Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Reinforcement Learning and Trajectory Planning based on Model Approximation with Neural Networks applied to Transition Problems

Max Pritzkoleit * **Carsten Knoll** * **Klaus Röbenack** *

* *TU Dresden, Faculty of Electrical and Computer Engineering, Institute of Control Theory, Dresden, Germany (e-mail: {max.pritzkoleit,carsten.knoll,klaus.roebenack}@tu-dresden.de).*

Abstract: In this paper we use a multilayer neural network to approximate the dynamics of nonlinear (mechanical) control systems. Furthermore, these neural network models are combined with offline trajectory planning, to form a model-based reinforcement learning (RL) algorithm, suitable for transition problems of nonlinear dynamical systems. We evaluate the algorithm on the swing-up of the cart-pole benchmark system and observe a significant performance gain in terms of data efficiency compared to a state-of-the-art model-free RL method (Deep Deterministic Policy Gradient (DDPG)). Additionally, we present first experimental results on a cart-triple-pole system test bench. For a simple transition problem, the proposed algorithm shows a good controller performance.

Keywords: Trajectory planning, Reinforcement learning, Learning control, Neural-network models, Model approximation, Tracking control,

1. INTRODUCTION

In recent years, reinforcement learning gained a lot of attention in- and outside the scientific community through advances made possible by incorporationg deep neural networks for function approximation. These model-free [1] algorithms (i.e. DDPG, DQN) try to approximate the Hamiltonian (often called "Q-function") of the underlying optimal control problem based on data, which is generated by interacting with the system. The main drawback of these algorithms is the poor data efficiency, which means a lot of data has to be gathered for convergence. This makes it hard to apply them to real world systems due to wear-out and time consumption.

In contrast, model-based algorithms, which learn a representation of the system dynamics, need less data to achieve a certain task (i.e., equilibrium transition) when applicable. These algorithms try to automate the classical process of obtaining a model of the system dynamics by first principles, identification of physical parameters, and controller design. In the context of reinforcement learning, a priori knowledge about the system is assumed to be not available or very limited. This makes the control problem unnecessarily hard from a control theoretic point of view, but reduces model bias and empowers generalization of the learning algorithm.

In this work we combine neural network dynamics models with trajectory planning to form a model-based reinforcement learning algorithm. This approach is similar to Yamaguchi and Atkeson (2016), where a stochastic trajectory planning is performed. Other works use Locally Weighted Projection Regression (LWPR) (Mitrovic et al., 2008) or Gaussian Process (GP) (Bechtle et al., 2019; Lee et al., 2017) for model learning.

This paper is structured as follows: In Sec. 2 the basic concepts of model-based reinforcement learning are presented. In Sec. 3 it is shown, how a neural network can be used to learn a dynamics model. In Sec. 4 the iterative Linear-Quadratic Regulator (iLQR) is described, which is used for trajectory planning. Finally we combine the two concepts to form a model-based RL algorithm in Sec. 5 and present results in simulation and on a test bench in Sec. 6.

2. PRELIMINARIES

Reinforcement learning (RL) deals with general decision making problems similar to optimal control. In reinforcement learning, an agent interacts with a system by applying a control $\boldsymbol{u}_k \in \mathbb{U}_k$ to it. The system (1) transits from a state $\boldsymbol{x}_k \in \mathbb{X}_k$ at time step k to a successor state $\boldsymbol{x}_{k+1} \in \mathbb{X}_{k+1}$ at time step $k+1$ and emits a cost signal $c_k \in \mathbb{R}$, which holds information about the fullfilment of the given task.

In contrast to the classical RL setting, the system dynamics

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{1}$$

consists of a deterministic sampled-data system with underlying continuous dynamics and the incremental cost function

$$c_k : \mathbb{X}_k \times \mathbb{U}_k \mapsto \mathbb{R} \tag{2}$$

[1] Model-free in the sense, that the system dynamics are not explicitly represented.

is known.

The agent's goal is to minimize the total cost

$$J = \sum_{k=0}^{N-1} c_k(\boldsymbol{x}_k, \boldsymbol{u}_k) + c_N(\boldsymbol{x}_N) \tag{3}$$

over a given time-horizon N, starting from an inital state $\boldsymbol{x}_0$ by choosing appropriate controls. If the system transits to a terminal state $\boldsymbol{x}_{k+1} \in \mathbb{X}^-$ or $k = N$, the episode terminates, and the system is reset to $\boldsymbol{x}_0$. At every time step the transition tuple $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$, where $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$, is added to a data set $\mathcal{D} = \{(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')_i\}_{i=0}^{T}$, where T is the total experience of the agent.

2.1 Problem statement

This leads to a classical formulation of the optimal control problem:

$$\min_{\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1}} \quad J(\boldsymbol{x}_0, \boldsymbol{u}_0, ..., \boldsymbol{u}_{N-1}) \tag{4a}$$

$$\text{s.t. } \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad \forall k \in [0, N-1], \tag{4b}$$

$$\boldsymbol{u}_k \in \mathbb{U}_k, \boldsymbol{x}_k \in \mathbb{X}_k, \quad \text{giv. } \boldsymbol{x}_0. \tag{4c}$$

2.2 Transition problems

It is assumed, that $\boldsymbol{x}_0$ is a stable equilibrium of (1) and the goal is to drive the system into a goal state $\boldsymbol{x}^\star$ in N time steps. An equilibrium $\boldsymbol{x}_e$ of system (1) is defined as:

$$\boldsymbol{x}_e = \boldsymbol{f}(\boldsymbol{x}_e, \boldsymbol{u}_e), \tag{5}$$

where $\boldsymbol{u}_e \in \mathbb{U}$ is an admissable control, such that (5) is satisfied. To transcribe the transition problem to an optimal control problem (4), the cost function has to be designed, such that the cost in the equilibrium vanishes:

$$c_k(\boldsymbol{x}_e, \boldsymbol{u}_e) \stackrel{!}{=} 0, \quad c_N(\boldsymbol{x}_e) \stackrel{!}{=} 0. \tag{6}$$

A quadratic cost function is thus a suitable choice.

2.3 Model-based RL

In the context of RL, the dynamics $\boldsymbol{f}$ are initially unkown and problem (4) can't be solved. Therefore, in model-based RL the dynamics are approximated by a parameterized model $\boldsymbol{f}_{\boldsymbol{\theta}}$ using $\mathcal{D}$. With the learned dynamics model an approximate optimal control problem can be solved:

$$\min_{\boldsymbol{u}_0,\dots,\boldsymbol{u}_{N-1}} \quad J(\boldsymbol{x}_0, \boldsymbol{u}_0, ..., \boldsymbol{u}_{N-1}) \tag{7a}$$

$$\text{s.t. } \boldsymbol{x}_{k+1} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad \forall k \in [0, N-1] \tag{7b}$$

$$\boldsymbol{u}_k \in \mathbb{U}_k, \boldsymbol{x}_k \in \mathbb{X}_k, \quad \text{giv. } \boldsymbol{x}_0 \tag{7c}$$

There are numerous ways to solve problem (7). In *policy search* the global solution is approxmiated by optimizing a parametrized control law (policy) $\pi(\boldsymbol{x}; \boldsymbol{\phi})$, see e.g. Deisenroth and Rasmussen (2011). Nagabandi et al. (2018) and Lee et al. (2017); Mitrovic et al. (2008) use a model predictive control scheme, where (7) is solved over a short horizon $\leq N$ at every time step.

The optimized control law is applied to system (1) and the resulting trajectory $\boldsymbol{\tau} = \{(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')_i\}_{i=0}^{N}$ is added to $\mathcal{D}$. With the aggregated data set, the model $\boldsymbol{f}_{\boldsymbol{\theta}}$ can be improved and the process repeats until a sufficiently good control law is found, as summerized in Alg. 1.

Algorithm 1 Model-based RL

Require: initial model $\boldsymbol{f}_{\boldsymbol{\theta}}$, data set $\mathcal{D}$
  initialize $\mathcal{D}$ by random sampling
  while not converged **do**
    1) train $\boldsymbol{f}_{\boldsymbol{\theta}}$ using $\mathcal{D}$
    2) solve the optimal control problem (7)
    3) rollout the resulting control law on system (1)
    4) add the resulting trajectory $\boldsymbol{\tau}$ to the data set $\mathcal{D}$
  end while

3. LEARNING A NEURAL NETWORK DYNAMICS MODEL

To approximate the dynamics, the neural network proposed by Nagabandi et al. (2018) is used.

The dynamics $\boldsymbol{f}$ are split into two terms:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_k). \tag{8}$$

To approximate the difference model $\boldsymbol{f}_d$, a feedforward neural network $\boldsymbol{f}_{d,\boldsymbol{\theta}}$ is used:

$$\boldsymbol{f}_{d,\boldsymbol{\theta}}(\boldsymbol{x}_k, \boldsymbol{u}_k) \stackrel{!}{\approx} \boldsymbol{f}_d(\boldsymbol{x}_k, \boldsymbol{u}_k). \tag{9}$$

The network is composed of two hidden layers and uses ReLU activation functions:

$$\boldsymbol{h}^{[1]} = \text{ReLU}\left(\boldsymbol{W}^{[0]}\left(\boldsymbol{x}_k^{\mathrm{T}}, \boldsymbol{u}_k^{\mathrm{T}}\right)^{\mathrm{T}} + \boldsymbol{b}^{[0]}\right), \tag{10a}$$

$$\boldsymbol{h}^{[2]} = \text{ReLU}\left(\boldsymbol{W}^{[1]}\boldsymbol{h}^{[1]} + \boldsymbol{b}^{[1]}\right), \tag{10b}$$

$$\boldsymbol{f}_{d,\boldsymbol{\theta}} = \boldsymbol{W}^{[2]}\boldsymbol{h}^{[2]} + \boldsymbol{b}^{[2]}. \tag{10c}$$

Hereby $\boldsymbol{h}^{[1]}$ and $\boldsymbol{h}^{[2]}$ denote the intermediate results of the internal network layers.

The networks weight matrices and bias vectors together form the parameter vector $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} := \{\boldsymbol{W}^{[0]}, \boldsymbol{b}^{[0]}; \boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}; \boldsymbol{W}^{[2]}, \boldsymbol{b}^{[2]}\}.$$

To learn an approximate dynamics model, the mean-sqared error (MSE) of the difference model derived from (8) and (9) is minimized:

$$\min_{\boldsymbol{\theta}} \frac{1}{T}\sum_{i=0}^{T} \left\| \left(\boldsymbol{x}'^{(i)} - \boldsymbol{x}^{(i)}\right) - \boldsymbol{f}_{d,\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}\right) \right\|_2^2, \tag{11}$$

where the superscript $^{(i)}$ labels data points in the i-th tuple of $\mathcal{D}$. Due to a possibly different range of values in the state dimensions the error residuals are not weighted equally. As proposed in Nagabandi et al. (2018), we use data standardization to preprocess the training data.

3.1 Data Standardization

The data standardization is performed by computing the standard deviation $\boldsymbol{\sigma}$ and mean $\boldsymbol{\mu}$ vectors of $\Delta\boldsymbol{x}_k := \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$, $\boldsymbol{x}_k$ and $\boldsymbol{u}_k$ over the whole data set $\mathcal{D}$:

$$\boldsymbol{\mu}_{\Delta\boldsymbol{x}} := \mathbb{E}_{\mathcal{D}}[\Delta\boldsymbol{x}] \qquad \boldsymbol{\sigma}_{\Delta\boldsymbol{x}} := \sqrt{\text{Var}_{\mathcal{D}}[\Delta\boldsymbol{x}]}$$

$$\boldsymbol{\mu}_{\boldsymbol{x}} := \mathbb{E}_{\mathcal{D}}[\boldsymbol{x}] \qquad \boldsymbol{\sigma}_{\boldsymbol{x}} := \sqrt{\text{Var}_{\mathcal{D}}[\boldsymbol{x}]}$$

$$\boldsymbol{\mu}_{\boldsymbol{u}} := \mathbb{E}_{\mathcal{D}}[\boldsymbol{u}] \qquad \boldsymbol{\sigma}_{\boldsymbol{u}} := \sqrt{\text{Var}_{\mathcal{D}}[\boldsymbol{u}]}$$

and applying the following transformation to the data:

$$\Delta\hat{\boldsymbol{x}}_k := \text{diag}(\boldsymbol{\sigma}_{\Delta\boldsymbol{x}}^{-1})(\Delta\boldsymbol{x} - \boldsymbol{\mu}_{\Delta\boldsymbol{x}}),$$

$$\hat{\boldsymbol{x}}_k := \text{diag}(\boldsymbol{\sigma}_{\boldsymbol{x}}^{-1})(\boldsymbol{x} - \boldsymbol{\mu}_{\boldsymbol{x}}),$$

$$\hat{\boldsymbol{u}}_k := \text{diag}(\boldsymbol{\sigma}_{\boldsymbol{u}}^{-1})(\boldsymbol{u} - \boldsymbol{\mu}_{\boldsymbol{u}}).$$

The neural network $\hat{\boldsymbol{f}}_{d,\boldsymbol{\theta}}$ is trained in the standardized coordinates:

$$\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_{i=0}^{T} \left\| \Delta \hat{\boldsymbol{x}}^{(i)} - \hat{\boldsymbol{f}}_{d,\boldsymbol{\theta}} \left(\hat{\boldsymbol{x}}^{(i)}, \hat{\boldsymbol{u}}^{(i)} \right) \right\|_2^2. \tag{13}$$

A retransformation to the original coordinates leads to the final model:

$$\boldsymbol{f}_{d,\boldsymbol{\theta}} := \operatorname{diag}(\boldsymbol{\sigma}_{\Delta \boldsymbol{x}}) \hat{\boldsymbol{f}}_{d,\boldsymbol{\theta}} \left(\hat{\boldsymbol{x}}_k, \hat{\boldsymbol{u}}_k \right) + \boldsymbol{\mu}_{\Delta \boldsymbol{x}}. \tag{14}$$

3.2 Data acquisition heuristic

When the agent interacts with the system and collects data points $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$, these data points are not sampled equally distributed in the $\mathbb{X} \times \mathbb{U} \times \mathbb{X}$ space. At the beginning of the learning process they are obviously dense near the initial state $(\boldsymbol{x}_0, \star, \star)$. When a trajectory is rolled out, that does not lie in the distribution of the data set $\mathcal{D}$, these new data points are sparse and therefore treated as outliers in the training process. To make sure the neural network model also captures the information of these data points the density of the data in $\mathcal{D}$ has to be bounded. Therefore, a simple heuristic is used, that makes sure a data point is only added to $\mathcal{D}$ if it is not already captured by the model. Namely, if the prediction error of the latest sample $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$ is higher than a specified bound ϵ, the sample is added to $\mathcal{D}$:

$$\left\| \Delta \boldsymbol{x}^{(i)} - \boldsymbol{f}_{d,\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)} \right) \right\|_2^2 > \epsilon. \tag{15}$$

3.3 Exploiting the Model Structure of Mechanical Systems

When dealing with mechanical systems, some properties of the system dynamics can be exploited to ease the learning process.

When adopting a Lagrangian perspective, the state of a mechanical system $\boldsymbol{x}$ is composed of its generalized coordinates and the corresponding velocities $\boldsymbol{x} := \begin{pmatrix} \boldsymbol{q}^{\mathrm{T}} & \dot{\boldsymbol{q}}^{\mathrm{T}} \end{pmatrix}^{\mathrm{T}}$. The continuous time state space model is then given by:

$$\dot{\boldsymbol{x}} = \begin{pmatrix} \dot{\boldsymbol{q}} \\ \boldsymbol{g}(\dot{\boldsymbol{q}}, \boldsymbol{q}, \boldsymbol{u}) \end{pmatrix}, \tag{16}$$

where $\ddot{\boldsymbol{q}} = \boldsymbol{g}$ are the relations stemming from the Lagrangian equations of motion (which are considered as a first principle). For a sufficiently small sample-time Δt an approximation of the discrete time dynamics is given by:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta t \begin{pmatrix} \dot{\boldsymbol{q}}_k \\ \boldsymbol{g}(\dot{\boldsymbol{q}}_k, \boldsymbol{q}_k, \boldsymbol{u}_k) \end{pmatrix}. \tag{17}$$

The upper hyper-row of the system dynamics consists of known definitional equations which can be exactly represented (i. e. hardcoded) easily by a neural network. Thus only the mapping $\boldsymbol{g}$, which models the acceleration dynamics has to be learned.

If the system has angles as generalized coordinates, the system dynamics are typically not dependent on the absolute value of the angle, but depend on its sine and/or cosine components. Thus, instead of the state component x_i the two pseudo-state-components $(\sin x_i, \cos x_i)$ can be considered. If this is done for all angle-coordinates, one obtains the pseudo-state-vector $\boldsymbol{o}$. In the learning process, $\boldsymbol{o}$ is used instead of $\boldsymbol{x}$ as the input of the neural network, but its output remains $\Delta \boldsymbol{x}$.

4. TRAJECTORY PLANNING

For solving the approximate optimal control problem (7) with (14) as $\boldsymbol{f}_{\boldsymbol{\theta}}$, trajectory planning techniques can be used. Instead of only solving for the open loop controls $\boldsymbol{u}_0^*, ..., \boldsymbol{u}_{N-1}^*$, the optimization performed in this work leads to a time-varying feedback-controller with feedforward:

$$\boldsymbol{u}_k = \boldsymbol{K}_k (\boldsymbol{x}_k - \boldsymbol{x}_k^*) + \boldsymbol{u}_k^*, \forall k \in [0, N-1], \tag{18}$$

that stabilizes the system around the optimized state trajectory $\boldsymbol{x}_0^*, ..., \boldsymbol{x}_{N-1}^*$.

4.1 Iterative Linear-Quadratic Regulator (iLQR)

The iLQR is a second-order gradient method based on Differential Dynamic Programming (DDP) (Mayne, 1966), that solves problem (4) in an iterative fashion. In this work, the control-limited variant of iLQR from Tassa et al. (2014) is used. In this section, a brief overview of the algorithm is presented, for details see Li and Todorov (2004); Tassa et al. (2014). Neglecting a regularization heuristic and the performed line-search, the basic algoritm can be boiled down to three steps as shown in Alg. 2.

Algorithm 2 iLQR (simplified)

Require: $\boldsymbol{f}$, c_k, c_N
  choose initial controls $\boldsymbol{U}^{[0]} := \{\boldsymbol{u}_0^{[0]}, ..., \boldsymbol{u}_{N-1}^{[0]}\}$
  $\boldsymbol{X}^{[0]} := \{\boldsymbol{x}_0^{[0]}, ..., \boldsymbol{x}_N^{[0]}\}$ rollout by applying $\boldsymbol{U}^{[0]}$ to $\boldsymbol{f}$
  for $j = [0, ..., j_{\max}]$ **do**
    1) approximate (4) around $\boldsymbol{\tau}^{[j]} := \{\boldsymbol{X}^{[j]}, \boldsymbol{U}^{[j]}\}$
    2) solve the resulting time-varying LQR problem
    3) $\boldsymbol{\tau}^{[j+1]}$: rollout the resulting control law on (1)
    if $J(\boldsymbol{\tau}^{[j]}) - J(\boldsymbol{\tau}^{[j+1]}) < \epsilon_J$ **then**
      break
    end if
  end for
  return $\boldsymbol{\tau}^* = \{\boldsymbol{X}^*, \boldsymbol{U}^*\} := \boldsymbol{\tau}^{[j+1]}$

1) Local approximation of the optimal control problem Given the current trajectory $\{\bar{\boldsymbol{X}}, \bar{\boldsymbol{U}}\} =: \boldsymbol{\tau}^{[j]}$ a taylor expansion of the system dynamics and cost is performed around it. In contrast to DDP, iLQR only uses a first order expansion of the dynamics:

$$\tilde{\boldsymbol{x}}_{k+1} \approx \boldsymbol{A}_k \tilde{\boldsymbol{x}}_k + \boldsymbol{B}_k \tilde{\boldsymbol{u}}_k, \tag{19}$$

with

$$\tilde{\boldsymbol{x}}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k, \quad \tilde{\boldsymbol{u}}_k = \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k, \forall k. \tag{20}$$

The cost approximation is of second order:

$$\tilde{c}_k \approx \begin{pmatrix} \tilde{\boldsymbol{x}}_k^{\mathrm{T}} & \tilde{\boldsymbol{u}}_k^{\mathrm{T}} \end{pmatrix} \left[\begin{pmatrix} \boldsymbol{c}_{\boldsymbol{x},k} \\ \boldsymbol{c}_{\boldsymbol{u},k} \end{pmatrix} + \frac{1}{2} \begin{pmatrix} \boldsymbol{C}_{\boldsymbol{x}\boldsymbol{x},k} & \boldsymbol{C}_{\boldsymbol{u}\boldsymbol{x},k} \\ \boldsymbol{C}_{\boldsymbol{x}\boldsymbol{u},k} & \boldsymbol{C}_{\boldsymbol{u}\boldsymbol{u},k} \end{pmatrix} \begin{pmatrix} \tilde{\boldsymbol{x}}_k \\ \tilde{\boldsymbol{u}}_k \end{pmatrix} \right], \tag{21a}$$

$$\tilde{c}_N \approx \tilde{\boldsymbol{x}}_N^{\mathrm{T}} \boldsymbol{c}_{\boldsymbol{x},N} + \frac{1}{2} \tilde{\boldsymbol{x}}_N^{\mathrm{T}} \boldsymbol{C}_{\boldsymbol{x}\boldsymbol{x},N} \tilde{\boldsymbol{x}}_N. \tag{21b}$$

This leads to the optimization objective:

$$\tilde{J} = \sum_{k=0}^{N-1} \tilde{c}_k(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k) + \tilde{c}_N(\tilde{\boldsymbol{x}}_N). \tag{22}$$

2) Solving the local approximation of the optimal control problem From the linear-quadratic cost function (22) and the linear time-varying dynamics (19) an optimal control problem can formulated:

$$\min_{\tilde{\boldsymbol{u}}_0,\dots,\tilde{\boldsymbol{u}}_{N-1}} \tilde{J}(\tilde{\boldsymbol{x}}_0, \tilde{\boldsymbol{u}}_0, ..., \tilde{\boldsymbol{u}}_{N-1}), \tag{23a}$$

$$\text{s.t. } \tilde{\boldsymbol{x}}_{k+1} = \boldsymbol{A}_k\tilde{\boldsymbol{x}}_k + \boldsymbol{B}_k\tilde{\boldsymbol{u}}, \forall k \in [0, N-1], \tag{23b}$$

$$\boldsymbol{u}_k \in \mathbb{U}_k, \boldsymbol{x}_k \in \mathbb{X}_k, \text{giv. } \boldsymbol{x}_0. \tag{23c}$$

Problem (23) can be solved by dynamic programming (Bertsekas, 2005):

$$\tilde{V}_N(\tilde{\boldsymbol{x}}_N) = \tilde{c}_N(\tilde{\boldsymbol{x}}_N), \tag{24a}$$

$$\tilde{V}_k(\tilde{\boldsymbol{x}}_k) = \min_{\tilde{\boldsymbol{u}}_k} \underbrace{\left[\tilde{c}_k(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k) + \tilde{V}_{k+1}(\tilde{\boldsymbol{x}}_{k+1})\right]}_{=:\tilde{Q}_k(\tilde{\boldsymbol{x}}_k, \tilde{\boldsymbol{u}}_k)}. \tag{24b}$$

The optimal controller can be derived in analogy to the discrete-time LQR backwards in time starting from $k = N-1$ with

$$\boldsymbol{V}_{\boldsymbol{xx},N} = \boldsymbol{C}_{\boldsymbol{xx},N} \quad \boldsymbol{v}_{\boldsymbol{x},N} = \boldsymbol{c}_{\boldsymbol{x},N}. \tag{25}$$

The Hamiltonian $\tilde{Q}_k$ is linear-quadratic:

$$\tilde{Q}_k = \begin{pmatrix}\tilde{\boldsymbol{x}}_k^{\mathrm{T}} & \tilde{\boldsymbol{u}}_k^{\mathrm{T}}\end{pmatrix}\left[\begin{pmatrix}\boldsymbol{q}_{\boldsymbol{x},k}\\ \boldsymbol{q}_{\boldsymbol{u},k}\end{pmatrix} + \frac{1}{2}\begin{pmatrix}\boldsymbol{Q}_{\boldsymbol{xx},k} & \boldsymbol{Q}_{\boldsymbol{ux},k}\\ \boldsymbol{Q}_{\boldsymbol{xu},k} & \boldsymbol{Q}_{\boldsymbol{uu},k}\end{pmatrix}\begin{pmatrix}\tilde{\boldsymbol{x}}_k\\ \tilde{\boldsymbol{u}}_k\end{pmatrix}\right], \tag{26}$$

with

$$\boldsymbol{Q}_{\boldsymbol{xx},k} = \boldsymbol{C}_{\boldsymbol{xx},k} + \boldsymbol{A}_k^{\mathrm{T}}\boldsymbol{V}_{\boldsymbol{xx},k+1}\boldsymbol{A}_k, \tag{27a}$$

$$\boldsymbol{Q}_{\boldsymbol{xu},k} = \boldsymbol{Q}_{\boldsymbol{xu},k}^{\mathrm{T}} = \boldsymbol{C}_{\boldsymbol{xu},k} + \boldsymbol{A}_k^{\mathrm{T}}\boldsymbol{V}_{\boldsymbol{xx},k+1}\boldsymbol{B}_k, \tag{27b}$$

$$\boldsymbol{Q}_{\boldsymbol{uu},k} = \boldsymbol{C}_{\boldsymbol{uu},k} + \boldsymbol{B}_k^{\mathrm{T}}\boldsymbol{V}_{\boldsymbol{xx},k+1}\boldsymbol{B}_k, \tag{27c}$$

$$\boldsymbol{q}_{\boldsymbol{x},k} = \boldsymbol{c}_{\boldsymbol{x},k} + \boldsymbol{A}_k^{\mathrm{T}}\boldsymbol{v}_{\boldsymbol{x},k+1}, \tag{27d}$$

$$\boldsymbol{q}_{\boldsymbol{u},k} = \boldsymbol{c}_{\boldsymbol{u},k} + \boldsymbol{B}_k^{\mathrm{T}}\boldsymbol{v}_{\boldsymbol{x},k+1}. \tag{27e}$$

Solving for the optimal control law leads to the following feedback law with feedforward:

$$\tilde{\boldsymbol{u}}_k^* = \boldsymbol{K}_k\tilde{\boldsymbol{x}}_k + \boldsymbol{k}_k, \forall k \in [0, N-1]. \tag{28}$$

The feedback matrix and feedforward are given by:

$$\boldsymbol{K}_k = -\boldsymbol{Q}_{\boldsymbol{uu},k}^{-1}\boldsymbol{Q}_{\boldsymbol{ux},k}, \quad \boldsymbol{k}_k = -\boldsymbol{Q}_{\boldsymbol{uu},k}^{-1}\boldsymbol{q}_{\boldsymbol{u},k}. \tag{29}$$

Substituting the optimal control law (28) into the Hamiltonian (26) results in the cost-to-go:

$$\tilde{V}_k(\tilde{\boldsymbol{x}}_k) = \tilde{\boldsymbol{x}}_k^{\mathrm{T}}\boldsymbol{v}_{\boldsymbol{x},k} + \frac{1}{2}\tilde{\boldsymbol{x}}_k^{\mathrm{T}}\boldsymbol{V}_{\boldsymbol{xx},k}\tilde{\boldsymbol{x}}_k + \text{const.}, \tag{30}$$

with

$$\boldsymbol{V}_{\boldsymbol{xx},k} = \boldsymbol{Q}_{\boldsymbol{xx},k} + \boldsymbol{K}_k^{\mathrm{T}}(\boldsymbol{Q}_{\boldsymbol{ux},k} + \boldsymbol{Q}_{\boldsymbol{uu},k}\boldsymbol{K}_k) + \boldsymbol{Q}_{\boldsymbol{xu},k}\boldsymbol{K}_k, \tag{31a}$$

$$\boldsymbol{v}_{\boldsymbol{x},k} = \boldsymbol{q}_{\boldsymbol{x},k} + \boldsymbol{K}_k^{\mathrm{T}}(\boldsymbol{q}_{\boldsymbol{u},k} + \boldsymbol{Q}_{\boldsymbol{uu},k}\boldsymbol{k}_k) + \boldsymbol{Q}_{\boldsymbol{xu},k}\boldsymbol{k}_k. \tag{31b}$$

3) Rollout of the optimal control law The optimal control law (28) for problem (23) can be solved for $\boldsymbol{u}_k$ resulting in (18):

$$\boldsymbol{u}_k = \boldsymbol{K}_k(\boldsymbol{x}_k - \underbrace{\bar{\boldsymbol{x}}_k}_{=:\boldsymbol{x}_k^*}) + \underbrace{\boldsymbol{k}_k + \bar{\boldsymbol{u}}_k}_{=:\boldsymbol{u}_k^*}, \forall k. \tag{32}$$

Applying (18) to (1) leads to a new trajectory starting from $\boldsymbol{x}_0 := \bar{\boldsymbol{x}}_0\ \forall k \in [0, N-1]$:

$$\boldsymbol{u}_k = \boldsymbol{K}_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \boldsymbol{k}_k + \bar{\boldsymbol{u}}_k, \tag{33a}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k). \tag{33b}$$

The new trajectory $\boldsymbol{\tau}^{[j+1]} := \{\boldsymbol{X}, \boldsymbol{U}\}$ is used for the next iteration.

4.2 Computing the final control law

Dealing with tranisition problems, it is assumed that $\boldsymbol{x}_k = \boldsymbol{x}_e$ and $\boldsymbol{u}_k = \boldsymbol{u}_e$ for $k > N$. To stabilize the system in $\boldsymbol{x}_e$, a standard method is to design a LQ controller

$$\boldsymbol{u}_k = \boldsymbol{K}_{LQR}(\boldsymbol{x}_k - \boldsymbol{x}_e) + \boldsymbol{u}_e, \quad k > N. \tag{34}$$

Under the given assumptions $\boldsymbol{K}_N = \boldsymbol{K}_{LQR}$ can be achieved, i.e. the time-varying policy (18) transits smoothly to the constant LQR policy. To accomplish this, the DARE [2] of the LQR problem for $\boldsymbol{A}_e = \partial_{\boldsymbol{x}}\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_e, \boldsymbol{u}_e)$ and $\boldsymbol{B}_e = \partial_{\boldsymbol{u}}\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_e, \boldsymbol{u}_e)$ is solved, with $\boldsymbol{R} := \boldsymbol{C}_{\boldsymbol{uu},N}, \boldsymbol{Q} := \boldsymbol{C}_{\boldsymbol{xx},N}, \boldsymbol{N} := \boldsymbol{C}_{\boldsymbol{xu},N}$. The resulting matrix $\boldsymbol{P}$, which is the solution of the DARE, is used to initialize the backward pass in step 2):

$$\boldsymbol{V}_{\boldsymbol{xx},N} := \boldsymbol{P}, \quad \boldsymbol{v}_{\boldsymbol{x},N} := \boldsymbol{0}. \tag{35}$$

However, while the trajectory optimization yields $\boldsymbol{x}_N^\star \approx \boldsymbol{x}_e$ it does not guarantee $\boldsymbol{x}_N^\star \equiv \boldsymbol{x}_e$. To stabilize the system in the desired state $\boldsymbol{x}_e$, for $k > N$, the following control law is applied instead of (18):

$$\boldsymbol{u}_k = \boldsymbol{K}_N(\boldsymbol{x}_k - \boldsymbol{x}_e) + \boldsymbol{u}_e. \tag{36}$$

5. COMBINING MODEL APPROXIMATION AND TRAJECTORY OPTIMZATION

The proposed neural network model in Sec. 3 for approximating the dynamics (1) and the trajectory planning approach in Sec. 4 can be combined to form the model-based reinforcement learning algorithm Alg. 3, which is a special case of Alg. 1.

Algorithm 3 Model-based RL with offline trajectory planning

Require: model $\boldsymbol{f}_{d,\boldsymbol{\theta}}$, data set $\mathcal{D}$
  initialize $\mathcal{D}$ by random sampling
  while not converged **do**
    1) train $\boldsymbol{f}_{d,\boldsymbol{\theta}}$ using (13) on the data set $\mathcal{D}$
    2) solve the optimal control problem (7) with iLQR
    3) apply the resulting policy (18) on system (1)
    4) add the resulting trajectory $\boldsymbol{\tau}$ to the data set $\mathcal{D}$
  end while

6. RESULTS

6.1 Simulation Results [3]

As a first simulation experiment, Alg. 3 is applied to the cart-pole system (Fig. 1).

The state is composed of the position q_0 and velocity $\dot{q}_0$ of the cart, as well as the angle q_1 and angular velocity $\dot{q}_1$ of the pole:

$$\boldsymbol{x} := \begin{pmatrix}q_0 & q_1 & \dot{q}_0 & \dot{q}_1\end{pmatrix}. \tag{37}$$

The system is controlled by the force F_0, that acts on the cart. The continuous dynamics in an input-output linearized form, where the control input $\boldsymbol{u} = (u_1)$ is the acceleration of the cart $\ddot{q}_0$, are given by:

[2] Discrete Algebraic Riccati Equation
[3] Code available at: `https://github.com/TUD-RST/pygent`

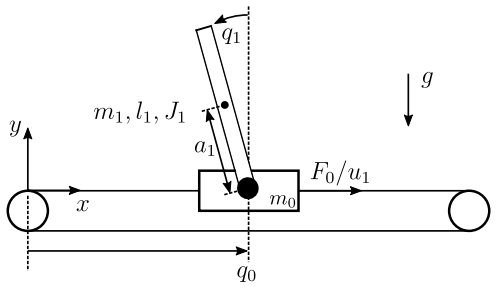

Fig. 1. Cart-pole system

$$\dot{x}_1 = x_3, \quad \dot{x}_2 = x_4, \quad \dot{x}_3 = u_1, \tag{38a}$$

$$\dot{x}_4 = a_1 m_1 (g \sin(x_2) + u_1 \cos(x_2)) - \frac{d_1}{J_1 + a_1^2 m_1} x_4, \tag{38b}$$

where a_1 is the position of the center of gravity of the pole, m_1 is the mass of the pole, J_1 is the angular momentum of the pole, d_1 is a damping coefficient, and g is gravity. The control objective is to swing-up and balance the pole in the upward unstable equlibrium $\boldsymbol{x}^\star = (0\ 0\ 0\ 0)^\mathrm{T}$ starting from $\boldsymbol{x}_0 = (0\ \pi\ 0\ 0)^\mathrm{T}$.

Controller performance The learning curve of Alg. 3 applied to the cart-pole system is shown in Fig. 2. For the first five episodes, the agent samples controls from a uniform distribution to initialize the data set $\mathcal{D}$. At the 14-th episode (about 2500 time steps), the swing-up and stabilization of the pole in the upward position is accomplished for the first time. After 19 episodes (about 3600 time steps), the learning based controller asymptotically matches the performance of a controller, that has access to the true system dynamics. Compared to DDPG (Lillicrap et al., 2016), a popular model-free RL method, our approach is about 70 times more data-efficient on the swing-up task and has a better final performance, cf. (Pritzkoleit, 2019).

Prediction performance of the approximated model To examine the prediction quality of the neural network model, we consider the error along the (optimal) trajectory [4], i. e., $\boldsymbol{e}(k) := \hat{\boldsymbol{f}}_{d,\boldsymbol{\theta}}(\boldsymbol{x}_k^*, \boldsymbol{u}_k^*) - \boldsymbol{f}_d(\boldsymbol{x}_k^*, \boldsymbol{u}_k^*)$ at different training stages (episodes). Because the first two components of $\boldsymbol{f}$ are definitional equations (cf. (17) and (38a)), only the last two components – which correspond to the acceleration of the cart and the pendulum angle – have to be considered. From Fig. 3 we see good convergence with increasing episode number. Furthermore, we notice that, even after about 49 episodes the approximated model still differs perceptibly but that does not impair the algorithm from successfully performing the swing-up task.

6.2 Experimental results

To further investigate the performance of Alg. 3, it is applied on a real cart-triple-pole test bench system. The goal of the experiment is to side-step the downward hanging poles from $q_0 = 0$ to $q_0 = 0.8$, while the poles must rest at the end of the transition. To initialize the data set $\mathcal{D}$ we first planned a twice differentiable

[4] The optimal trajectory was calculated with the true dynamics using iLQR.

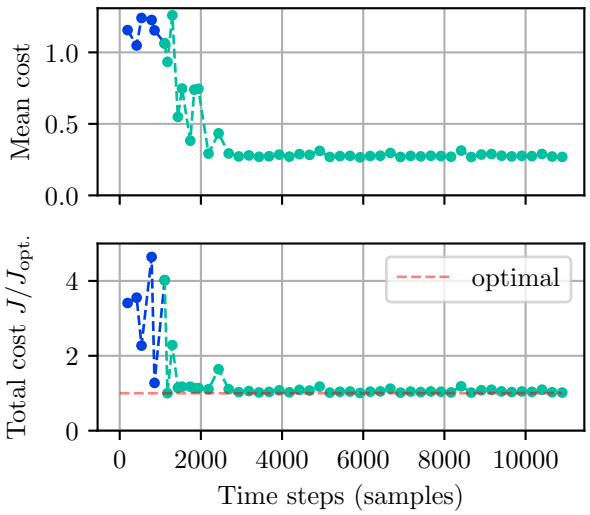

Fig. 2. Learning Process: After around 3600 time steps (72 s interaction time), the learning based algorithm converges to the performance of the controller with access to the true dynamics (red dashed). The total cost is normalized. The initialization episodes (blue) have a high mean cost, but a varying total cost, due to early termination of these episodes.

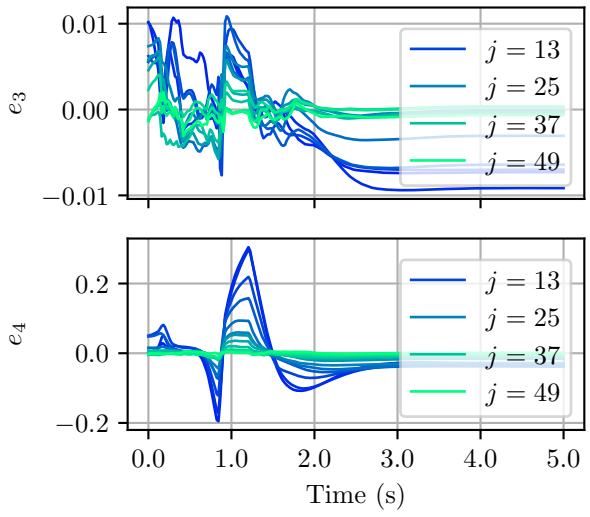

Fig. 3. Approximation error: The values of the neural network model $\hat{\boldsymbol{f}}_{d,\boldsymbol{\theta}}$ converge to the real model $\boldsymbol{f}_d$ with increasing episode number j.

trajectory for q_0, which tranisitions smoothly between several randomly chosen positions. Based on the planned trajectory, a feedforward control for the input $u_1 = \ddot{q}_0$ is derived. After three [5] iterations of Alg. 3 the transition can be accomplished (see Fig. 4 and the respective video at `https://github.com/TUD-RST/pygent`).

[5] After only one iteration, the algorithm drives the system to 0.8, but the poles are not perfectly at rest.

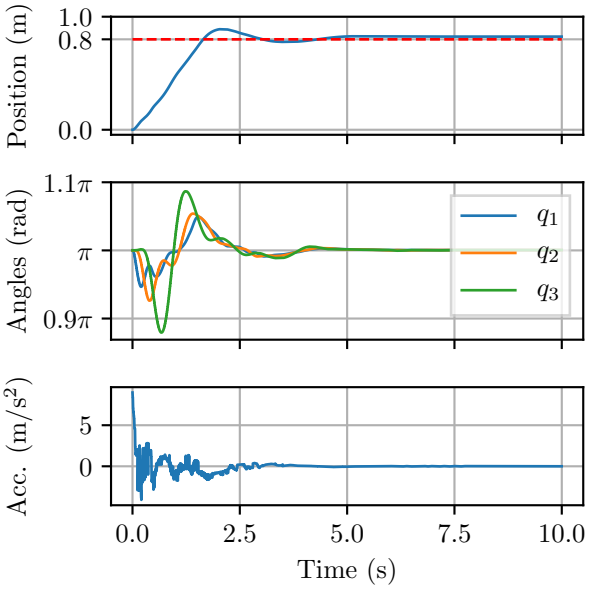

Fig. 4. Cart-triple-pole system transition from $q_0 = 0.0$ to $q_0 = 0.8$ with downward hanging poles.

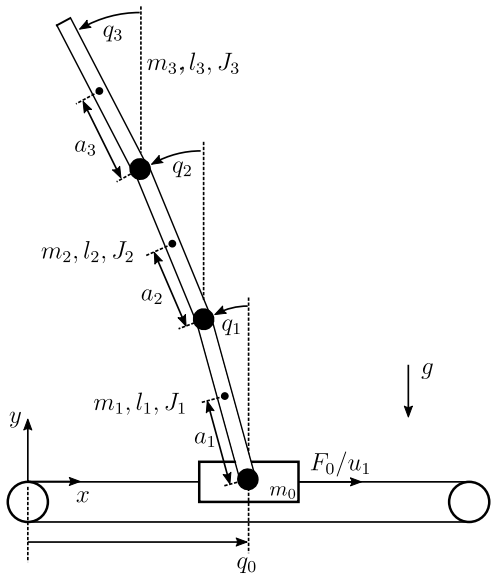

Fig. 5. Cart-triple-pole system.

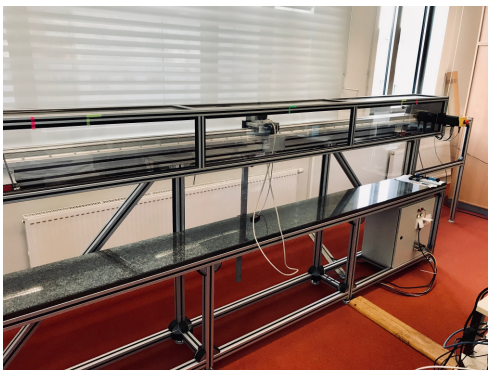

Fig. 6. Cart-triple-pole test bench at our institute.

7. CONCLUSION AND OUTLOOK

In this work we demonstrated, that the combination of neural network dynamics models and offline trajectory planning is suitable for solving certain transition problems on the single and cart-triple-pole systems without prior knowledge of the dynamics through a reinforcement learning scheme. Thereby, the *data efficiency* of the swing-up task of the cart-pole system is significantly higher, compared to model-free methods and matches that of other model-based algorithms (Chua et al., 2018, Fig. 3). One advantage of our approach solving the swing-up task of the cart-pole system over PILCO (Deisenroth and Rasmussen, 2011) and Deep PILCO (Gal et al., 2016) is the lower *computational effort*. The authors in Gal et al. (2016) report 31.05 (PILCO)/8.78 (Deep PILCO) minutes of average training time per episode, where the presented approach only requires 4.13 minutes [6] per episode and scales linear with the data. In constrast to using a GP dynamics model, the applicability of the neural network model is thus not as limited by the size of the data set $\mathcal{D}$.

An open research question is, if Alg. 3 is also suited for learning the swing-up of the cart-triple-pole system. Here, the remaining model error, as pointed out in Sec. 6.1 could be more problematic, due to the strong nonlinearity of the system and its chaotic behaviour. The small sample-time further increases the problem of error propagation in multi-step trajectory planning. To tackle this problem, one approach could be to use a multi-step prediction error instead of (11).

One general drawback with the presented deterministic modeling approach is that *the model does not know what it does not know*. To overcome this problem, the prediction uncertainty could be incorporated by using so called deep ensemble models (Lakshminarayanan et al., 2017) which were used in the RL setting in Chua et al. (2018).

In this work we used iLQR as a planning algorithm, which sometimes got stuck in local minima in the early stages of trajectory planning, when using the neural network model of dynamical systems that were not included in this paper. Therefore, other trajectory planning algorithms like collocation could be compared to iLQR in future works.

From a control theoretic point of view it may seem unnecessary effortful to learn the full dynamics, when a reasonable good model can be found by first-principles and system identification. The cart-triple-pole system however is rather difficult to control and an approach that combines model learning with a physical modeling based prior could be beneficial. Kaheman et al. (2019) and Lee et al. (2017), where only the modeling error is learned, are promising approaches.

ACKNOWLEDGEMENTS

We thank the Center for Information Services and High Performance Computing (ZIH) at TU Dresden for generous allocations of compute resources. We also thank the anonymous reviewers for their valuable comments.

[6] Hardware: 1.3 GHz Dual-Core Intel Core i5 (2013).

REFERENCES

Bechtle, S., Rai, A., Lin, Y., Righetti, L., and Meier, F. (2019). Curious iLQR: Resolving uncertainty in model-based RL. *arXiv:1904.06786*.

Bertsekas, D.P. (2005). *Dynamic Programming and Optimal Control*, volume 1. Athena Sientific, Bellmont, MA, 3rd edition.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, 4754–4765.

Deisenroth, M. and Rasmussen, C.E. (2011). PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 465–472.

Gal, Y., McAllister, R., and Rasmussen, C.E. (2016). Improving pilco with bayesian neural network dynamics models.

Kaheman, K., Kaiser, E., Strom, B., Kutz, J.N., and Brunton, S.L. (2019). Learning discrepancy models from experimental data. *arXiv:1909.08574*.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, 6402–6413.

Lee, G., Srinivasa, S.S., and Mason, M.T. (2017). GP-iLQG: Data-driven robust optimal control for uncertain nonlinear dynamical systems. *arXiv:1705.05344*.

Li, W. and Todorov, E. (2004). Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the 1st International Confenrence on Informatics in Control, Automation and Robotics*, 222–229.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *arXiv: 1509.02971*.

Mayne, D. (1966). A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1), 85–95.

Mitrovic, D., Klanke, S., and Vijayakumar, S. (2008). Optimal control with adaptive internal dynamics models. In *Proceedings of the 5th International Confenrence on Informatics in Control, Automation and Robotics*.

Nagabandi, A., Kahn, G., Fearing, R.S., and Levine, S. (2018). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 7559–7566.

Pritzkoleit, M. (2019). *Bestärkendes Lernen zur Steuerung und Regelung nichtlinearer dynamischer Systeme*. Diploma thesis, TU Dresden, Germany. URL `https://github.com/TUD-RST/pygent`.

Tassa, Y., Mansard, N., and Todorov, E. (2014). Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 1168–1175. IEEE.

Yamaguchi, A. and Atkeson, C.G. (2016). Neural networks and differential dynamic programming for reinforcement learning problems. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 5434–5441.