IEEE Control Systems Letters paper presented at
2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Neural ODEs for Data-driven Automatic Self-Design of Finite-time Output Feedback Control for Unknown Nonlinear Dynamics

Simon Bachhuber, Ive Weygers, and Thomas Seel

***Abstract*—Many application fields, e.g., robotic surgery, autonomous piloting, and wearable robotics greatly benefit from advances in robotics and automation. A common task is to control an unknown nonlinear system such that its output tracks a desired reference signal for a finite duration of time. A learning control method that automatically and efficiently designs output feedback controllers for this task would greatly boost practicality over time-consuming and labour-intensive manual system identification and controller design methods. In this contribution we propose Automatic Neural Ordinary Differential Equation Control (ANODEC), a data-efficient automatic design of output feedback controllers for finite-time reference tracking in systems with unknown nonlinear dynamics. In a two-step approach, ANODEC first identifies a neural ODE model of the system dynamics from input-output data of the system dynamics and then exploits this data-driven model to learn a neural ODE feedback controller, while requiring no knowledge of the actual system state or its dimensionality. In-silico validation shows that ANODEC is able to —automatically— design competitive controllers that outperform two controller baselines, and achieves an on average $\approx 30\,\%$ / $17\,\%$ lower median RMSE. This is demonstrated in four different nonlinear systems using multiple, qualitatively different and even out-of-training-distribution reference signals.**

I. Introduction

Advances in robotics and automation continue to have a significant impact on a large range of application fields: Cars that autonomously perform certain maneuvers, assembly lines with robotic manipulators that perform repetitive tasks, and patients that regain the ability to perform functional motions through combinations of functional electrical stimulation and wearable robotics.

A common task in all of these application fields is to control the output of a system with initially unknown nonlinear dynamics such that it follows a desired reference signal for a finite duration of time. Finding a feedback controller that solves this task is typically time-consuming and labour-intensive, since it requires modeling and/or system identification as well as controller design and adaptation. This greatly contrasts practical needs for methods and algorithms that exploit small amounts of input-output data from unknown nonlinear dynamics and autonomously design feedback controllers that enable accurate tracking of agile finite-time references (cf. Fig. 1).

In the present contribution we provide an approach that solves this task and neither assumes knowledge of the system dynamics nor of the system state or its dimensionality. It automatically designs an output feedback controller that tracks *real-time* reference signals and does not require these references to be repetitive or known ahead of time (AOT).

S.B., I.W., and T.S. are part of the Department Artificial Intelligence in Biomedical Engineering, FAU Erlangen-Nürnberg, 91052 Erlangen, Germany. Corresponding author: S.B. (simon.bachhuber@fau.de)

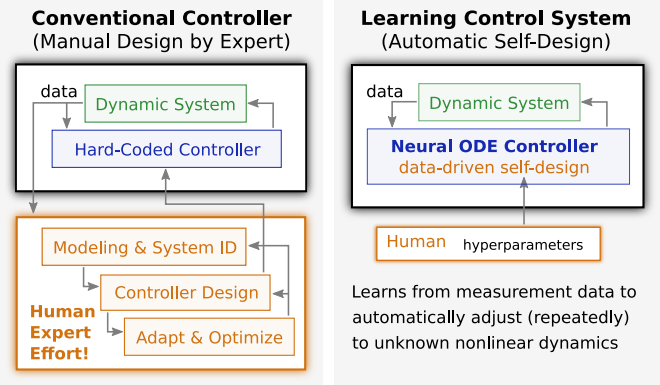

Fig. 1. Core concept of automatic self-design in learning control systems. Conventional controller design requires manual effort by human experts, which is expensive, time-consuming and does not scale. In contrast, the proposed ANODEC approach automatically designs competitive controllers from input-output data of unknown nonlinear system dynamics.

Similar problems have been addressed by several previous approaches, most notably in the field of reinforcement learning (RL) [1], [2]. These methods, however, typically require full state knowledge, which severly limits applicability [3]. Within the field of RL, this limitation can be overcome using Partially Observable Markov Decision Processes (POMDP). RL methods tailored for POMDPs have been developed [4] and include, e.g., a recurrent policy function [5], stacking multiple observations, or, as a patch, a recurrent filter prior to a MDP-RL algorithm[6]. Unfortunately, the applicability of these methods is limited by large data requirements [1], [7]. Moreover, while some RL methods have been used to effectively learn feedback control, they always assumed a single-objective trial-invariant state-dependent reward function which encodes information about a reference signal or target state that is usually known AOT [8].

More system-and-control-rooted approaches to similar tasks include optimal control (OC) and iterative learning control (ILC) methods and almost always require a known system model. OC approaches, including model-predictive control, typically optimize a state-dependent quadratic loss and thus assume full state knowledge [9]. Even recent data-driven methods still require knowledge of a nominal model of the dynamics and only learn the model mismatch from data [10]. ILC approaches, see e.g. [11], [12], may not assume full state knowledge, but their design requires a system model or experimental tuning, and they assume a reference signal that is known AOT and typically trial-invariant.

In the present work we propose a solution based on *neural ordinary differential equations (ODEs)*, in which the right-hand-side of an ODE is parameterized by a neural network.

This concept is enabled by numerical integrators being differentiable operations in the context of automatic differentiation and has recently gained traction [13]. Neural ODEs are not only used for modeling [13], [14] but also for representing controllers [15], [16]. In contrast to previous work on the latter, we assume to have no prior knowledge about the state or the dynamics of the system to be controlled.

Precisely, we propose Automatic Neural ODE Control (ANODEC) (cf. Fig. 1), a data-efficient automatic design of neural ODE feedback controllers for finite-time reference tracking in systems with unknown nonlinear dynamics. ANODEC consistently outperforms two common control approaches, and achieves an on average $\approx 30\,\%$ / $17\,\%$ lower median RMSE. This is demonstrated using multiple, qualitatively different and out-of-training-distribution reference signals in several double-pendulum dynamics and vehicle steering dynamics.

II. Problem Formulation

Assume there exists some *unknown* system dynamics Ψ that maps a time-varying, finite-time input signal $u(t) \in \mathbb{R}^p$ to a possibly noisy output vector $y(t) \in \mathbb{R}^q$, that is

$$y(t) = \Psi\left[u(t' < t)\right] \quad \forall t \in [0, T], \tag{1}$$

where $T \in \mathbb{R}$ is the finite trial duration. Here, Ψ can be thought of as any causal, time-invariant, potentially nonlinear, deterministic dynamical system, which includes, e.g. linear state-space models, nonlinear higher-order differential equations, and lifted-system dynamics. However, at least in the present work, the systems should be well-behaved in the sense that they exhibit none of the following very challenging phenomena: Finite escape times [17], ill-conditioned linearizations [18], or non-minimum phase characteristics [19].

We want to design a feedback controller that manipulates $u(t)$ to let $y(t)$ follow a given time-varying reference signal $r(t) \in \mathbb{R}^q$. Thus, we seek to find a controller dynamics Φ that maps $r(t)$ and $y(t)$ to the input vector $u(t)$, i.e.

$$u(t) = \Phi\left[r(t' < t), y(t' < t), t\right] \quad \forall t \in [0, T], \tag{2}$$

such that it minimizes the tracking error between the output and the reference signal, i.e.

$$\Phi^* = \arg\min_{\Phi} \int_0^T \left\| r(t) - y(t) \right\|_2 dt, \tag{3}$$

over the finite trial duration. Here, Φ can be used to express any causal, potentially nonlinear and time-variant, deterministic dynamical system, which includes e.g. transfer functions, dynamic nonlinearities, and recurrent neural networks.

Synthesizing a feedback controller that only measures the output of the system dynamics Ψ, and not the state, is a particularly challenging problem, especially when the reference signal is only provided in real time and not ahead of time. Note that this problem formulation assumes no knowledge of the inner or physical state of the system dynamics Ψ or of its dimensionality. We do, however, assume that the system captured by Ψ is repeatedly reset to some trial-invariant initial state before every trial.

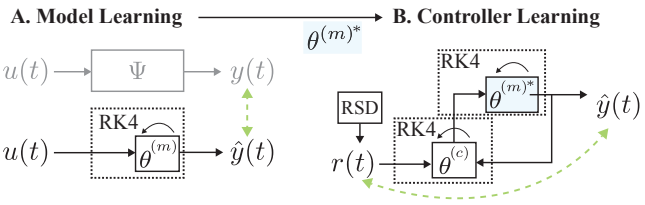

Fig. 2. Internal two-step approach underlying Automatic Neural ODE Control (ANODEC): (A) First, a neural ODE that models the unknown nonlinear dynamics of system Ψ is learned from input-output data (green arrow left side). (B) Then, a performant neural ODE controller is learned from thousands of forward simulations of the closed-loop system of the frozen model (blue highlighting) and the neural ODE controller. The optimization objective of the neural ODE controller is accurate reference tracking in the closed loop system (green arrow right side) subject to randomly drawn reference signals $r(t)$ from the Reference Signal Distribution (RSD).

III. Automatic Neural ODE Control (ANODEC)

Here, we present the mathematical formulation of the proposed method ANODEC. Generally speaking, ANODEC designs controllers in a two-step approach of model learning and controller learning (cf. Fig. 2). However, the two steps are *internal* to the algorithm and don't require any intermediate human attention, which contrasts the typical manual approach of system identification and controller design. The content of this section is split up according to these two steps in Sections III-A and III-B, respectively.

A. Model Learning

The requirement of learning a feedback controller using the model, necessarily forces us to model causally. One-shot models that map a sequence to a sequence are not possible. While ordinary differential equations (ODEs) have a long history for modeling of physical (causal) phenomena, neural networks are a first choice for function approximation from data. We thus choose the combination of both, i.e. neural ODEs, for automatic data-driven model learning. Additionally, neural ODEs are differentiable, which will facilitate fast controller optimization in Section III-B.

We approximate Ψ in (1) by a neural ODE that is learned from experimental input-output data (u_i, y_i) (more on data requirements in Section IV-A). The state of the dynamical system Ψ is not observed. The neural ODE that approximates Ψ is given by

$$\begin{aligned} \frac{d\xi^{(m)}(t)}{dt} &= f_\theta^{(m)}\left(\xi^{(m)}(t), u(t)\right), \\ \hat{y}(t) &= g_\theta^{(m)}\left(\xi^{(m)}(t)\right), \end{aligned} \tag{4}$$

where $f_\theta^{(m)}$ and $g_\theta^{(m)}$ are feedforward neural networks, $u(t) \in \mathbb{R}^p$ the network input, $\hat{y}(t) \in \mathbb{R}^q$ the network output, and $\xi^{(m)}$ is the latent state vector of arbitrary dimension in which the neural ODE evolves. We denote the vector of all parameters of $f^{(m)}, g^{(m)}$ by $\theta^{(m)}$ and use supervised learning and training data of input-output pairs (u_i, y_i) to estimate and then optimize for these parameters

$$\theta^{(m)^*} = \arg\min_{\theta^{(m)}} \mathop{\mathbb{E}}_{(u,y)} \left[\int_0^T \left\| y(t) - \hat{y}(t) \right\|_2 dt \right]. \tag{5}$$

B. Controller Learning

As a second step, we design a controller using the trained model (cf. Section III-A) with frozen parameters $\theta^{(m)^*}$. In contrast to previous approaches [15], [16], we assume to have no measurements of the state of the dynamical system, i.e. we consider output feedback control. A performant controller, in the sense of Section II, is learned from a very large number of forward simulations of the trained model with different reference signals.

We represent the controller Φ as a neural ODE, given by

$$\begin{aligned} \frac{d\xi^{(c)}(t)}{dt} &= f_\theta^{(c)}\left(\xi^{(c)}(t), r(t), y(t), t\right), \\ u(t) &= g_\theta^{(c)}\left(\xi^{(c)}(t)\right), \end{aligned} \tag{6}$$

where $f_\theta^{(c)}$ and $g_\theta^{(c)}$ are feedforward neural networks and $\xi^{(c)}$ is the latent state vector of arbitrary dimension in which the neural ODE evolves. We then close the loop between the neural ODEs that approximate the system dynamics Ψ and the controller Φ. The resulting closed-loop ODE takes the reference $r(t)$ as an input and outputs $\hat{y}(t)$. Ideally, this mapping should be identity, and thus we optimize the controller parameters $\theta^{(c)}$ via

$$\begin{aligned} \theta^{(c)^*} = \arg\min_{\theta^{(c)}} \Bigg(& \lambda^{(c)} \left\| \theta^{(c)} \right\|_2 + \\ & \mathop{\mathbb{E}}_{r(t)\sim\mathcal{R}} \left[\int_0^T \left\| r(t) - \hat{y}(t) \right\|_2 dt \right] \Bigg) \end{aligned} \tag{7}$$

where $\lambda^{(c)}$ is a small regularization weight, and $\mathcal{R}$ is the reference signal distribution (RSD) (cf. Section IV-D). Note that ANODEC can easily be trained for a certain noise type and level by disturbing the model output i.e. $\hat{y} \to \hat{y} + \epsilon$ in (7) where ϵ can be drawn from any distribution. Finally, note that deliberate choices in the neural network architecture of the model (cf. Section IV-B) allow to omit additional cost terms in (7) that are typically used to prevent model inversion. Such terms are not required here, as a similar effect is achieved by limiting the rate of change of the model output (cf. Section IV-B).

IV. Algorithm

In this section we layout the required details to implement ANODEC. This includes the specific choices of feedforward neural networks f_θ, g_θ that parameterize (4) and (6) and how integration, expectation, and minimization operators of (5) and 7 are resolved.

A. Data Requirements

To apply ANODEC, a (training) dataset $\mathcal{D}$ that comprises N pairs of input-output data (each corresponding to one trial) must first be gathered from the system dynamics Ψ. The model learning (cf. Section III-A) then uses the dataset

$$\mathcal{D} = \{(u_i(t), y_i(t)) | i \in 1 \dots N\}. \tag{8}$$

Input trajectories $u(t)$ for gathering the data thus probing the system are drawn from a cosine frequency generator and a smooth Gaussian process with an overall dataset that is split with a ratio of $3:1$. These functions are provided as pseudo-code in Algorithm 1. Between each trial the system resets to a trial-invariant initial state. Without loss of generality, we here make the choice of trials that last $T = 10\,\mathrm{s}$ and are recorded at a sampling rate of $100\,\mathrm{Hz}$. This leads to e.g. an array representation of $u_i \in \mathbb{R}^{1000\times p}\ \forall i$.

Finally, for the remainder of this document we assume that in addition to the training dataset there exists a separate validation dataset that comprises three trials.

Algorithm 1 Functions for drawing $u(t)$ to probe the system

Precondition: ts is array of timesteps, seed is integer, scale is float
function DRAWGP(ts, seed, scale$= 0.25$)
  kernel $\leftarrow 0.15$ `squaredExpKernel`(length$= 0.75$)
  us $\leftarrow$ `gaussianProcess`(kernel, ts, seed)
  us $\leftarrow \left(\frac{\mathtt{us}-\mathtt{mean(us)}}{\mathtt{std(us)}}\right) \times$ scale
  return us
function DRAWCOS(ts, seed)
  $\omega \leftarrow 2\pi\times$ seed
  us $\leftarrow \mathtt{cos}(\omega \times \mathrm{ts})\frac{\sqrt{\omega}}{2}$
  return us

B. Neural Network Architecture

To parameterize the right-hand-side in (4), we use a fully-connected neural network $f_\theta^{(m)} : \mathbb{R}^{N^{(m)}\times p} \to \mathbb{R}^{N^{(m)}}$

$$\begin{aligned} \left(\xi^{(m)}(t)^\mathsf{T}, u(t)^\mathsf{T}\right)^\mathsf{T} & \to \mathtt{Linear} \to \mathtt{Relu} \\ & \to \mathtt{Linear} \to \mathtt{Tanh} \\ & \to \frac{d\xi^{(m)}(t)}{dt} \quad \forall t \in [0, T], \end{aligned} \tag{9}$$

with a width of $N^{(m)}$, a latent state vector $\xi^{(m)} \in \mathbb{R}^{N^{(m)}}$ and a layer size of $N^{(m)}$. Note the usage of `Tanh` in (9) to limit the rate of change of the latent state. Bounding the absolute rate of change from above in combination with a finite trial duration ensures that the model state and output will also stay within bounds. Similarly, in (4) $g_\theta^{(m)} : \mathbb{R}^{N^{(m)}} \to \mathbb{R}^q$ is parameterized as a single `Linear` layer, i.e. as a linear affine transformation. Note that the `Tanh` is used for system dynamics $f_\theta^{(m)}$ and not for system output $g_\theta^{(m)}$, and since the latent state $\xi^{(m)}$ can be arbitrarily scaled by $g_\theta^{(m)}$, limiting the rate of change of $\xi^{(m)}$ does not impose restrictions on the model output range.

To parameterize the controller, i.e. the right-hand-side in (6), we let $f_\theta^{(c)} : \mathbb{R}^{N^{(c)}\times q\times q} \to \mathbb{R}^{N^{(c)}}$ be a `Linear` layer and $g_\theta^{(c)} : \mathbb{R}^{N^{(c)}} \to \mathbb{R}^p$ with $\xi^{(c)}(t) \to \mathtt{Linear} \to \mathtt{Tanh} \to u(t) \quad \forall t \in [0, T]$. Together the two latent state dimensionalities of model $N^{(m)}$ and controller $N^{(c)}$ are the hyperparameters of ANODEC.

C. Time Integration

We use Runge-Kutta of fourth order (RK4) to obtain a numerical solution of the integration in (5) and (7) together with the initial condition $\xi^{(m)}(t=0) = 0$ and $\xi^{(m)}(t=0) = \xi^{(c)}(t=0) = 0$, respectively.

D. Expectation and Minimization Operators

For model training the training dataset $\mathcal{D}$ is split into four minibatches. In (5) the expectation operator $\mathop{\mathbb{E}}_{(u,y)}$ is estimated using a single minibatch. Gradients are then computed and an optimizer step is performed. After an epoch is finished (equaling four optimizer steps) the dataset is shuffled. For model training the minimization operation $\arg\min_{\theta^{(m)}}$ is performed using the Adam optimizer, a cosine-decaying learning rate schedule and norm-based and absolute value gradient clipping. The model is trained for (at most) 700 epochs and the validation dataset is used for early stopping.

The expectation operator $\mathop{\mathbb{E}}_{r(t)\sim\mathcal{R}}$ in (7) is estimated using a minibatch of six references where at every epoch thirty reference signals (containing five minibatches) are drawn randomly from the training RSD $\mathcal{R} = \texttt{steps}(0, 3)$ defined by

$$\texttt{steps}(a, b) = \left\{ r(t) = sr_0 \; \forall t \in [0, T] \;\middle|\; \begin{array}{l} r_0 \in \mathcal{U}([a, b]) \\ s \in \mathcal{U}(\{-1, 1\}) \end{array} \right\} \quad (10)$$

where $a, b \in \mathbb{R}$ and $\mathcal{U}(\cdot)$ denotes a uniform distribution over a set. For controller training the minimization operation $\arg\min_{\theta^{(c)}}$ is performed using the same optimization strategy as it is used for model training with the addition of a small regularization weight $\lambda^{(c)} = 0.1$ in (7). The controller is trained for 1200 epochs.

E. Software Implementation

A software implementation of ANODEC and of the entire research project, including the routines to simulate the example systems used for validation (cf. Section V-A, V-B), is available

https://github.com/SimiPixel/chain_control.

The Python-based software uses JAX [20] and MuJoCo [21]. The ANODEC pipeline requires less than two minutes of compute time on a i7-1165G7. The trained neural ODE controller can be applied in real time at well above 100 Hz.

V. In-silico Validation

In this section we validate ANODEC (cf. Section III, IV) on four simulated complex nonlinear dynamical systems: (A) three different spring-damper double pendulum systems (DPs) (cf. Section V-A) and (B) a vehicle with Ackermann steering (AST) (cf. Section V-B). To showcase generalization capabilities of ANODEC in Section V-E, we propose several unseen reference signal distributions that, in Section V-F, will be used to validate ANODEC's performance compared to a system-specific optimal baseline (cf. Section V-D).

A. Double Pendulum (DP) System

A first dynamical system Ψ (cf. Section II) that is used for validation is a DP attached to a cart that moves horizontally on a slider using a single motor input. A connected DP with two consecutive hinge joint segment pairs is attached to the cart. Both hinge joints are damped with a factor $\gamma \in \mathbb{R}$, and are being pulled into a straight position by springs with stiffness $k \in \mathbb{R}$. Gravity would lead to less challenging dynamics, so it is disabled. The single input of the system is the cart's motor input moving the cart left-to-right on the slider. The single output is the left-to-right Cartesian coordinate of the end effector of the double pendulum, i.e. the end position of the second segment.

We investigate three parametrizations of this system corresponding to DP1/DP2/DP3, $\gamma = 6.0 \times 10^{-2}/3.0 \times 10^{-2}/1.5 \times 10^{-2}$, and $k = 6.0/3.0/1.5$, respectively. This results in notably different pendulum swinging characteristics. All three systems are simulated using MuJoCo [21], the training dataset (8) contains $N = 12$ trials, and the neural network widths of model and controller are chosen as $N^{(m)} = 50$ and $N^{(c)} = 15$.

B. Ackermann Steering (AST) System

To showcase applicability across different application systems, we also validate ANODEC for the system dynamics of a vehicle with Ackermann Steering. The vehicle with width $w_r = 1.5\,\mathrm{m}$ and length $l_r = 4.0\,\mathrm{m}$ drives forward with a constant velocity of $v_r = 3\,\mathrm{m\,s^{-1}}$. The single input of the system is the steering wheel angle ϕ_r in radians, clipped to a range $[-30.0°, 30.0°]$. The single output of the system is the Cartesian y-position of the vehicle $y_r \in \mathbb{R}$. The dynamics are given by $\frac{d\theta_r(t)}{dt} = \frac{v_r}{\frac{l_r}{\tan(\phi_r(t)+\epsilon)} - \frac{w_r}{2}}$, and $\frac{d}{dt}\begin{bmatrix} x_r(t) \\ y_r(t) \end{bmatrix} = v_r \begin{bmatrix} \cos(\theta_r(t)) \\ \sin(\theta_r(t)) \end{bmatrix}$, where $\theta_r \in \mathbb{R}$ is the angle between the cart's forward / length axis and the global x-axis, and $x_r \in \mathbb{R}$ is the Cartesian x-position of the vehicle. Initially, $x_r(t=0) = y_r(t=0) = \theta_r(t=0) = 0$, i.e. the vehicle is aligned with the global x-axis.

For this system the training dataset (8) contains $N = 24$ trials, and the neural network widths of model and controller are chosen as $N^{(m)} = 25$ and $N^{(c)} = 25$.

C. Noisy System Output

In all simulations we consider noisy output measurements by adding Gaussian white noise with a standard deviation of $0.02\,\mathrm{m}$ to the system output $y(t)$ in (1). Note that in Fig. 4 we merely plot the noise free system output, yet the controller still observes the noisy system output.

D. Baseline Controllers

We compare the performance of ANODEC to two baseline controllers: One controller that assumes the ability to perfectly tune a PI or PD controller, one controller that assumes that perfect model knowledge is available at least locally in the form of a linearization of the plant around its initial state.

1) Optimal-Tuning Baseline Controller: As a first baseline controller we determine, for each of the four systems, an optimal PI or PD controller (whichever performs better) by performing two exhaustive grid searches for both gains on the true system dynamics. At every grid point we evaluate the PI (or PD) controller's performance for 15 reference signals drawn from the training RSD (cf. Section IV-D) that are fixed across grid points. Note that this yields the best possible PI/PD controller for the given task and that performing such an exhaustive parameter optimization would require enormous amounts of interaction time in real-world systems.

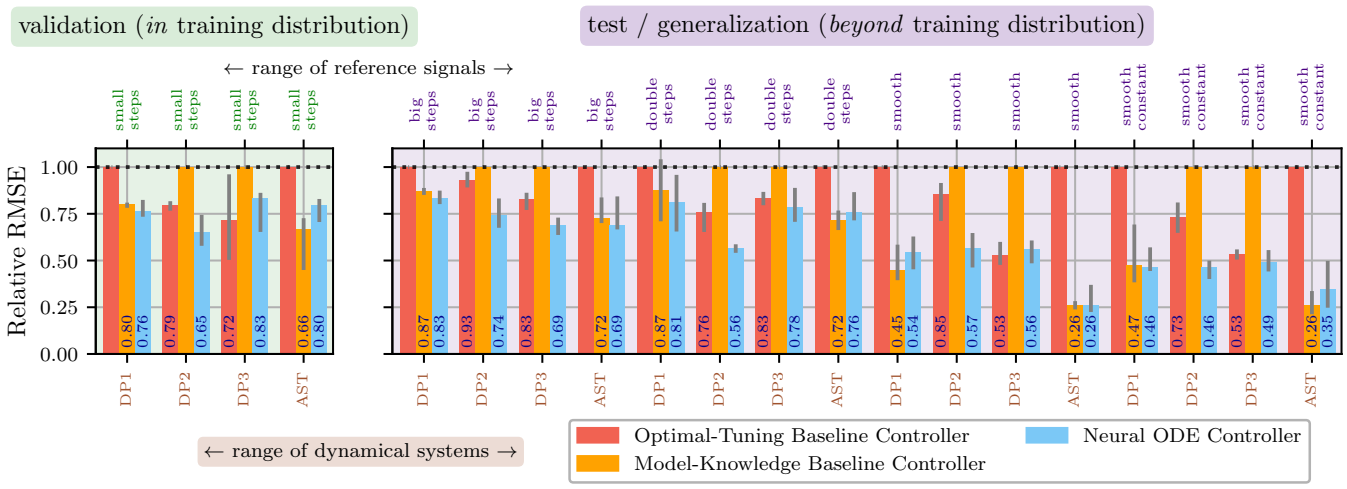

Fig. 3. Trained neural ODE controller's performance relative to the optimal-tuning controller baseline and model-knowledge controller baseline on all combinations of four systems and five reference signal distributions (RSDs). For each combination of system and RSD, the $25\%/50\%/75\%-$percentiles are obtained from 100 randomly drawn references from the RSD. The neural ODE controller achieves an on average $\approx 30\,\%$ / $17\,\%$ / $7\,\%$ / $40\,\%$ lower median RMSE across all validation and test combinations compared to the optimal-tuning controller baseline / model-knowledge controller baseline / the best out of the two baselines / the worst out of the two baselines.

2) Model-Knowledge Baseline Controller: As a second baseline controller we determine, for each of the four systems, a 5th-order transfer function controller by linearization of the nonlinear system around the initial state and pole placement of the closed loop. We find a suitable set of poles by grid search.

Note that these controllers constitute well-performing baselines. Of course better performing controllers could be obtained at the cost of increased tuning efforts or more restrictive assumptions such as perfect nonlinear model knowledge, but this is beyond the scope of this contribution.

E. Generalization to Tracking of Unseen Reference Signals

A common criticism of data-driven/machine-learning-based solutions is that they might only perform well in cases that are covered by the training+validation dataset. To verify that ANODEC generalizes beyond reference signals from the training+validation RSD, we test the performance on various additional, qualitatively different RSDs. We denote the training RSD $\texttt{steps}(0,3)$ (cf. Section IV-D) by 'small steps', and we additionally define:

- 'big steps': $\texttt{steps}(4,8)$, (cf. Equation (10))
- 'double steps': like 'small steps', but re-draws step amplitude mid trial at $t = 5\,\mathrm{s}$,
- 'smooth':
$$\Big\{ r(t) = \Psi\left[\texttt{drawGP}(\text{seed})\right] \ \forall t \in [0,T] \ |\text{seed} \in \mathcal{U}(\mathbb{N}) \Big\},$$
- 'smooth constant': like 'smooth', but the reference becomes a constant mid trial at $t = 5\,\mathrm{s}$.

Note that only references from 'smooth' RSD are feasible.

F. ANODEC Reference Tracking Performance

We validate the proposed method (cf. Section III, IV) against the optimal-tuning and model-knowledge baseline controllers (cf. Section V-D) for each of the four systems (cf. Section V-A, V-B) and five choices of RSD (cf. Section V-E). To this end we draw, for every combination of system and RSD, 100 signals from the corresponding RSD. After forward simulation of the 100 trials, the tracking RMSE (mean across time) of the neural ODE controller is divided by the tracking RMSE of the controller with the worst performance out of the three. Finally, the $25\%/50\%/75\%-$percentiles of this relative RMSE are determined across the 100 trials. We choose to visualize the relative RMSE since the absolute RMSE varies orders of magnitude across systems and RSDs.

Results are shown in Fig. 3 and we observe: Even on previously unseen reference signals, ANODEC outperforms both baseline controllers in 12 out of the 16 cases and achieves an on average $\approx 30\,\%$ / $17\,\%$ / $7\,\%$ / $40\,\%$ lower median RMSE across all validation and test combinations compared to the optimal-tuning controller baseline / model-knowledge controller baseline / (for each combination of system and RSD) best out of the two baselines / worst out of the two baselines.

Fig. 4 showcases the performance of the trained neural ODE controller compared to the optimal-tuning baseline controller (the better out of the two baselines for this system and RSD combination) for the example of the DP2 system with two beyond-training-distribution reference signals (drawn from two RSDs). The neural ODE controller not only significantly outperforms the optimal-tuning baseline controller in a sense of error norm. It also successfully dampens occurring oscillations, unlike the optimal-tuning baseline controller, and follows reference steps not only faster but in a more controlled way.

VI. Conclusion

In this contribution we have proposed Automatic Neural ODE Control (ANODEC), a fully automatic design of feedback control for finite-time output tracking in systems with unknown nonlinear dynamics. ANODEC is able to — automatically— design controllers that outperform two baselines, and achieve an on average $\approx 30\,\%$ / $17\,\%$ lower median

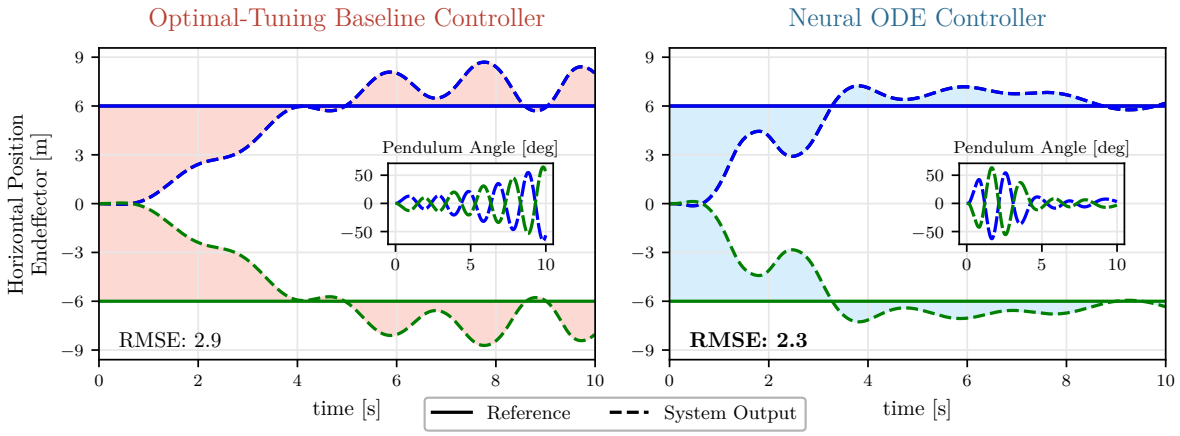

Fig. 4. Tracking performance of the optimal-tuning baseline controller (left) compared to the trained neural ODE controller (right) on two exemplary test reference signals that go *beyond* training distribution (drawn from 'big steps') in the double pendulum system (DP2). The system when controlled by the neural ODE controller reaches the reference point faster and with less oscillations. Video (https://youtu.be/tttkFFD81Qw) showcasing ANODEC on both reference signals in consecutive order (blue then green).

RMSE compared to two common approaches that either obtain a baseline controller by optimal tuning or by assuming perfect linear model knowledge around the initial state. This has been demonstrated in four different nonlinear systems using multiple, qualitatively different and even out-of-training-distribution reference signals. ANODEC achieves this whilst requiring only two (or four) minutes of interaction data. ANODEC marks an important step towards data-efficient yet performant learning control solutions that generalize across different application systems and are easy-to-use.

Future work should aim to overcome the assumption of a trial-invariant initial state and include broad experimental evaluation of the method, also on high-dimensional multiple-input multiple-output systems with potentially strong nonlinearities. Furthermore, we aim to apply ANODEC in an iterative fashion allowing for automation of a minimal (yet sufficient) data retrieval, and provide more comparison of ANODEC to state-of-the-art data-driven methods. Finally, future work should involve a theoretical analysis of ANODEC to assess stability, guarantee safety, and quantify control performance trade-offs.

References

[1] M. Deisenroth and C. Rasmussen, "PILCO: A Model-Based and Data-Efficient Approach to Policy Search.," in *ICML 2011*, 2011.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, 2017. DOI: 10.48550/arXiv.1707.06347.

[3] I. Ayed, E. de Bézenac, A. Pajot, J. Brajard, and P. Gallinari, *Learning Dynamical Systems from Partial Observations*, 2019. DOI: 10.48550/arXiv.1902.11136.

[4] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR*, 2015. DOI: 10.48550/arXiv.1507.06527.

[5] D. Wierstra, A. Forster, J. Peters, and J. Schmidhuber, "Recurrent policy gradients," *Logic Journal of IGPL*, vol. 18, no. 5, 2010. DOI: 10.1093/jigpal/jzp049.

[6] A. M. Schäfer, "Reinforcement Learning with Recurrent Neural Networks," Ph.D. dissertation, 2008.

[7] E. Schuitema, "Reinforcement learning on autonomous humanoid robots," Ph.D. dissertation, 2012. DOI: 10.4233/UUID:986EA1C5-9E30-4AAC-AB66-4F3B6B6CA002.

[8] E. Friedman and F. Fontaine, *Generalizing Across Multi-Objective Reward Functions in Deep Reinforcement Learning*, 2018. DOI: 10.48550/arXiv.1809.06364.

[9] P. Bevanda, M. Beier, S. Heshmati-Alamdari, S. Sosnowski, and S. Hirche, "Towards Data-driven LQR with Koopmanizing Flows," *IFAC-PapersOnLine*, 6th IFAC Conference, vol. 55, no. 15, 2022. DOI: 10.1016/j.ifacol.2022.07.601.

[10] G. Torrente, E. Kaufmann, P. Foehn, and D. Scaramuzza, "Data-Driven MPC for Quadrotors," *CoRR*, 2021. DOI: 10.48550/arXiv.2102.05773.

[11] D. Bristow, M. Tharayil, and A. Alleyne, "A survey of iterative learning control," *IEEE Control Systems Magazine*, vol. 26, no. 3, 2006. DOI: 10.1109/MCS.2006.1636313.

[12] M. Meindl, D. Lehmann, and T. Seel, "Bridging reinforcement learning and iterative learning control," *Frontiers in Robotics and AI*, vol. 9, 2022. DOI: 10.3389/frobt.2022.793512.

[13] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural Ordinary Differential Equations," in *NeurIPS*, vol. 31, 2018. DOI: 10.48550/arXiv.1806.07366.

[14] P. Kidger, J. Morrill, J. Foster, and T. Lyons, "Neural Controlled Differential Equations for Irregular Time Series," in *NeurIPS*, vol. 33, 2020. DOI: 10.48550/arXiv.2005.08926.

[15] T. Asikis, L. Böttcher, and N. Antulov-Fantulin, "Neural ordinary differential equation control of dynamics on graphs," *Physical Review Research*, vol. 4, no. 1, 2022. DOI: 10.1103/PhysRevResearch.4.013221.

[16] I. O. Sandoval, P. Petsagkourakis, and E. A. del Rio-Chanona, "Neural ODEs as Feedback Policies for Nonlinear Optimal Control," 2022. DOI: 10.48550/arXiv.2210.11245.

[17] K.-K. Kim, "Stability analysis of riccati differential equations related to affine diffusion processes," *Journal of Mathematical Analysis and Applications*, vol. 364, 2010. DOI: 10.1016/j.jmaa.2009.11.020.

[18] S. Skogestad, M. Morari, and J. Doyle, "Robust control of ill-conditioned plants: High-purity distillation," *IEEE Transactions on Automatic Control*, vol. 33, no. 12, 1988. DOI: 10.1109/9.14431.

[19] D. Ho and J. Karl Hedrick, "Control of nonlinear non-minimum phase systems with input-output linearization," in *2015 ACC*, 2015. DOI: 10.1109/ACC.2015.7171957.

[20] J. Bradbury, R. Frostig, P. Hawkins, *et al.*, *JAX: Composable transformations of Python+NumPy programs*, 2018.

[21] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IROS*, IEEE, 2012. DOI: 10.1109/IROS.2012.6386109.