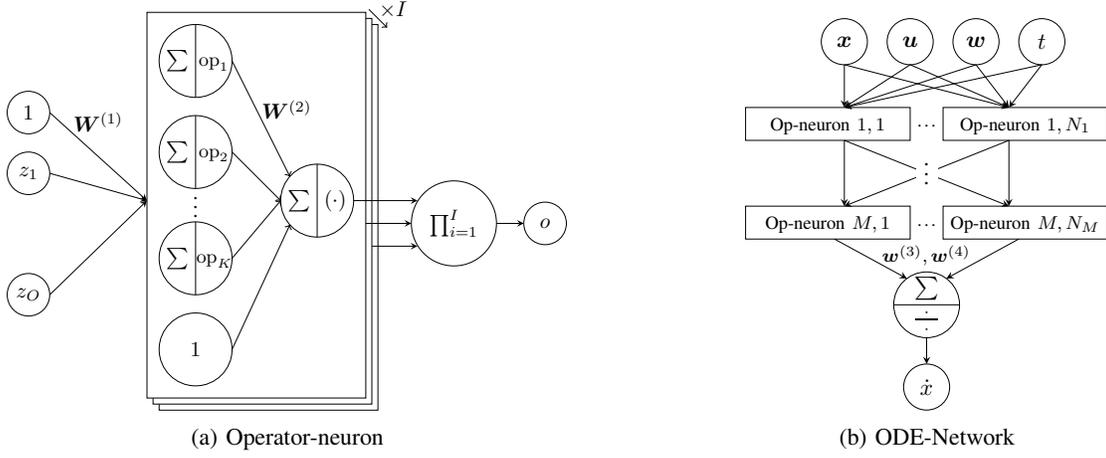


Fig. 1: Sketches of the proposed architecture of a single operator-neuron (left) and of the whole ODE-Network (right).

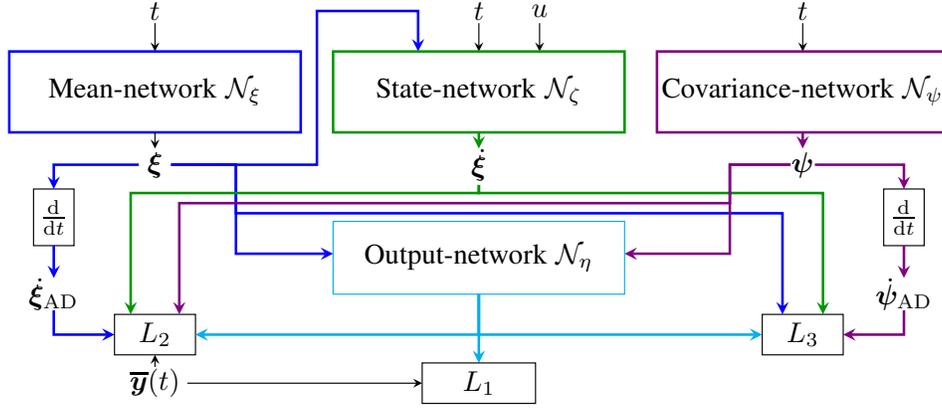


Fig. 2: Sketch of the ODE-Learner concept, describing the interactions of the first three loss functions. The different colors indicate the influence of a network to the respective loss.

an EKBF. The approach to solve an EKBF in a physics-informed way has first been introduced in [20] as Kalman-Bucy-Informed Neural Network (KBINN), with the aim on estimating unknown real-valued parameters in an otherwise known state-space model. In this section, we extend the KBINN by including the ODE-Network in order to identify a continuous-time, nonlinear system with noisy measurements. Here, the ODE-Network estimates the dynamics in the form of a state-space representation. However, in order to fit a state-space representation to data, the ODE of the state needs to be solved, which is done by the KBINN.

We implement four neural networks in total as shown in Fig. 2. The *mean-network* \mathcal{N}_ξ is a standard fully-connected neural network that approximates the state's mean vector by means of $\hat{x}(t) \approx \xi(t) = \mathcal{N}_\xi(t)$ with the network's output $\xi(t)$. The *covariance-network* $\mathcal{N}_\psi(t)$ is also a fully-connected neural network, which approximate the state's covariance matrix by means of $\hat{P}(t) \approx \psi(t) = \mathcal{N}_\psi(t)$ with the network output $\psi(t)$. The *state-network* \mathcal{N}_ζ is an ODE-Network and aims at approximating the ODE $f(\cdot)$ in (1) by using the state's estimated mean value according to $\dot{x} = \mathcal{N}_\zeta(x, u, w, t)$.

Finally, the *output-network* \mathcal{N}_η is also an ODE-network and approximates the system's measurement equation $g(\cdot)$ by calculating $y(t) = \mathcal{N}_\eta(x, u, v, t)$.

The idea is summarized as follows: We train the networks in a PINN-framework, which is constrained by the differential equations of an EKBF's mean value and covariance matrix that we introduced in (3) and (5). This approach allows including measurement noise and process noise. Additionally, we do not need every state to be directly accessible. Hence, we train the ODE-Learner by minimizing a loss function that comprises four terms according to

$$L = \frac{1}{N} \sum_{i=1}^N (\alpha_1 L_{1,i} + \alpha_2 L_{2,i} + \alpha_3 L_{3,i} + \alpha_4 L_{4,i}) , \quad (11)$$

with weights $\alpha_j > 0$, $j \in \{1, 2, 3, 4\}$ to adjust the influence of the single components. The first term $L_{1,i}$ is a maximum-likelihood function that keeps our identified system's output close to the measurements $\bar{y}(t_i)$ according to

$$L_{1,i} = - \sum_{j=1}^q \log (\mathcal{L}_j (\xi(t_i), \psi(t_i); \bar{y}_j(t_i))) , \quad (12)$$

with the scalar output values $\bar{y}_j(t_i)$, $j = 1, \dots, q$ of the measurement vector $\bar{\mathbf{y}}(t_i)$. Since the noise is assumed to be Gaussian, \mathcal{L}_j describes the normal probability distribution function by means of

$$\mathcal{L}_j(\cdot) = \frac{1}{\sqrt{2\pi\sigma_j^2(t_i)}} \exp\left\{-\frac{(\bar{y}_j(t_i) - \mu_j(t_i))^2}{2\sigma_j^2(t_i)}\right\} \quad (13)$$

The mean value $\mu_j(t_i)$ and variance $\sigma_j^2(t_i)$ are the first two stochastic moments of the estimated state, when propagated through the output network by means of

$$\begin{aligned} \mu_j(t_i) &= \text{mean}(\eta_j(t_i)) \quad \text{and} \\ \sigma_j^2(t_i) &= \text{var}(\eta_j(t_i)) \end{aligned} \quad (14)$$

Here, $\eta_j(t_i)$ is acquired by propagating the estimated mean value $\boldsymbol{\xi}$ and the estimated covariance matrix $\boldsymbol{\psi}$ through the output-network $\boldsymbol{\eta}(t_i) = \mathcal{N}_\eta(\boldsymbol{\xi}, \boldsymbol{\psi}; \mathbf{u}, \mathbf{v}, t_i)$. Then, the output's mean value and variance can be both calculated in closed-form if the function $\mathcal{N}_\eta(\cdot)$ is (piece-wise) linear, trigonometric, or polynomial [13]. If the mean and variance cannot be calculated in closed-form, an approximate calculation by sampling or a numerical integration is required. Hence, it is advantageous to constrain the output-network to the mentioned cases.

The second term $L_{2,i}$ is defined by means of

$$L_{2,i} = \|\boldsymbol{\xi}(t_0) - \mathbf{x}_0\|_2 + \left\| \dot{\boldsymbol{\xi}}_{\text{AD}}(t_i) - \boldsymbol{\Xi}(\boldsymbol{\xi}, t_i) \right\|_2 \quad (15)$$

with

$$\begin{aligned} \boldsymbol{\Xi}(\boldsymbol{\xi}, t_i) &= \mathcal{N}_\zeta(\boldsymbol{\xi}(t_i), \mathbf{u}, \mathbf{0}, t_i) \\ &+ \mathbf{K}(t_i) \cdot (\bar{\mathbf{y}}(t_i) - \mathcal{N}_\eta(\boldsymbol{\xi}(t_i), \mathbf{u}, \mathbf{0}, t_i)) \quad . \end{aligned} \quad (16)$$

and the Kalman gain $\mathbf{K}(t_i) = \boldsymbol{\psi}(t_i) \cdot \hat{\mathbf{C}}^\text{T}(t_i) \cdot \hat{\mathbf{R}}^{-1}(t_i)$. $\dot{\boldsymbol{\xi}}_{\text{AD}}$ denotes the temporal derivative of $\boldsymbol{\xi}(t)$, acquired by automatic differentiation. Note that (16) describes the mean's temporal evolution in an EKBF, which we introduced in (3). In its essence, $L_{2,i}$ ensures that the state-network \mathcal{N}_ζ approximates the derivative with respect to time of the mean-network \mathcal{N}_ξ , while reducing the influence of measurement or process noise.

The third term $L_{3,i}$ is defined by means of

$$L_{3,i} = \left\| \boldsymbol{\psi}(t_0) - \hat{\mathbf{P}}_0 \right\|_{\text{F}} + \left\| \dot{\boldsymbol{\psi}}_{\text{AD}}(t_i) - \boldsymbol{\Psi}(\boldsymbol{\psi}, t_i) \right\|_{\text{F}} \quad (17)$$

and

$$\begin{aligned} \boldsymbol{\Psi}(\boldsymbol{\psi}, t_i) &= \hat{\mathbf{A}}(t_i)\boldsymbol{\psi}(t_i) + \boldsymbol{\psi}(t_i)\hat{\mathbf{A}}(t_i)^\text{T} \\ &- \mathbf{K}(t_i)\hat{\mathbf{C}}(t_i)\boldsymbol{\psi}(t_i) + \hat{\mathbf{Q}}(t_i) \quad . \end{aligned} \quad (18)$$

The matrices in (18) are acquired by linearization in a similar way to (6) by replacing $\mathbf{f}(\cdot)$ with $\mathcal{N}_\zeta(\cdot)$, $\mathbf{g}(\cdot)$ with $\mathcal{N}_\eta(\cdot)$ and $\hat{\mathbf{x}}$ with $\boldsymbol{\xi}$. $\dot{\boldsymbol{\psi}}_{\text{AD}}$ is the temporal derivative of $\boldsymbol{\psi}(t)$, acquired by automatic differentiation. Hence, the third loss enforces the temporal evolution of the estimated covariance matrix $\boldsymbol{\psi}(t)$ to be consistent with the estimated state-space models \mathcal{N}_ζ and \mathcal{N}_η .

Summarized, $L_{1,i}$ keeps the estimated system's output close to the measurement data, while $L_{2,i}$ and $L_{3,i}$ force

the estimated system to be compatible to the EKBF's mean and covariance matrix. The fourth loss $L_{4,i}$ is formed from regularization terms, which are described in the following. We want to enforce the network to only use the operators, which are deemed necessary. At the same time, we do not want to punish large coefficient values, since they might be necessary for a successful identification. Thus, the first regularization term is given by

$$R_0(w) = \frac{a_1}{1 + \exp\{-a_2 \cdot |w| + a_3\}} + a_4 \cdot |w| \quad (19)$$

with constants a_1, \dots, a_4 and a scalar weight value w . The regularization performs a very low punishment, if the weight value is close to zero and increases heavily afterwards. For larger values, the increase in regularization is marginal but present, which avoids vanishing gradients.

As we already mentioned in Sec. IV-A, it is necessary to punish an approach of the ODE-Network's denominator to the pole. Thus, we define another regularization term

$$R_1(\mathbf{o}, \mathbf{w}) = \max(0, \delta - \mathbf{o}^\text{T} \cdot \mathbf{w}^{(4)}) \quad , \quad (20)$$

which is similar to the method used in [25]. The loss $L_{4,i}$ from (11) is created from

$$L_{4,i} = \alpha_{4,1}R_0(w) + \alpha_{4,2}R_1(w) \quad (21)$$

with additional weights $\alpha_{4,1}, \alpha_{4,2} > 0$.

V. VALIDATION

We validate our method on three benchmarks. First, we focus on pure system identification by training an ODE-Learner and solve the resulting ODE numerically in order to compare the predicted outcome to the ground truth. Afterwards, we embed our ODE-Learner in a model-based RL-framework in order to reach a target state by using as little data as possible. As it is mentioned in Sec. IV-A, the ODE-Network can be substituted with a standard MLP. However, this does neither allow the inclusion of prior knowledge, nor can MLPs extrapolate very well. Nevertheless, in the following experiments, we compare the results of using an ODE-Network with a standard MLP. Additionally, we use state of the art algorithms that allow identifying a continuous-time ODE to compare it to our ODE-Learner framework. To be more precise, we choose SINDy, which is well-established in the system identification community, Mathwork's neural state-space, which is a Matlab-implementation of a neural ODE and, finally, the INN, as a counter-part to the neural ODE that does not rely on differentiation. We already introduced all three algorithms in Sec. II. To obtain an implementation, we use the publicly available GitHub-repositories for SINDy [7] and the INN [17], respectively. The neural state-space model is available in Matlab's system identification toolbox at version R2022b or later. We want to emphasize that we have chosen the hyperparameters of the methods to the best of our knowledge and belief. Furthermore, it is important to note that the results may vary significantly with different hyperparameters. In the following, we give an overview of our experiments.

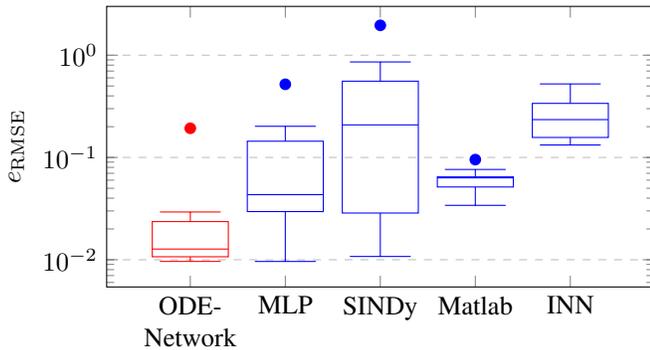


Fig. 3: Root mean squared error after identifying the Duffing oscillator with ten different parameter scenarios. Dots indicate outliers.

A. Duffing Oscillator

The Duffing oscillator is a 2nd-order benchmark system, which is very popular in the system identification community [10], [32]. It contains nonlinear operators and has a chaotic behavior, which means it is crucial to obtain a model that is able to generalize well from the provided data. Its state-space equation is given by means of

$$\begin{aligned} \dot{x}_1(t) &= x_2(t) \\ \dot{x}_2(t) &= b_1 x_2(t) + b_2 x_1(t) + b_3 x_1(t)^3 - b_4 \cos(\omega_0 t) \\ y(t) &= x_1(t) + v(t), \end{aligned} \quad (22)$$

with parameters $b_1, \dots, b_4 \in \mathbb{R}$, which define the system behavior, and the frequency of the input signal $\omega_0 \in \mathbb{R}$. We assume that we do not know the state-space equation in the identification process, apart from the operators, and only use (22) to synthesize data. Note, however, that we exacerbate the usual problem by only making $x_1(t)$ measurable after adding some zero-mean Gaussian-distributed noise $v(t)$.

To prove the effectiveness of the ODE-Learner, we build ten systems by sampling $b_1, \dots, b_4, \omega_0$ and the initial states $x_1(0), x_2(0)$ from a uniform distribution $\mathcal{U}(0, 1)$. Afterwards, we simulate measurement data by using Euler's method to solve (22) numerically over a period of $t_{\text{end}} = 48\text{s}$ with a sampling time of $\Delta t = 0.02\text{s}$ and add zero-mean Gaussian noise according to $v(t) \sim \mathcal{N}(0, 0.01^2)$. We use the first 40s of measurement data as a training set and the last 8s to validate our model. We identify a 2nd-order ODE, which corresponds to situation (iii) according to Section I, i.e., the

TABLE I: Root mean squared errors of the system identification tasks for the Duffing oscillator and the Cascaded Tank.

Method	Duffing oscillator	Cascaded Tank
ODE-Learner	0.03 ± 0.06	0.61
MLP	0.11 ± 0.16	1.23
SINDy	0.43 ± 0.61	1.53
Matlab	0.06 ± 0.02	1.26
INN	0.27 ± 0.13	$0.81 (0.41)^1$

¹The e_{RMSE} value of 0.41 was reported in [17].

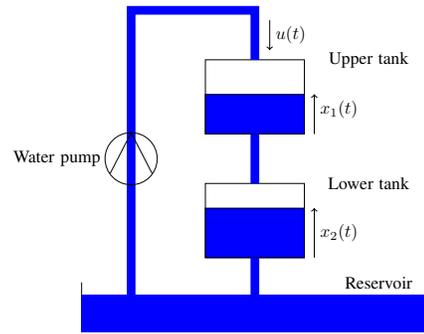


Fig. 4: Sketch of the cascaded tank system.

entire ODE needs to be identified. The Operator-neuron allows a cubic calculation. Afterwards, we use Euler's method to create a simulated trajectory of the states and feed them through the output equation to obtain the estimated system output $\hat{y}(t_i)$. The quality of the identification is then measured by calculating the root mean squared error e_{RMSE} between the identification $\hat{y}(t_i)$ and the ground truth $y(t_i)$.

The results of all identification runs are shown in Fig. 3 as a boxplot. The ODE-Learner using an ODE-Network enables a more precise identification than the other methods. Substituting the ODE-Network with an MLP leads to a slightly worse fit. In one out of the ten identification runs, we observe an outlier, which is caused by a gradient being stuck in a local minimum during the training. This could be corrected by restarting the training with new weights, however we kept the results in Fig. 3 unchanged. Additionally, we want to emphasize that the Matlab method requires all states to be accessible. Thus, we simplified the problem for Matlab and used both states. Additionally, Table I shows the mean value and standard deviation of the obtained identification errors.

B. Cascaded Tanks

In this paragraph we test the effectiveness of our method on a real-world system identification benchmark [28]. It is composed of two vertically arranged water tanks and a water reservoir, which is placed below. Fig. 4 features a sketch of the system.

A water pump transports the water from the reservoir into the upper tank. Each tank features a small opening at its bottom, allowing water to flow from the upper tank to the lower one and, finally, from the lower tank into the reservoir. However, both tanks do not have a lid. If the water pump's power is too high, both tanks can overflow. This behavior is nonlinear and stochastic, since water from the upper tank can flow partly into the lower tank or into the reservoir. The benchmark authors provide an ODE, which describes the state-space of the water flow by using Bernoulli's equation according to

$$\begin{aligned} \dot{x}_1(t) &= -k_1 \sqrt{x_1(t)} + k_4 u(t) + w_1(t) \\ \dot{x}_2(t) &= k_2 \sqrt{x_1(t)} - k_3 \sqrt{x_2(t)} + w_2(t) \\ y(t) &= x_2(t) + v(t), \end{aligned} \quad (23)$$

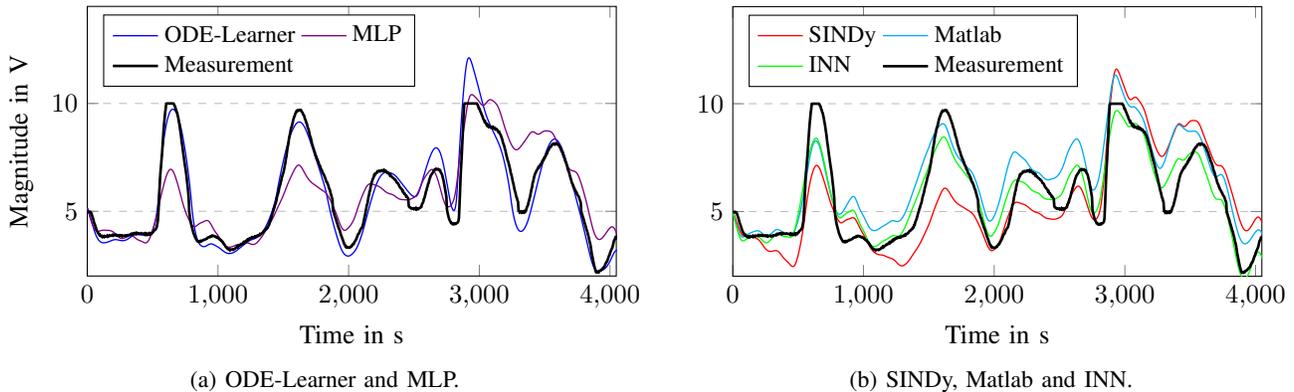


Fig. 5: Plots of the simulated height for the cascaded tank problem. (a) shows the result of our identified ODE, obtained by an ODE-Network and an MLP, compared to the original test data, while (b) shows state of the art system identification methods.

with $x_1(t)$ and $x_2(t)$ denoting the water level in the upper and lower tank, respectively. $u(t)$ denotes the system input and k_1, \dots, k_4 are unknown system parameters. The benchmark authors use a D/A-converter as system input, which controls the pump’s power by setting a voltage. The system’s output is the water level of the lower tank, which is measured by a capacitive water level sensor. The measurement values are, again, voltage values, this time from an A/D-converter. Neither sensor nor water pump dynamics are provided, which are, together with the overflow behavior and the initial state values \mathbf{x}_0 , subject to identification. Hence, this use case corresponds to situation (ii) according to Section I, i.e., some segments as well as all parameters of the ODE need to be identified. The benchmark authors provide a training and a test data set with 1,024 instances, respectively, which are each sampled at $\Delta t = 4$ s. We train an ODE-Learner by providing a pre-conditioned ODE-Network according to the dynamics, given in (23). We extend the network by nine operator-neurons to model the unknown dynamics. The output network used only performs a linear transformation, since we assume the capacitive water level sensor to exhibit linear behavior.

The e_{RMSE} of the identified systems are shown in Table I. Note that we can perform this validation only once, since it is a real-world example and we do not have multiple data sets available. Fig. 5 shows the plots and error values of the simulated outputs and the provided validation data for each investigated method. It is visible that we identified an ODE that shows the most similar behavior to the original one. The INN is very close to ours. We hypothesize that this is caused by the low sampling rate of only 4 s, which is too coarse for differentiation-based methods, but better suits methods that rely on integration. The authors of the INN-method even reached a root-mean-squared error of 0.41 with another hyperparameter setting. Replacing the ODE-Network by an MLP leads to results being similar to SINDy and Matlab.

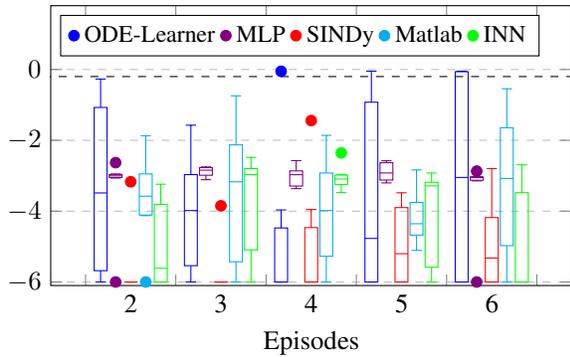
C. Inverted Pendulum on a Cart

The inverted pendulum on a cart is a frequently used benchmark for controllers. A pendulum is mounted on a

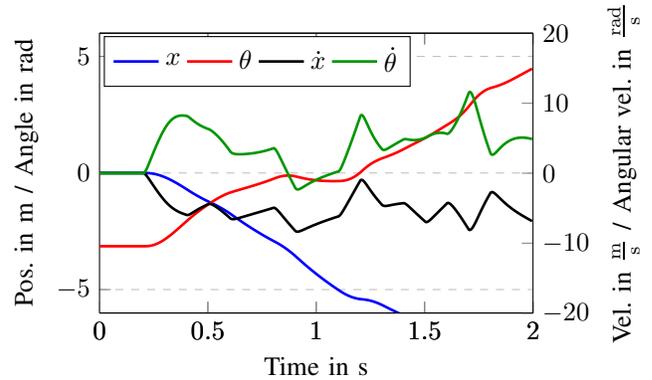
movable cart, which has one degree of freedom. The control target is to swing up the pendulum from its lower equilibrium point and keep it stable in an upright position. The system has a rank of $n = 4$ and a state vector $\mathbf{x} = [x \ \dot{x} \ \theta \ \dot{\theta}]^T$ with the cart position x , the cart velocity \dot{x} , the pendulum angle θ , and the angular velocity $\dot{\theta}$. The use case corresponds to situation (iii) according to Section I. The pendulum is in an upright position, when $\theta = 0$.

In the literature, the control job is often split up into the swing-up and the stabilization task [19]. Thus, we use an MPC-algorithm to swing up the pendulum. When it reaches its target state, a Linear Quadratic Regulator (LQR) keeps it stable in an upright position. We define the reward function by means of $R = \frac{1}{d} \sum_{i=N-d}^N -|\theta(t_i)|$ with the pendulum angle $\theta(t_i)$ at time step t_i and a time window length $d \in \mathbb{N}$. We consider the pendulum stable, if $R > -0.2$. This allows a very small oscillation of less than $\pm 11.5^\circ$ around the target position to be counted as successful swing-up. The initial state of the system is $\mathbf{x}_0 = [0 \ 0 \ -\pi \ 0]^T$. We perform the following routine: First, we simulate training data by moving a cart with a random input signal, which is capped at $-25 \text{ N} < u < 25 \text{ N}$. Afterwards, we train a first ODE by using the randomly gathered data. We then use this identified ODE in an MPC-framework to perform an L-BFGS-B-optimization to calculate the optimal input values, which we apply to the inverse pendulum system. Each episode comprises a data sample of $t_{\text{end}} = 2$ s with a sampling rate of $\Delta t = 4$ ms. The MPC solves an optimization problem of minimizing $J_{\text{MPC}} = -R(\mathbf{x})$ over a prediction horizon of $n_{\text{ph}} = 1$ s. We switch from the MPC to an LQR, as soon as the angle is close to the target position for a short period. To be precise, $|\theta| < \pi/6$ during 95 % of the last 0.2 s. When this switching criterion is reached, we use the identified ODE and calculate an LQR.

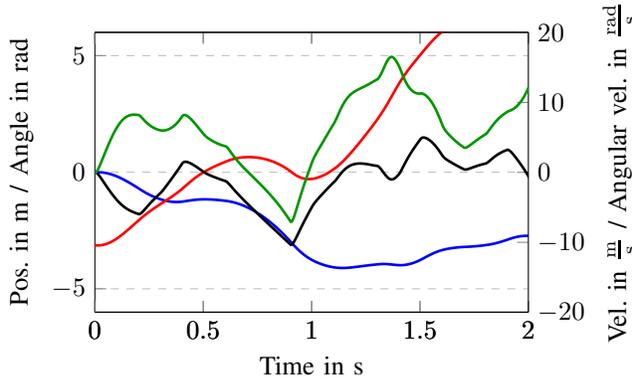
Fig. 6a shows the obtained reward for different episodes after six identification runs with different initial input values. In three out of six cases, we could swing up the pendulum and stabilize it after six episodes. In the remaining three cases, we did not observe a swing-up in the first six episodes. With more than six episodes, we obtained a large amount of training data,



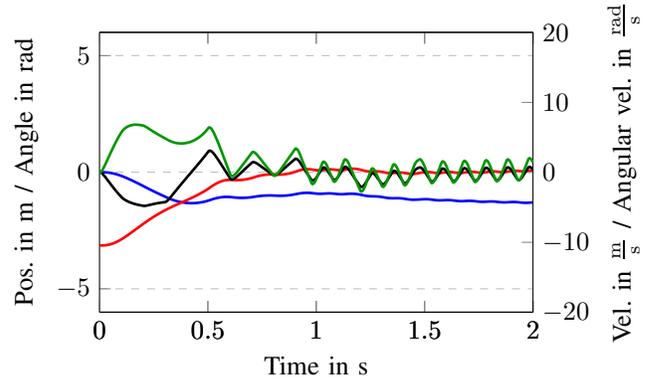
(a) Achieved rewards of the inverted pendulum. The dashed black line indicates a successful swing-up.



(b) Episode 1: Random actions



(c) Episode 2: Pure MPC, because the switching criterion has not been fulfilled.



(d) Episode 3: MPC until $t = 0.83$ s. Afterwards an LQR controls the input

Fig. 6: Achieved results for the inverse pendulum on a cart. The first plot shows a box plot of obtained reward after different episodes for the investigated methods. The next three plots show the trajectory of the controlled cart for one showcase, where we reached a swing-up at the third episode.

which becomes increasingly difficult to handle with the ODE-Learner, which is why we refrained from performing further episodes and considered these attempts as failed. We did not observe a swing-up with the other methods. Note that we capped the negative reward at $R_{\min} = -6$. Fig. 6b to 6d show the trajectories of a successful swing-up after three episodes, which we observed in another test run as a showcase.

VI. DISCUSSION

In this section, we want to list the strengths and limitations of our method. As we showed in Sec. V, our method reliably allows identifying an ODE, even when noisy data is provided. Additionally, it is possible to include human knowledge, which is not the case for methods like a neural ODE, the INN or an MLP. The prediction error remains reasonably small and our method is also applicable in a model-based RL framework. Furthermore, it is possible to extract a differential equation with a better root mean squared error than established methods.

However, there are some drawbacks. Firstly, the training time is rather high. We used an Nvidia A100 GPU with training times between 45 min for the duffing oscillator and

36 h for the complete run of the inverted pendulum task. This is due to the need to linearize the state-network in order to obtain the state matrix A in (18), which requires multiple differentiation steps. If we do not manage a successful run in the first few episodes of a model-based RL-problem, the accumulated data grows to such an extent that the training time becomes unreasonably high, which is what happened in the inverted pendulum task. All other methods require significantly less training time. Additionally, our method has a large number of hyperparameters. Especially the architecture of all four networks has a major impact on the result. The only way to find the correct network architecture is by trial-and-error or by utilizing a neural architecture search [22]. As is visible in Fig. 3, the ODE-Learner sometimes shows an outlier. We observe this behavior in approximately one in ten cases. We suspect that the reason for this lies in the backpropagation algorithm, which can get stuck in a local minimum. If we restart the algorithm with new weighting matrices, the outlier disappears. Nevertheless, we are still performing better than the state of the art in terms of data efficiency, since there is no method available that allows swinging up a pendulum after less

than 6 s of measurement data. Another drawback is the non-interpretability of the identified ODEs. Despite experimenting a lot with increasing the weight of the introduced regularization terms, we were not able to identify ODEs, which are compact enough to interpret the mathematical model and to visually check, whether the model is correct. Finally, despite identifying a continuous-time differential equation, the control is performed by an MPC-algorithm, which discretizes the identified ODE afterwards. Using a continuous-time control law might leverage the advantages of continuous-time identification further.

VII. CONCLUSION AND OUTLOOK

We presented a new method to identify ODEs from noisy measurement data that allows human knowledge to be incorporated and that does not demand every state to be directly accessible. We showed that the identification works very precisely and proved that it is possible to embed it into a model-based RL framework. For future work, we plan to optimize the control strategy, in order to use the trained ODEs more efficiently and reduce the data usage even further. Additionally, we are working on training a nonlinear control function in parallel to the identified ODE.

REFERENCES

- [1] Luis Antonio Aguirre. A bird's eye view of nonlinear system identification. *arXiv preprint arXiv:1907.06803*, 2019.
- [2] Devin Akman, Olcay Akman, and Elsa Schaefer. Parameter estimation in ordinary differential equations modeling via particle swarm optimization. *Journal of Applied Mathematics*, 2018:1–9, 2018.
- [3] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.
- [4] J. M. Caicedo and Shamim Pakzad, editors. *Proceedings of the 35th IMAC, a Conference and Exposition on Structural Dynamics, 2017*. Conference Proceedings of the Society for Experimental Mechanics Series. Springer, Cham, Switzerland, 2017.
- [5] Ci Chen, Hamidreza Modares, Kan Xie, Frank L. Lewis, Yan Wan, and Shengli Xie. Reinforcement learning-based adaptive optimal exponential tracking control of linear systems with unknown dynamics. *IEEE Transactions on Automatic Control*, 64(11):4423–4438, 2019.
- [6] Ricky T.Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.
- [7] Brian de Silva, Kathleen Champion, Markus Quade, Jean-Christophe Loiseau, J. Kutz, and Steven Brunton. Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data. *Journal of Open Source Software*, 5(49):2104, 2020.
- [8] Vrushabh S. Donge, Bosen Lian, Frank L. Lewis, and Ali Davoudi. Data-efficient reinforcement learning for complex nonlinear systems. *IEEE transactions on cybernetics*, pages 1–12, 2023.
- [9] U. Fasel, J. N. Kutz, B. W. Brunton, and S. L. Brunton. Ensemble-SINDy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 478(2260), April 2022.
- [10] S. Khatiry Goharoodi, K. Dekemele, L. Dupre, M. Loccufier, and G. Crevecoeur. Sparse Identification of Nonlinear Duffing Oscillator From Measurement Data. *IFAC-PapersOnLine*, 51(33):162–167, 2018. 5th IFAC Conference on Analysis and Control of Chaotic Systems CHAOS 2018.
- [11] Viktor Grimm, Alexander Heinlein, Axel Klawonn, Martin Lanser, and Janine Weber. Estimating the time-dependent contact rate of sir and seir models in mathematical epidemiology using physics-informed neural networks. *Electron. Trans. Numer. Anal.*, 56:1–27, 2022.
- [12] S. Hassouna, P. Coirault, and T. Poinot. Non-linear system identification using volterra series expansion. *IFAC Proceedings Volumes*, 33(15):947–952, 2000. 12th IFAC Symposium on System Identification (SYSID 2000), Santa Barbara, CA, USA, 21–23 June 2000.
- [13] Marco Huber. *Nonlinear Gaussian Filtering: Theory, Algorithms, and Applications*. KIT Scientific Publishing, Karlsruhe, March 2015.
- [14] R. E. Kalman and R. S. Bucy. New Results in Linear Filtering and Prediction Theory. *Journal of Basic Engineering*, 83(1):95–108, 1961.
- [15] Ying Ma, Haopeng Liu, Yunpeng Zhu, Fei Wang, and Zhong Luo. The narx model-based system identification on nonlinear, rotor-bearing systems. *Applied Sciences*, 7(9):911, 2017.
- [16] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3952–3963. Curran Associates, Inc., 2020.
- [17] Bojan Mavkov, Marco Forgiione, and Dario Piga. Integrated neural networks for nonlinear continuous-time system identification. *IEEE Control Systems Letters*, 4(4):851–856, 2020.
- [18] Eshan D. Mitra and William S. Hlavacek. Parameter estimation and uncertainty quantification for systems biology models. *Current Opinion in Systems Biology*, 18:9–18, 2019.
- [19] Shozo Mori, Hiroyoshi Nishihara, and Katsuhisa Furuta. Control of unstable mechanical system control of pendulum†. *International Journal of Control*, 23(5):673–692, 1976.
- [20] Tobias Nagel and Marco F. Huber. Kalman-bucy-informed neural network for system identification. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 1503–1508, 2022.
- [21] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations, 2017.
- [22] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34, 2021.
- [23] Hegazy Rezk, Seydali Ferahtia, Ali Djeroui, Aissa Chouder, Azeddine Houari, Mohamed Machmoum, and Mohammad Ali Abdelkareem. Optimal parameter estimation strategy of pem fuel cell using gradient-based optimizer. *Energy*, 239:122096, 2022.
- [24] Samuel Rudy, Alessandro Alla, Steven L. Brunton, and J. Nathan Kutz. Data-driven identification of parametric partial differential equations. *SIAM Journal on Applied Dynamical Systems*, 18(2):643–660, 2019.
- [25] Subham S. Sahoo, Christoph H. Lampert, and Georg Martius. Learning Equations for Extrapolation and Control. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.
- [26] Simo Särkkä. The Use of Gaussian Processes in System Identification. *Encyclopedia of Systems and Control*, 2019.
- [27] Johan Schoukens and Lennart Ljung. Nonlinear system identification: A user-oriented road map. *IEEE Control Systems*, 39(6):28–99, 2019.
- [28] Maarten Schoukens, Per Mattson, Torbjörn Wigren, and Jean-Philippe Noël. Cascaded Tanks Benchmark Combining Soft and Hard Nonlinearities, 2020.
- [29] Paul M.J. van den Hof, Peter S.C. Heuberger, and József Bokor. System identification with generalized orthonormal basis functions. *Automatica*, 31(12):1821–1834, 1995.
- [30] Mats Viberg. Subspace-based state-space system identification. *Circuits, Systems, and Signal Processing*, 21(1):23–37, 2002.
- [31] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking Model-Based Reinforcement Learning, 2019.
- [32] Torbjörn Wigren and Johan Schoukens. Three free data sets for development and benchmarking in nonlinear system identification. In *2013 European Control Conference (ECC)*, pages 2933–2938, 2013.
- [33] Mario Zanon and Sebastien Gros. Safe reinforcement learning using robust mpc. *IEEE Transactions on Automatic Control*, 66(8):3638–3652, 2021.
- [34] Michael Zhang, Samuel Kim, Peter Y. Lu, and Marin Soljačić. Deep Learning and Symbolic Regression for Discovering Parametric Equations, 2022.