Application of Deep Reinforcement Learning Control of an Inverted Hydraulic Pendulum

Faras Brumand-Poor*, Lovis Kauderer,
Gunnar Matthiesen and Katharina Schmitz

RWTH Aachen University, Institute for Fluid Power Drives and Systems (IFAS), Campus-Boulevard 30, D-52074 Aachen, Germany
E-mail: faras.brumand@ifas.rwth-aachen.de
**Corresponding Author*

Received 05 October 2022; Accepted 10 February 2023;
Publication 29 April 2023

Abstract

Deep reinforcement learning (RL) control is an emerging branch of machine learning focusing on data-driven solutions to complex nonlinear optimal control problems by trial-and-error learning. This study aims to apply deep reinforcement learning control to a hydromechanical system. The investigated system is an inverted pendulum on a cart with a hydraulic drive. The focus lies on implementing a comprehensive framework for the deep RL controller, which allows for training a control strategy in simulation and solving the tasks of swinging the pendulum up and balancing it. The RL controller can solve these challenges successfully; therefore, reinforcement learning presents a possibility for novel data-driven control approaches for hydromechanical systems.

Keywords: Reinforcement learning control, hydraulic control learning control, inverted hydraulic pendulum.

International Journal of Fluid Power, Vol. 24_2, 393–418.
doi: 10.13052/ijfp1439-9776.2429

1 Introduction

1.1 Motivation

Fluid-powered plants are typically operated with a linear or optimal controller [1]. Linear controllers are tuned for a fixed operating range. Both types of control require a profound system understanding and exact physical models. Especially the modeling of nonlinearities in the hydraulic system due to effects like fluid compressibility poses a challenge in the process of implementing an optimal controller. Furthermore, multi-task control problems often need to be divided into individual subproblems. Moreover, apart from the complex characterization, most optimal control methods lack flexibility regarding the adjustment of the control structure. Therefore, the maintenance of an optimal controller usually yields a high workload.

Reinforcement learning provides a more flexible approach to the problem. Instead of providing a detailed mathematical characterization of the investigated system, the RL controller aims to grasp the system's behavior by trial and error. Reinforcement learning is simultaneously a class of problems and solution methods for these problems [2]. The problems of RL involve learning a behavior, i.e., how to map states to actions, such that a numerical reward signal is maximized. In comparison to the other machine learning branches, RL is highly focused on goal-directed learning from interaction. The challenges RL is applied to are closed-loop since the learning algorithm's actions influence its further inputs. From a control systems perspective, RL is a class of direct adaptive optimal controllers [3] since the control is determined directly by estimating a real-valued function by maximizing an obtained reward and learning by the resulting system's reaction.

An inverted pendulum with a hydraulic drive was chosen as an exemplary system for this study. Swinging up and balancing such an inverted pendulum pole is a classical demonstration example in control theory, due to its simple set-up but intriguing properties of being highly non-linear and noisy [4]. The system consists of a cart with pivoted pole, with the cart being attached to a hydraulic cylinder, which is actuated by a servo valve. The RL control loop is displayed in Figure 1 and shows the RL controller and the investigated system. The controller sets the valve position y_t and obtains, in comparison to a conventional control loop, not an error signal, instead, it is provided with the current states st. Moreover, the plant is computing a reward signal r_t in addition to the next system states $s_{(t+1)}$.

This paper aims to develop a control methodology based on an RL controller, which is trained and eventually validated in a simulation. The focus

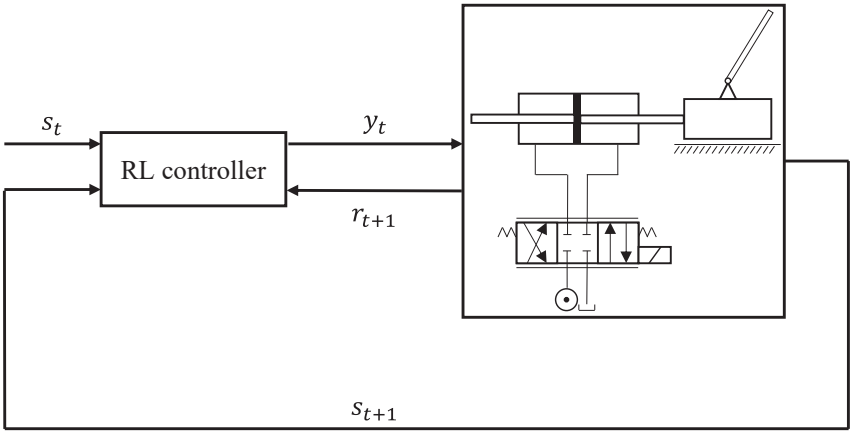

Figure 1 Control loop with RL controller and the investigated system.

of this paper lies in the whole process of developing an RL control for the inverted pendulum. Especially, the reward system of the environment and the parts of the control algorithm are investigated. In the following subsections, the challenges of RL and the state-of-the-art of RL in fluid power systems are introduced. Afterward, the implementation of the control loop is presented, and eventually, the results are discussed.

This paper was originally published in the Proceedings of the 13th International Fluid Power Conference [5] and is now extended with novel implementations and results.

1.2 Challenges

Reinforcement learning faces several challenges. Firstly, in contrast to other machine learning branches the agent needs to be trained with an interactive system instead of just a collected data set. Secondly, the goal is to act such that not only the immediate reward but also the subsequent reward is maximized. Finally, yet importantly, deep reinforcement learning (DRL) contains neural network architectures, thus the algorithm cannot give insight into the discovered understanding of how to behave for improved performance (e.g. DeepMind's AlphaZero chess algorithm [6]). Reinforcement learning has proven its potential as an optimal controller in a series of artificial domains [7–10].

The domain of RL control is relatively new, which results in a lack of rules and regulations for implementation. Especially, finding the optimal several

parameters defined in the RL controller and the reward function represent a challenge in the development of the control algorithm. Furthermore, the optimizer is a crucial part of the controller, since it needs to be stable and able to obtain an optimum quickly.

1.3 Reinforcement Learning Control in Fluid Power

In the following, a short survey of related and promising works in the field of RL control in fluid power, both hydraulics, and pneumatics is given.

Generally, most research has been conducted in simulation environments. Usually, RL algorithms are black box models, therefore an investigation and validation of the RL controller in a simulation is the first step for a successful application to the real world. Egli et al. implemented an RL controller for trajectory tracking of highly nonlinear hydraulic excavator arms with proximal policy optimization (PPO) in simulation [11]. Andersson et al. applied deep RL with PPO to hydraulically actuated crane manipulators of forestry machines in a simulated environment. The results yield successful log grasping in an energy-efficient manner, which was achieved by the energy-optimization goal in the reward function, resulting in smoother motion and acceleration profiles [12]. Karpenko et al. coordinate a pair of hydraulic manipulators with RL aiming to reduce the interaction forces during the positioning of an object along a trajectory [13]. Karpenko and Anderson build up on this multi-agent learning system for the same task and investigate it in both simulation and real-world experiment [14]. Wang et. al extend a PD controller with RL of a simulated biped robot with pneumatic actuators to improve robustness against ground disturbances [15]. Satheeshbabu et al. train a position controller of a pneumatic soft continuum robot arm with deep Q-learning in simulation, and directly transfer it to the real system for validation, with a natural loss of robustness [16].

2 Simulation-Based Training

2.1 The Inverted Hydraulic Pendulum

The inverted pendulum is a benchmark problem in control theory. In this paper, a hydraulic double-rod cylinder is connected to a cart and moves horizontally on a rail. Attached to the cart is a pivoted pole, constituting the inverted pendulum. Contrary to ordinary inverted pendulum plants in automatic control, the cart is not actuated with a direct current step motor, but with a hydraulic cylinder and a servo valve. More precisely, the double-acting

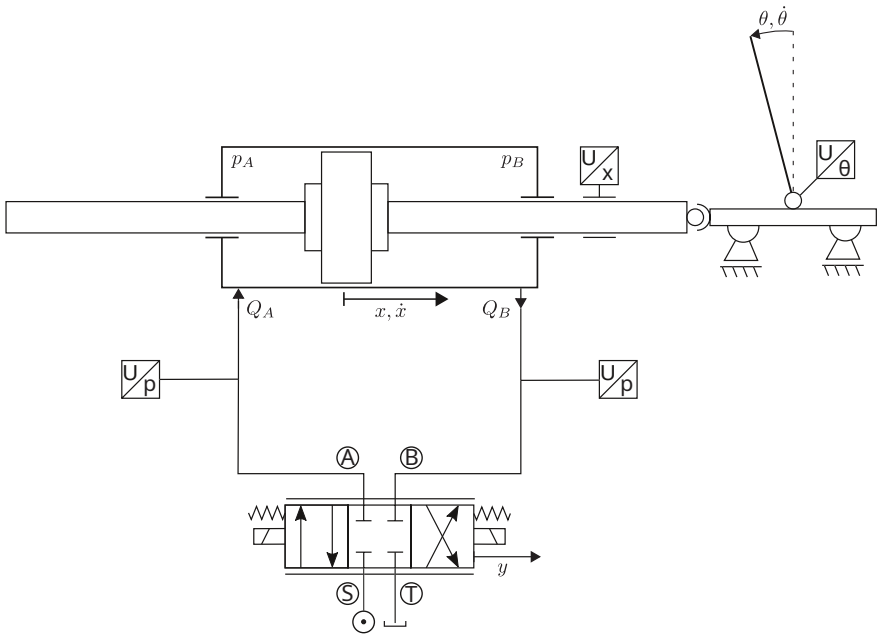

Figure 2 Illustration of the hydraulic-mechanical inverted pendulum.

hydraulic cylinder is steered with a zero overlapped 4/3-way valve. The valve is provided with constant pressure from the source. Figure 2 shows the schematic diagram of the hydraulic-mechanical part that is relevant for controlling. The control task is to swing up and balance the pendulum pole in an upright position by actuating the cylinder position via the valve position y of the servo valve. Thus, the valve position is the RL agent's action that it must infer based on the environmental state in every interaction step.

In order to train the agent in a simulated environment, a mathematical model of the hydraulic-mechanical system in Figure 2 is modeled. The system is simulated in terms of a mathematical white-box model, i.e., a system of algebraic and ordinary differential equations describing the physical relationships. The simulation model is implemented directly in Python by setting up the mathematical equations and solving these numerically with a fourth-order Runge-Kutta algorithm. The RL algorithm, which is introduced in the subsequent subsection, is also developed in Python. This allows for fast training without additional overhead due to communication with an external simulation model. Furthermore, the complete Python program including simulation and controller can be debugged and executed

Table 1 List of the model input and outputs with regard to the RL agent, i.e. what the RL agent actuates (model input or action), based on what it senses (model outputs or environmental state)

	Model Variable	Unit	Description
Input	y	–	relative valve position $[-1, 1]$
Outputs	θ	rad	pole angle
	$\dot{\theta}$	rad/s	angular velocity of the pole
	$\dot{x}$	m/s	cart velocity
	x	m	cart position

quickly on different systems without additional setup requirements, e.g., high-performance computing clusters.

In the following, the constructed mathematical model is explained by means of its parameters, variables, in-, and outputs as well as the hydraulic and mechanical equations. Table 1 summarizes the model's parameters with regard to the RL agent.

The parameter values are determined according to data sheets of the machine components, measurements, and parameter identifications carried out in preliminary works, as well as simple heuristics. For further information, the reader is advised to cover the prior work [5].

First, the hydraulic part is presented. The zero overlapped 4/3-way valve is modeled via orifice formulas for the volume flow into the left cylinder chamber Q_A and the volume flow out of the right cylinder chamber Q_B, depending on the valve position y:

$$
\begin{aligned}
Q_A &= \begin{cases} \operatorname{sign}(p_S - p_A) y \, C_v \sqrt{|p_S - p_A|} & \text{if } y > 0 \\ -\operatorname{sign}(p_A - p_T) y \, C_v \sqrt{|p_A - p_T|} & \text{if } y < 0 \\ 0 & \text{else} \end{cases} \\
Q_B &= \begin{cases} \operatorname{sign}(p_B - p_T) y \, C_v \sqrt{|p_B - p_T|} & \text{if } y > 0 \\ -\operatorname{sign}(p_S - p_B) y \, C_v \sqrt{|p_S - p_B|} & \text{if } y < 0 \\ 0 & \text{else} \end{cases}
\end{aligned}
\tag{1}
$$

Here, p_S and p_T are the source and tank pressures (Ⓢ and Ⓣ in Figure 2) and p_A and p_B are the pressures in the left and right cylinder chambers (Ⓐ and Ⓑ in Figure 2). C_v is the flow coefficient and defined as $C_v = \frac{Q_N}{\sqrt{\Delta p_N}} = A_{o,\max} \alpha_d \sqrt{\frac{2}{\rho}}$ with the volumetric flow rate, Q_N, per control edge at a given

pressure difference $\sqrt{\Delta p_N}$, the maximum valve orifice opening area $A_{o,\max}$, the discharge coefficient α_d and the fluid density ρ. The pressure build-up in the left and right cylinder chambers is modeled with the following ODEs:

$$\begin{aligned} \dot{p}_A &= \frac{K(p_A)}{(V_{A,0} + A_c x)}(Q_A - A_c \dot{x}) \\ \dot{p}_B &= \frac{K(p_B)}{(V_{B,0} - A_c x)}(-Q_B + A_c \dot{x}) \end{aligned} \tag{2}$$

Here, $K(p)$ is the pressure-dependent equivalent bulk modulus for an HLP 46 hydraulic oil [17].

$V_{A,0}$ and $V_{B,0}$ are the initial chamber volumes and A_c is the effective piston area of the cylinder. The pressure difference between both cylinder chambers translates into the pressure force:

$$F_p = (p_A - p_B)A_c \tag{3}$$

which actuates the cylinder piston and the attached mechanical cart and pole system. At the same time, the force of the attached mechanical system F_{link} acts against the cylinder piston. Thus it *links* the hydraulic and mechanical systems. Furthermore, a friction force F_r acts against the cylinder piston. The equation used to model this friction force combines Stribeck, Coulomb, and viscous components and is given by [18]:

$$\begin{aligned} F_r = C_{\text{visc}}\dot{x} + \sqrt{2\exp(1)}(F_{\text{brkwy}} - F_{\text{col}}) \left(\frac{\dot{x}}{\sqrt{2}v_{\text{brkwy}}} \right) \\ \exp\left(-\left(\frac{\dot{x}}{\sqrt{2}v_{\text{brkwy}}} \right)^2 \right) + F_{\text{col}} \tanh\left(\frac{\dot{x}}{\frac{v_{\text{brkwy}}}{10}} \right) \end{aligned} \tag{4}$$

where $\dot{x}$ is the piston (and cart) velocity and C_{visc}, F_{brkwy}, F_{col} and v_{brkwy} are the viscous friction coefficient, the breakaway friction force, the Coulomb friction force and the breakaway friction force, respectively.

With the pressure force F_p, connecting force F_{link} and friction force F_r, the equation of motion for the piston is given by:

$$m_{\text{piston}}\ddot{x} = F_p - F_{\text{link}} - F_r, \tag{5}$$

where $\ddot{x}$ is the acceleration of the piston (and cart) and m_{piston} is the piston mass. From this equation, together with the equation of motion for the cart position of the mechanical cart and pole system, the connecting force F_{link} can be deduced, as shown below.

The following equations of the mechanical part are gathered from Green [19] and adjusted to the present needs. The acceleration of the cart is given by:

$$\begin{aligned}\ddot{x} = & \frac{1}{(m_{\text{pole}}\hat{l}\cos\theta)^2 - (m_{\text{pole}}\hat{l}^2 + J)(m_{\text{cart}} + m_{\text{pole}})} \\ & \left[\left(m_{\text{pole}}\hat{l}\right)^2 g \sin\theta\cos\theta - \left(m_{\text{pole}}\hat{l}^2 + J\right)\right. \\ & \left.\left(F_{\text{link}} + m_{\text{pole}}\hat{l}\dot{\theta}^2\sin\theta\right) - m_{\text{pole}}\hat{l}M_f\cos\theta\right]\end{aligned} \tag{6}$$

where g is the gravitational acceleration, m_{pole} and m_{cart} are the pole and cart mass respectively, $\hat{l}$ is the half pole length, J is the moment of inertia of the pole, M_f is the moment of friction of the joint, and θ and $\dot{\theta}$ are the pole angle and its angular velocity. The moment of friction consists of two parts to model the joint friction of the pivoted pole precisely:

$$M_f = \mu_p\dot{\theta} + \tilde{\mu}_p\frac{m_{\text{pole}}g}{2}\operatorname{sign}(\dot{\theta}) \tag{7}$$

If viewed individually, the connecting force F_{link} in Equation (6) and Equation (5) is still unknown, but by plugging Equation (6) into Equation (5) and rearranging the equation, the connecting force F_{link} is given by:

$$\begin{aligned}F_{\text{link}} = & \frac{1}{(m_{\text{pole}}\hat{l}^2 + J)(m_{\text{piston}} + m_{\text{cart}} + m_{\text{pole}}) - (m_{\text{pole}}\hat{l}\cos\theta)^2} \\ & \cdot\left(m_{\text{piston}}\left[\left(m_{\text{pole}}\hat{l}\right)^2 g\sin\theta\cos\theta - \left(m_{\text{pole}}\hat{l}^2 + J\right)\right.\right. \\ & \left.\cdot\, m_{\text{pole}}\hat{l}\dot{\theta}^2\sin\theta - m_{\text{pole}}\hat{l}M_f\cos\theta\right] + \left[\left(m_{\text{pole}}\hat{l}\cos\theta\right)^2\right. \\ & \left.\left.- \left(m_{\text{pole}}\hat{l}^2 + J\right)\left(m_{\text{cart}} + m_{\text{pole}}\right)\right]\right](F_r - F_p)\Bigg)\end{aligned} \tag{8}$$

Knowing F_{link}, the equation of motion for the angular acceleration of the pole can be evaluated as:

$$\begin{aligned}\ddot{\theta} = & \frac{1}{(m_{\text{cart}} + m_{\text{pole}})(m_{\text{pole}}\hat{l}^2 + J) - (m_{\text{pole}}\hat{l}\cos\theta)^2} \\ & \cdot \Big(\left(m_{\text{cart}} + m_{\text{pole}}\right) m_{\text{pole}} g \hat{l} \sin\theta - \left(m_{\text{pole}}\hat{l}\cos\theta\right) \\ & \cdot \left(F_{\text{link}} + m_{\text{pole}}\hat{l}\dot{\theta}^2 \sin\theta\right) - \left(m_{\text{cart}} + m_{\text{pole}}\right) M_f \Big)\end{aligned} \tag{9}$$

With F_{link} and $\ddot{\theta}$, the equation of motion for the acceleration of the cart can be evaluated as

$$\ddot{x} = \frac{F_{\text{link}} - m_{\text{pole}}\hat{l}\ddot{\theta}\cos\theta + m_{\text{pole}}\hat{l}\dot{\theta}^2\sin\theta}{m_{\text{cart}} + m_{\text{pole}}} \tag{10}$$

The result is a set of equations in state-space representation, where the derivative of the state vector is defined as

$$\dot{\boldsymbol{s}}(t) = \begin{bmatrix} \dot{p}_A(t) & \dot{p}_B(t) & \dot{x}(t) & \ddot{x}(t) & \dot{\theta}(t) & \ddot{\theta}(t) \end{bmatrix}^T$$

In order to obtain $\boldsymbol{s}(t)$, the system is integrated with a classical, 4th order, Runge–Kutta method.

2.2 The Control Agent

2.2.1 The reinforcement learning framework – state, action and reward

In reinforcement learning an agent aims to learn a behavior, or policy, such that the decisions made in a dynamic environment are maximizing a collectible reward signal provided by the environment. Mathematically, the agent remembers which actions result in a high reward in the past and further executes these actions with a higher probability. Due to the game-like approach of RL, these methods are much closer to human learning than the other algorithms of machine learning. The performance of an agent is measured for one episode, with an episode being a predetermined amount of time steps or interactions N. Furthermore, the environment can have constraints, which when violated by the agent, terminate an episode. The initial starting point of the RL agent in an episode is s_0 and in every discrete time step t, the agent resides in an environmental state s_t out of the set of

possible states S. Receiving an observation of the current state s_t, the agent chooses an action $a_t \in A$ either from a discrete and finite action set or from a set of continuous-valued actions. After taking the action a_t, the agent receives a scalar reward signal $r_t \in R$, which determines the action the agent took has been good (positive reward value) or bad (negative reward value i.e. punishment), while the absolute value of r_t indicates to what extent the chosen action was good or bad. The interaction of the environment of the agent can be represented by the tuple $(s_t, a_t, r_t, s_{(t+1)})$, with $t \in (0, N-1)$.

2.2.2 Actor and critic network architectures

The mathematical foundation of RL is often represented in the form of Markov decision processes (MDP) [2]. MDPs model the dynamics between the states of a finite state space. The recent state traverses to another state or back to itself, depending on the chosen action. Everything that is beyond the selection of an action is modeled as a part of the environment in an MDP. The agent maneuvers in a dynamic environment; therefore, a selected action in a particular state does not only affect the immediate reward but also the subsequent states and eventually, the following rewards received. Therefore, strictly learning from the immediate reward, it is unlikely that the agent maximizes its total reward in the end. The return function R_t summarizes the future rewards when progressing from a certain state s_t. The discount factor $\gamma \in (0, 1)$ is tuned to set the importance of later rewards, resulting in the following equation of the return function:

$$R_t = r_t + \gamma\, r_{(t+1)} + \gamma^2\, r_{(t+2)} + \cdots + \gamma^n\, r_{(t+n)} = \sum \gamma^i\, r_{(t+i)} \tag{11}$$

A greater value for γ results in more value for rewards lying further in the future. The policy of the agent $\pi(s)$ provides a probability for every possible action with a_t being sampled according to:

$$a \sim \pi(s) \tag{12}$$

The policy is determined by the value function $V_\pi(s)$. The value function provides a value of a state, which is defined as the expected future return when starting from the state following a certain policy.

$$V_\pi(s) = E_\pi[R_t|\pi, s_t = s] = E_\pi\left[\sum_{i=0}^{n} \gamma^i r_{(t+1)}|\pi, s_t = s\right] \;\forall\, s \in S \tag{13}$$

Reinforcement learning faces two fundamental problems. Firstly, the prediction problem, which aims to evaluate the value function for a given

policy, and secondly the control problem, which deals with finding an optimal policy for the given environment and the current estimation of its value function. The differences in solving the prediction problem result in the approach to estimating the value function. Regarding the control problem, a big challenge is called the exploration-exploitation problem. The agent always aims to behave in such a way that the obtained return is maximal. However, the agent needs advisement to choose an unknown path to get a better understanding of the environment and to find other ways of gaining rewards. Furthermore, to adapt to changing environmental conditions, an exploring agent is a requirement, since deviations in the environment can cause alternative actions to become the best choice in a certain state. However, too much exploration on the other hand can cause the agent to collect low rewards as the trained knowledge of rewards in the environment is never used, and states far away from the initial state are rarely reached causing a lack of exploration in these situations. In this paper, the actor-critic algorithm is implemented, because it deals with the prediction and control problem separately, thus poor performance can be tracked back and assigned to one of them. The method consists of two parts. Firstly, the actor selects and executes actions in the environment. Thus, the actor represents the policy or agent of this algorithm. Secondly, the critic estimates the value function of the current state, in which the actor resides. The actor uses the critic as a baseline for its suggested actions, i.e. the policy of the actor is tuned to the relative returns of taking a certain action in a state, instead of tuning in response to the total returns. For example, a certain action results in a negative reward of -50, however, the critic estimated the state to have a value of -80, thus the relative return is $+30$. The critic tunes its parameters depending on the error of the estimation of the state value. In this paper, a one-step temporal difference error is used, which estimates the value of a state by calculating the mean of different one-step returns obtained from this particular state. The complete control loop is displayed in Figure 3.

The actor $\pi(s)$ determines the valve position in percentage depending on the current state s_t. The state-space consists of a tuplet $(x, \dot{x}, \theta, \dot{\theta})$, with the normalized cart position x, the angle of the pendulum θ, which is provided as $\cos(\theta)$ and $\sin(\theta)$, the velocity of cart $\dot{x}$ and angle $\dot{\theta}$. The critic estimates the state value $V_\pi(s)$ based on the current state and passes it to the actor as a baseline for its parameter tuning. Both models receive the reward, which contributes to the loss functions of the model's parameters.

Artificial neural networks approximate the actor and critic. Figure 4 shows the implemented network architectures.

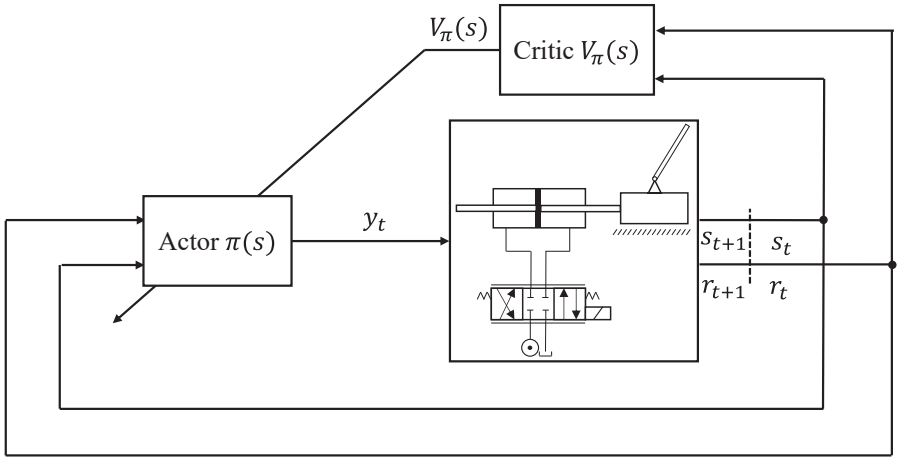

Figure 3 Control loop with actor and critic.

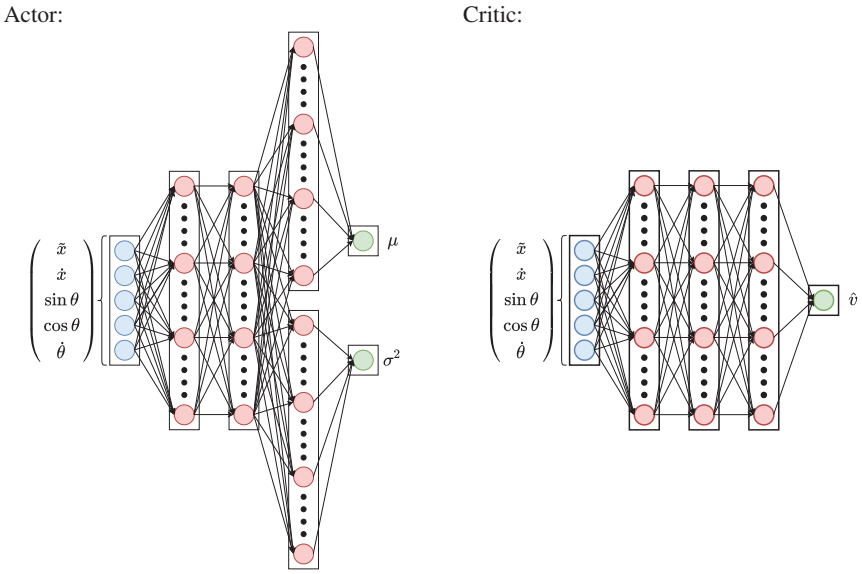

Figure 4 The neural network architectures of the RL agent's actor and critic, with a five dimensional state observation vector as inputs.

In a heuristic approach, the best configuration was the actor consisting of four layers, with the last two outputting the mean μ and the standard deviation σ. These create a probability distribution, which is sampled to generate the current action y_t. The critic consists of three hidden layers following the input

layer. The size of each layer of the actor and critic is 64, roughly following the results of Henderson et al. regarding what worked best for the same algorithm in their study [20]. The hyperbolic tangents (tanh) is used as an activation function for all layers except the actor's output layer for the variance σ^2 and the critic's output layer. The tanh activation outputs values in the range $[-1, 1]$. Since the variance is non-negative ($\sigma^2 \geq 0$), it cannot be used in this case. Instead, the softplus activation with range $(0, \infty)$ is a reasonable choice. In the case of the critic's output layer, no activation function is present because the state values are not supposed to be bound.

2.2.3 Proximal policy optimization

The neural network parameter optimization is done with the Proximal Policy Optimization [21]. PPO is a stochastic policy gradient (PG) method, meaning that a stochastic policy, $\pi_{\boldsymbol{\theta}}$, e.g. a neural network, is parametrized based on an estimated gradient of a performance objective function, the loss, w.r.t to the policy parameters [2]. PPO is an actor-critic-style policy optimization algorithm that aims to tackle some crucial challenges inherent in previous policy gradient methods [21]. It is more sample efficient, less sensitive to hyperparameter choices, and thus easier to tune than default PG methods [22]. The actor approximates the policy and its parameters are tuned to maximize an expected reward sum. It takes observed states of the environment as input and suggests actions to take as an output [22]. The parameters of a network, either actor or critic, are updated by the gradient of a specific loss, with the network's parameters $\boldsymbol{\theta} \in \mathbb{R}^{d'}$, where d' is the dimension of the network's weight matrix.

Each part of the whole loss is presented below, starting with the actor loss:

$$L^{CLIP}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\boldsymbol{\theta}) \hat{A}_t, \operatorname{clip} \left(r_t(\boldsymbol{\theta}), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right] \tag{14}$$

with ϵ being a hyperparameter to control how much the new policy is allowed to differ from the old. It is usually set to $\epsilon = 0.2$. The *clip*-function clips the value of the reward $r_t(\boldsymbol{\theta})$ to the interval $[1 - \epsilon, 1 + \epsilon]$. $\hat{A}_t$ is the so-called *advantage estimate* at time step t [21]. The *advantage estimate* indicates how much better or worse the selected action is compared to a randomly selected action [22,23]. It is computed by calculating the difference between the actual return G_t and a so-called *baseline* $b(S_t)$:

$$\hat{A}_t = G_t - b(S_t). \tag{15}$$

Usually, an estimate of the state value $\hat{v}_w(S_t)$ is chosen as the baseline, which is learned by the critic. Equation 14 unites following purposes [22]:

- It pushes the policy towards actions that yield a high positive advantage over the baseline.
- It prevents too large gradient updates, which otherwise might ruin the policy, in case of positive advantage, or cause the action probabilities to become zero, in case of negative advantage.
- It reverts an update which makes an action with negative advantage a lot more likely.

Regarding the critic loss, the network is updated based on the value function (VF) in a classical supervised learning fishing, by minimizing the following mean squared error loss with gradient descent [23]:

$$L^{VF}(\boldsymbol{w}) = \hat{\mathbb{E}}_t[(G_t - \hat{v}_{\boldsymbol{w}}(S_t))^2] \tag{16}$$

The critic estimates the baseline for the advantage calculation, therefore it qualifies the actor's choices: If the actual return is higher than the critic's estimated return, the advantage is positive, and vice versa. The advantage, therefore, indicates the relative value of an action in a particular state and *reinforces* this action according to its value.

The final loss function of PPO consists of the clipped objective L^{CLIP} for the policy network (actor), the MSE objective L^{VF} for the value function network (critic) and a third term, namely the entropy $S[\pi_{\boldsymbol{\theta}}](s)$:

$$L^{PPO}(\boldsymbol{\theta}, \boldsymbol{w}) = \hat{E}_t[L^{CLIP}(\boldsymbol{\theta}) - c_1 L^{VF}(\boldsymbol{w}) + c_2 S[\pi_{\boldsymbol{\theta}}](s)] \tag{17}$$

Entropy is a measure of how unpredictable the actions of the policy are. It ensures sufficient exploration during the training process by making the policy behave more randomly, until the other terms of the objective start dominating. c_1 and c_2 are hyperparameters to weight the terms and are determined in a heuristic approach to be 0.5 and 0.01, respectively.

2.3 The Control Loop

The complete control with deep RL controller and simulation model is illustrated in Figure 5. Since both the RL agent and the simulation are implemented in Python, there is no need for communication between the RL controller and the environment, allowing for more efficient training. Furthermore, the framework can be deployed on a high-performance cluster

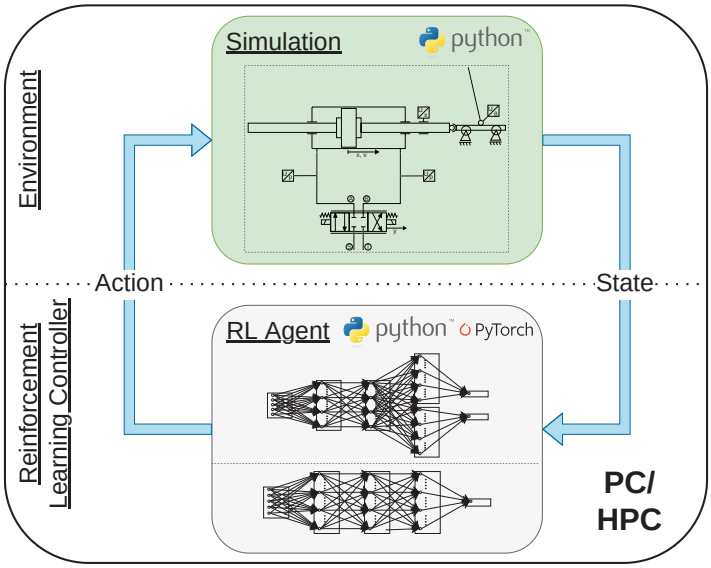

Figure 5 For the simulation-based training, the Python-based RL agent interacts with the Python-based simulated environment internally.

in order to decrease the necessary time for the parametrization of the neural networks.

The communication between RL agent and simulation is instant, however, in a real-world application, the sending, receiving of data, and calculation of the next valve position consume a significant amount of time. It is therefore crucial to include the communication delay in the training model. In order to achieve this the communication process, which is illustrated in Figure 6, is implemented. The interaction process begins with the RL agent sending an initial action a_0 to the plant controller. As soon as the plant controller receives the initial action a_0, it starts to pass on the latest received action to the valve in a fixed execution interval of length t_{sample}. Within this execution interval of length t_{sample}, the RL agent has to receive the current state, e.g. the initial state s_0, determine an action from it, and send this new action a_1 to the simulation model. Compared to instant communication, actions are now applied for more than one time step.

Algorithm 1 shows the procedure of the interaction process. The Environment refers to the simulation model. Apart from that, the interaction always starts in line 2 with receiving a state from, and afterwards sending an action to the environment. It is important to note that the returned state

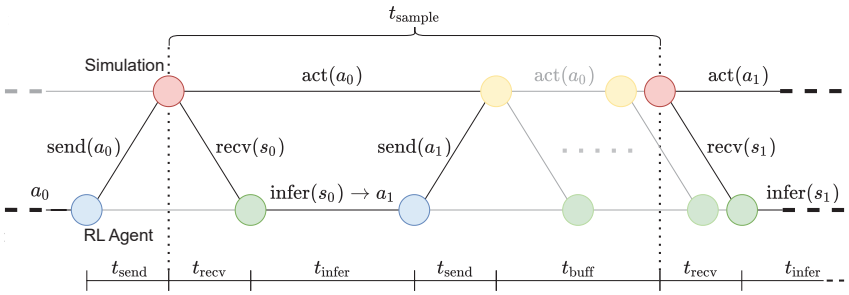

Figure 6 The communication between the RL agent and the simulation.

Algorithm 1 The interaction process of the actor-critic style PPO algorithm during training

Require: initialized agent and environment, initial action $a = 0$, episode counter $N_{\text{ep}} = 0$.

1: **while** $N_{\text{ep}} < N_{\text{ep,max}}$ **do** ▷ every iteration t_{sample} (2 ms) elapse
2: $s \leftarrow actInEnvironment(a)$
3: $r \leftarrow receiveReward(s, a)$
4: $a, \log \pi_\theta(a \mid s) \leftarrow inferAction(s)$ ▷ agent infers action based on current policy
5: $agent \leftarrow collectExperience(s, a, r, \log \pi_\theta(a \mid s))$
6: **if** enough experience collected **then** ▷ experience worth of e.g. 80 s
7: $agent \leftarrow updatePolicy$ ▷ the agent learns from the collected experience
8: $agent \leftarrow deleteExperience$ ▷ the collected experience is discarded
9: **end if**
10: **if** episode finished **then** ▷ after 20 s of if constraints violated
11: $Environment \leftarrow reset$
12: $N_{\text{ep}} \leftarrow N_{\text{ep}} + 1$
13: **end if**
14: **end while**

is not the consequence of the currently sent action, but instead of the prior action. Accordingly, the remaining commands of the current loop iteration have to take place within a specified time t_{sample}, in which the environment executes the action that it just received. A reward is calculated in line 3 based on the current state and action, according to the reward model described in subsection 2.4. Next, a new action is inferred, together with its log probability, by randomly sampling from the probability distribution that is constructed with the mean and variance that were output by the actor network based on the current state. In line 5, which is the last step in a normal interaction, the agent collects the state, action, reward, and log probability in a memory buffer for future usage, i.e. to update its policy and thus learns. A policy update according to algorithm 2 is executed in line 7, as soon as enough experience

is collected in terms of a hyperparameter called update interval I_{update}, which is determined to be 80.

This parameter specifies the number of interactions the agent uses to improve its policy. After the policy update, which is examined in more detail below, the collected experience is discarded by emptying the memory buffer in line 8. The current episode finishes either in terms of the number of interactions reaching the maximum episode length $L_{\text{ep,max}}$, or if constraints are violated, e.g. when moving the piston too close to the end position. In this case, the Environment is reset in line 11 so that the next episode can start from a defined initial position. The whole interaction sequence from line 2 to line 13 is repeated $N_{\text{ep,max}}$ many times, controlled by the while loop in line 1. When training the agent, the maximum number of episodes is usually set to $N_{\text{ep,max}} = \infty$. The interaction process is then manually stopped depending on the learning progress.

In the following, the policy update function called in line 7 of Algorithm 1 is regarded in more detail. As shown in Algorithm 2, it requires the vectors of collected experience from the preceding interactions. In line 2 of Algorithm 2, the discounted returns are estimated in Monte Carlo fashion based on the collected rewards. In line 3, the returns are normalized to have zero mean and standard deviation of 1, in order to improve stability during the gradient updates. In the subsequent *for loop* starting in line 4, the actor's and critic's network weights are repeatedly adjusted in line 10 according to the PPO loss

Algorithm 2 The PPO update procedure

Require: Collected experience vectors S, A, R and $\log \pi_{\theta_{\text{old}}}(A \mid S)$.

1: **function** UPDATEPOLICY
2: $G \leftarrow$ CALCULATERETURNS(R) ▷ Monte Carlo estimates of discounted returns
3: $G \leftarrow$ NORMALIZERETURNS(G) ▷ normalize to zero mean and standard deviation of 1
4: **for** $k = 0, \ldots, N_{\text{epochs}}$ **do**
5: $\log \pi_\theta(A \mid S), \text{Entropy}, \hat{V} \leftarrow$ EVALUATEPOLICY(S, A)
6: $\hat{A} \leftarrow$ COMPUTEADVANTAGES$(G, \hat{V})$ ▷ see Eq. 15
7: $L^{CLIP}(\theta) \leftarrow$ COMPUTEACTORLOSS$(\log \pi_\theta(A \mid S), \log \pi_{\theta_{\text{old}}}(A \mid S), \hat{A})$ ▷ see Eq. 14
8: $L^{VF}(\theta) \leftarrow$ COMPUTECRITICLOSS$(G, \hat{V})$ ▷ see Eq. 16
9: $L^{PPO}(\theta) \leftarrow$ COMPUTEAGENTLOSS$(L^{CLIP}, L^{VF}, \text{Entropy})$ ▷ see Eq. 17
10: Agent $\leftarrow$ ADJUSTWEIGHTS ▷ optimize actor and critic
11: **end for**
12: **end function**

constructed prior to this, in line 9. The function EVALUATEPOLICY(S, A) in line 5 basically does the same as INFERACTION(s) in line 4 of Algorithm 1, but instead the log probabilities are computed according to the probability distribution constructed from the outputs of the iteratively updating actor network with the batch of experienced states as inputs. Moreover, the probability distribution's entropy is computed and the estimated state values $\hat{V}$ are inferred from the critic network. The other parts of the PPO loss are computed in lines 6 to 8 according to the equations given in the respective comments.

2.4 Reward Shaping

The initial reward model is explained in prior work [5]. In this work the reward function is modified by punishing the valve position difference from the current and prior time step, instead of punishing the magnitude of the valve position. The reward provides the RL agent with the necessary information on what is good and bad in terms of fulfilling a task. In the case of swinging up an inverted pendulum, reaching a zero-degree angle of the pole is good and has to be rewarded, whereas driving the cylinder piston into the end positions is bad and has to be punished. However, a reward function should not show the agent how to fulfill a task. Otherwise, the agent is not properly learning by exploring the environment but is guided to the right decisions. This objects to the idea of a self-learning RL controller and perhaps reduces robustness and the possibility of the agent finding unexpected but better solutions. On the other hand, giving too little information may render learning progress impossible. If the obtained reward would only be positive at an upright position and zero for the remaining positions of the pole, the agent starting with the pole upside down would have no chance to learn what to do, as randomly swinging up to find out about the high reward at the top is highly unlikely. Therefore, the difficulty of designing a reward function is giving just enough information to allow for efficient training. The following reward function inspired by [24] is set up for training the RL agent to swing up the inverted pendulum:

$$r(t) = \begin{cases} \cos\theta - c_\omega\,\dot{\theta}^2 - c_y\,(y - y_{\text{prev}})^2 & \text{if } |x| \leq x_{\text{bound}} \\ \cos\theta - c_\omega\,\dot{\theta}^2 - c_y\,(y - y_{\text{prev}})^2 - c_x & \text{else} \end{cases} \tag{18}$$

The four terms of the reward function are displayed in Figure 7. If the pole dangles upside down, the pole angle reward will be around -1, whereas a balanced upright pole will give a reward of up to 1. The remaining terms are punishments to improve the agent's performance: High angular velocities,

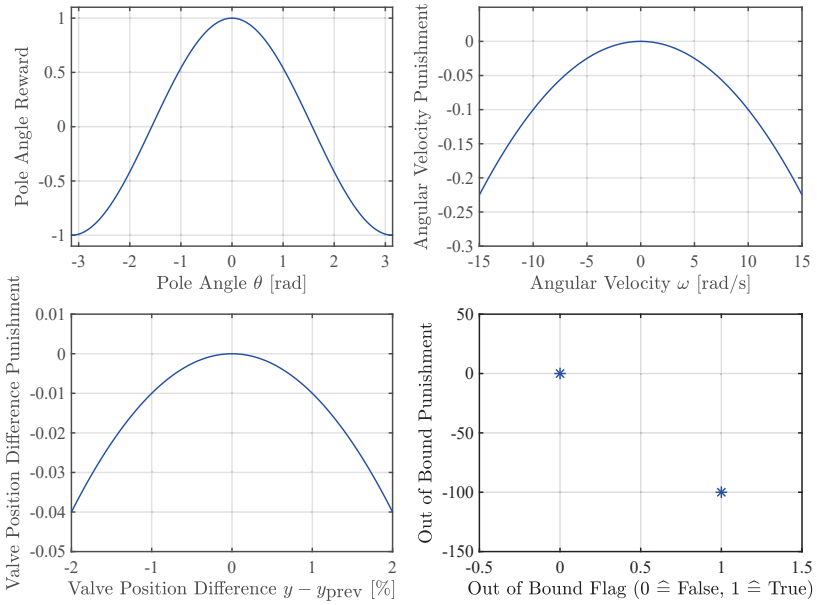

Figure 7 Illustration of the reward function terms.

big valve position differences, and a high punishment for moving the cylinder position too far toward the end position.

3 Training & Results

3.1 Rules and Conditions of the Training

The goal of the agent is to obtain the highest reward, which results in the pole being swung up quickly and balanced in a vertical position with as little valve actuation as possible. The cart starts in the middle of the slide with a hanging pole. In addition to the reward and the state space, the environment has certain termination conditions, which once met, immediately stop the episode. These conditions are application-specific and are listed below:

- An episode is played for more than 20 s.
- The cart position is over ± 0.2 m.

The second condition results from the out-of-bound punishment of the reward function. Because of safety concerns with regard to the application of the trained agent to the real plant, the valve position is set to a maximum value of $|y_{\text{max}}| = 20\,\%$ as a precaution. Accordingly, the agent's actions are

scaled and clipped to $[-20\,\%, 20\,\%]$. The action sampling time t_{sample} is set to 2 ms, to account for the communication delay. The 20 s time duration for one episode is selected by a heuristic approach. The simulation time needs to have a sufficient length in order for the actor to be able to solve the task. Furthermore, the longer an episode, the more information is collected in the episode, which is used for the subsequent optimization and eventually results in a shorter training time.

3.2 Training & Testing

The performance of the agent is validated during the training and eventually during testing. The training performance is investigated in Figure 8 by means of the running reward over the episodes. The running reward stagnates at around 740 after roughly 3000 episodes. The best-performing agent in terms of mean reward and swing-up speed is identified in episode 3929 after 892 optimization runs. This corresponds to roughly 19.82 h of interaction with the simulation.

Figure 9 shows the test results of the agent. It can be seen that the inferred actions or valve positions shown in the upper left plot successfully swing up the pole from its initial angle of $180° \approx 3.14\,\text{rad}$ to the upright position of $360° \approx 6.28\,\text{rad}$ in around $5\,\text{s}$, while the cart position remains within the allowed bounds of $x_{\text{bound}} = 0.2\,\text{m}$ to either side. For the swing-up, the actor infers the allowed valve opening of 20%. However, after successfully swinging the pole up, the actuation remains in a narrow band switching the

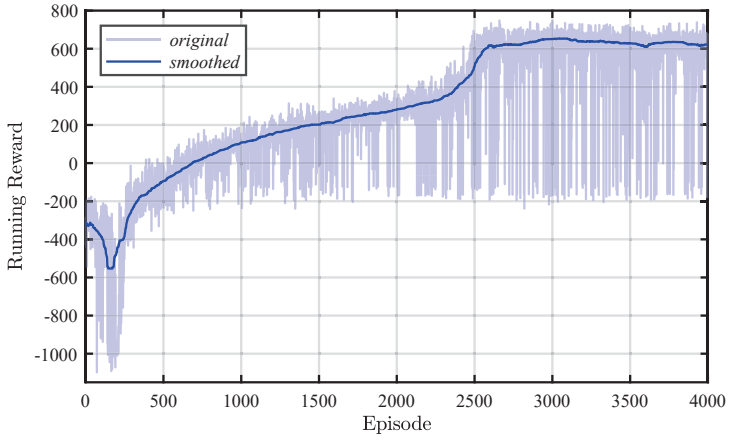

Figure 8 Illustration of the reward function terms.

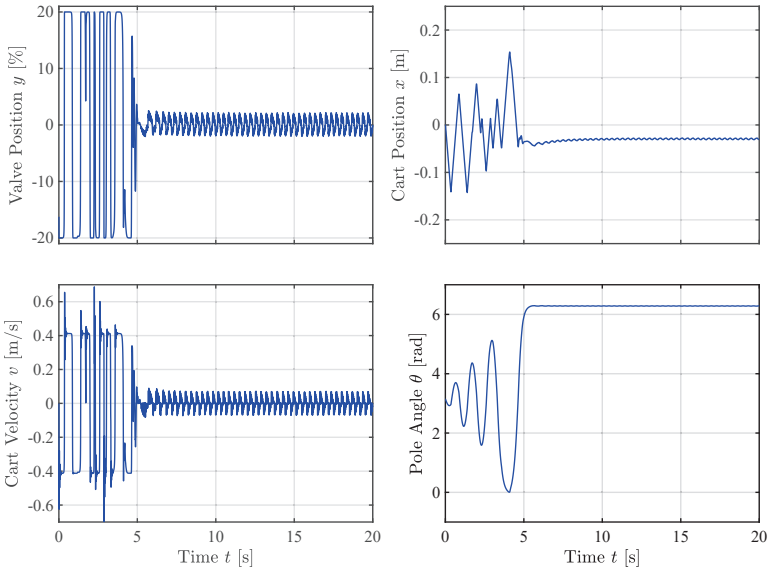

Figure 9 Illustration of the reward function terms.

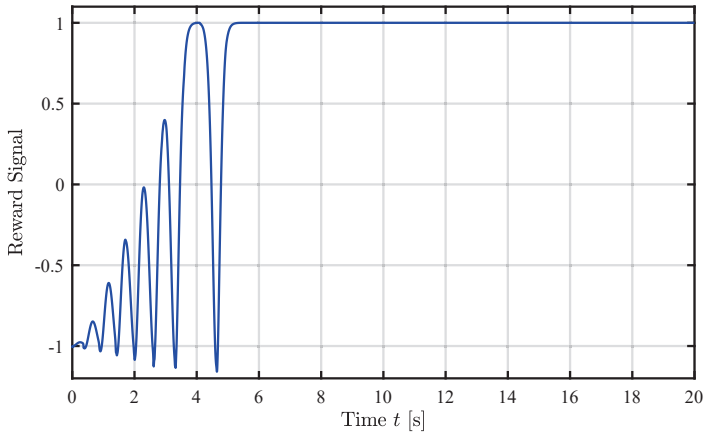

Figure 10 Illustration of the reward function terms.

valve opening direction. The agent does not aim to move the cart back to the middle of the slide since this behavior is not considered in the reward function. Figure 10 shows the reward signal during the course of the test application. Once the pendulum is in the upright balancing position, the reward signal remains almost 1, showing that the agent successfully learned

how to obtain the maximum reward and is able to keep the pole in a stable position.

4 Summary & Conclusion

RL presents a class of novel control algorithms for hydromechanical systems, which have a more flexible and adaptive structure compared to conventional controllers. In this paper, the possibilities offered by RL control of a hydromechanical system were demonstrated by solving the inverted pendulum-on-a-cart problem. The control algorithm was implemented based on the actor-critic approach and the sample efficient Proximal Policy Optimization. The controller was tested on a simulation model, which has a hydraulic drive and a mechanical part, consisting of a cart and pendulum. The agent was able to solve the inverted pendulum challenge. Compared to conventional control algorithms, which normally require two controllers for the swing up and pole balancing respectively, the same actor was able to solve both tasks.

Moreover, a flexible structure is implemented, which can be adjusted to changes in the model through further training. Further research can be conducted on the robustness of the agent, as disturbance and noise can be added to the environment. Eventually, the trained agent will be validated on a real-life test rig, which is available at the institute.

References

[1] J. Mohieddine, Hydraulic Servo-systems: modeling, identification, and control, In: Springer-Verlag London Limited, doi: 10.1007/978-1-4471-0099-7, 2003.

[2] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, In: The MIT Press, Second edition, 2018.

[3] R. Sutton, A. Barto, R. Williams, Reinforcement Learning is Direct Adaptive Optimal Control, In: IEEE Control Systems Magazine, vol. 12, no. 2, pp. 19–22, doi: 10.1109/37.126844, 1992.

[4] S. Adam, L. Busoniu, and R. Babuska, "Experience replay for real-time reinforcement learning control," Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, vol. 42, pp. 201–212, 2012.

[5] F. Brumand-Poor, Control of a Hydromechanical Pendulum with a Reinforcement Learning Agent, In: Proceedings of the 13th International Fluid Power Conference, Aachen, 2022.

[6] D. Bertsekas, Reinforcement Learning and Optimal Control, In: MIT, February 2019.

[7] S. Gottschalk, Differences and similarities between reinforcement learning and the classical optimal control framework, in PAMM. 19. doi: 10.1002/pamm.201900390, 2019.

[8] V. Mnih, Human-level control through deep reinforcement learning, Nature 518, 529–533, https://doi.org/10.1038/nature14236, 2015.

[9] D. Silver, Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484, 2016.

[10] D. Hafner, Learning latent dynamics for planning from pixels. arXiv preprint arXiv:1811.04551, 2018.

[11] P. Egli, A general approach for the automation of hydraulic excavator arms using reinforcement learning, IEEE Robotics and Automation Letters, 2021.

[12] J. Andersson, Reinforcement learning control of a forestry crane manipulator, arXiv, 2021.

[13] M. Karpenko, Coordination of hydraulic manipulators by reinforcement learning, in 2006 American Control Conference, pp. 6, 2006.

[14] M. Karpenko, Decentralized Coordinated Motion Control of Two Hydraulic Actuators Handling a Common Object, Journal of Dynamic Systems, Measurement, and Control, vol. 129, no. 5, pp. 729–741, 2007.

[15] S. Wang, Reinforcement learning control for biped robot walking on uneven surfaces, in The 2006 IEEE International Joint Conference on Neural Network Proceedings, pp. 4173–4178, 2006.

[16] S. Satheeshbabu, Open loop position control of soft continuum arm using deep reinforcement learning, in 2019 International Conference on Robotics and Automation (ICRA), pp. 5133–5139, 2019.

[17] K. Schmitz, and H. Murrenhoff,, Grundlagen der Fluidtechnik, Shaker Verlag, ISBN: 978-3-8440-6246-5, 2018

[18] Translational friction: Friction in contact between moving bodies. [Online]. Available: https://de.mathworks.com/help/simscape/ref/translationalfriction.html, 2022.

[19] C. D. Green. Equations of motion for the cart and pole control task. [Online]. Available: https://sharpneat.sourceforge.io/research/cart-pole/cart-pole-equations.html, 2020.

[20] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," arXiv, 2017.
[21] J. Schulman et al., "Proximal Policy Optimization Algorithms", in CoRR, arXiv: 1707.06347, 2017.
[22] X. Steenbrugge. An introduction to policy gradient methods - deep reinforcement learning. [Online]. Available: https://www.youtube.com/watch?v=5P7I-xPq8u8, 2018.
[23] J. Achiam, "Spinning up in deep reinforcement learning," GitHub repository, 2018.
[24] S. (https://stackoverflow.com/users/754136/simon), "How to choose the reward function for the cart-pole inverted pendulum task,", RL: https://stackoverflow.com/revisions/51684009/2, 2018.

Biographies

Faras Brumand-Poor received a bachelor's degree in electrical engineering from RWTH Aachen University in 2017, a master's degree in electrical engineering from RWTH Aachen University in 2019, a master's degree in automation engineering from RWTH Aachen University in 2020, respectively. He is currently working as a Research Associate at the Institute for Fluid Power Drives and Systems at RWTH Aachen University. His research areas include deep learning, physics-based learning, fluid transmission lines, and virtual sensory.

Lovis Kauderer received a bachelor's degree in computational engineering science from RWTH Aachen University in 2020, a master's degree in computational engineering science from RWTH Aachen University in 2022. He is currently working at Atlas Copco.

Gunnar Matthiesen received a bachelor's degree in mechanical engineering from RWTH Aachen University in 2011, a master's degree in mechanical engineering from RWTH Aachen University in 2013, a master's degree in management, business and economics from RWTH Aachen University in 2014, respectively. He is currently working at DMG Mori Aktiengesellschaft.

Katharina Schmitz received a graduate's degree in mechanical engineering from RWTH Aachen University in 2010 and an engineering doctorate from RWTH Aachen University in 2015. She is currently the director of the Institute for Fluid Power Drives and Systems (ifas), RWTH Aachen University.