International Journal of Dynamics and Control (2025) 13:347
https://doi.org/10.1007/s40435-025-01860-x

Model-based reinforcement learning for hydraulic systems: a PILCO-based control strategy for controlling an inverted hydraulic pendulum

Faras Brumand-Poor[1] · Paul Wagemann[2] · Timm Geibel[2] · Katharina Schmitz[1]

Received: 26 May 2025 / Revised: 7 September 2025 / Accepted: 9 September 2025

Abstract
In fluid power systems, hydraulic drives are widely used but often rely on linear controllers that require precise tuning to specific operational contexts. Traditional control methods, while effective, can be inflexible and struggle with the nonlinear aspects of hydraulic systems, such as fluid compressibility and friction. Experimental investigations to understand these systems are often impractical due to cost and safety concerns, making simulations a valuable alternative. However, accurate simulations are computationally intensive and not always feasible for real-time control. This work introduces a Probabilistic Inference for Learning and Control (PILCO) algorithm to address these challenges. PILCO is a model-based reinforcement learning (RL) algorithm that constructs a dynamics model from observed data, incorporating uncertainties to predict future system behavior with minimal environmental interactions. The algorithm was applied to control a physical inverted hydraulic pendulum, a complex task due to the system's nonlinearities. PILCO successfully learned precise control policies with only 64 seconds of interaction time, demonstrating the effectiveness of reward shaping and offline data initialization in accelerating learning. The results highlight PILCO's potential to advance optimal control in real-world hydraulic systems through efficient and adaptable RL strategies.

Keywords Control learning · Hydraulic control · Inverted hydraulic pendulum · Reinforcement learning control

1 Introduction

In fluid power, hydraulic drives are extensively employed across numerous applications [1]. Typically, these systems utilize linear controllers that must be finely tuned to their specific operational contexts [2, 3]. Implementing a suitable control design for fluid power systems requires a deeper understanding of the systems, which may be obtained by experimental or simulative investigations. However, experimental investigations are often not feasible for cost or safety reasons. Therefore, simulations represent a promising alternative to understand better the underlying physical phenomena of the system [4]. Nevertheless, a control design based on a simulation model mainly benefits from an accurate simulation, which a simple lumped-parameter simulation model does not guarantee. This is particularly complicated due to the nonlinear aspects of hydraulic systems, such as fluid compressibility and friction-related phenomena [2, 3]. Modeling systems with high accuracy is often achieved through distributed-parameter simulations, which can exhibit the precise behavior of the investigated system but usually require immense computational time and are thus not feasible in most cases for designing a control structure, especially when applying a real-time controller.

Additionally, complex control tasks often require the integration of multiple single-task controllers, rendering traditional control methods relatively inflexible, particularly as system properties change over time due to wear and tear. In contrast, reinforcement learning (RL) algorithms

✉ Faras Brumand-Poor
faras.brumand@ifas.rwth-aachen.de

Paul Wagemann
paul.wagemann@bucherhydraulics.com

Timm Geibel
Timm.geibel@bucherhydraulics.com

Katharina Schmitz
katharina.schmitz@ifas.rwth-aachen.de

1 Institute for Fluid Power Drives and Systems (ifas), RWTH Aachen University, Aachen, Germany

2 Bucher Hydraulics AG, Bucher Industries AG, Neuheim, Switzerland

offer flexible and promising solutions for dynamic, nonlinear systems [5]. With advances in computational power and successful applications in optimal control [6], RL leverages a data-driven approach to autonomously learn optimal control strategies through interaction with the environment [4]. Nevertheless, RL methods generally require extensive data, which poses challenges in real-world scenarios where frequent interactions can induce wear and safety risks [7]. Consequently, many RL-based control strategies are trained within simulation environments before being applied to actual systems. This process relies on the accuracy of the simulation model and may yield suboptimal performance if discrepancies occur.

This work presents a Probabilistic Inference for Learning (PILCO) algorithm, which uses a model-based RL algorithm to control an inverted hydraulic pendulum. Swinging up and balancing an inverted hydraulic pendulum is a classic yet complex control problem due to the system's inherent nonlinearities. PILCO constructs an underlying dynamics model solely from observed data and, by incorporating uncertainties, can forecast future system behavior, thereby significantly reducing the number of interactions required with the environment [8]. These advantages enable its direct application to a real-world setting without needing an elaborate manual simulation model. Furthermore, offline data helps minimize environmental interactions, accelerating learning while mitigating wear and tear. Implementing this model-based RL algorithm for controlling a real hydraulic system will provide valuable insights that further advance research in optimal control via reinforcement learning.

In Sect. 2, current applications of RL controllers in fluid power are presented. Subsequently, in Sect. 3, the control structure is described, by firstly introducing the test rig, secondly the PILCO controller, and thirdly the actual control loop. Afterward, the rules and conditions for the control task are outlined, and the utilization of offline data is described. Section 4 presents the results of the PILCO-based control strategy for the inverted hydraulic pendulum, and eventually a conclusion and an outlook are provided.

2 Reinforcement learning control in fluid power

While the field of RL control has been extensively researched and successfully applied in areas such as robotics and autonomous systems, the application of RL in hydraulic systems remains relatively unexplored due to the inherent difficulties of nonlinearity caused by fluid compression, flow properties, and friction. These characteristics make it difficult to effectively model and control hydraulic systems, limiting the broader adoption of RL control in fluid power. In addition, real-world application challenges complicate the practical application of RL control, which is why many researched approaches remain in simulation. Andersson et al. successfully applied a Proximal Policy Optimization (PPO) algorithm in a simulated environment to control a hydraulic crane manipulator [9]. Similarly, Wyrwał et al. developed a PPO algorithm to control the position of a hydraulic cylinder, outperforming a professionally tuned PID controller due to the RL algorithm's ability to better handle offsets and nonlinearities in the system [10]. In studies beyond simulations, RL algorithms were trained in virtual environments before they were transferred to and validated on real-world systems. For instance, Yao et al. used a model-based RL approach with an actor-critic algorithm to control an electro-hydraulic system [11]. The model was implemented manually, and the actor neural network approximated unmodeled and unknown friction in the system [11]. While the method was effective in simulation and physical experiments, the authors concluded that modeling complex hydraulic systems may not be feasible and suggested focusing on data-driven RL approaches in the future [11]. Similarly, Bécsi et al. applied Q-learning and policy gradient methods to control an electro-pneumatic gearbox actuator, showing good control results in simulation and physical validation [12]. Additionally, Berglund and Larsson developed a Deep Q-Network (DQN) to control gear selection in a hydraulic gearbox, training it in simulation and testing it on a physical test rig, with promising results [13]. Yuan et al. implemented a model-free policy gradient method (TD3) for a hydraulic servo system, where they first trained the agent using a simulation environment and then conducted real-world experiments that verified the feasibility and effectiveness of the trained control strategy [14]. Gao et al. also trained a model-free RL algorithm for an electrical-hydraulic valve-controlled system in simulation, which was then tested on balancing a real hydraulic inverted pendulum, although due to limitations of the test bench only under low-pressure conditions [15]. Brumand-Poor et al. took a similar approach, learning a deep RL algorithm to swing up and balance an inverted pendulum in simulation, followed by a feasible validation on a hydro-mechanical inverted pendulum [2, 3].

3 Hybrid PILCO controller for the inverted hydraulic pendulum

3.1 The test rig

The test rig used in this research is a hydraulically driven inverted pendulum, designed by BUCHER Hydraulics, to evaluate advanced control strategies, such as the Probabilistic Inference for Learning Control (PILCO) framework, and is shown in Fig. 1. It consists of two end plates connected by linear guides, which themselves support the cart of the pendulum. The cart has a mass of 600 kg and can be moved

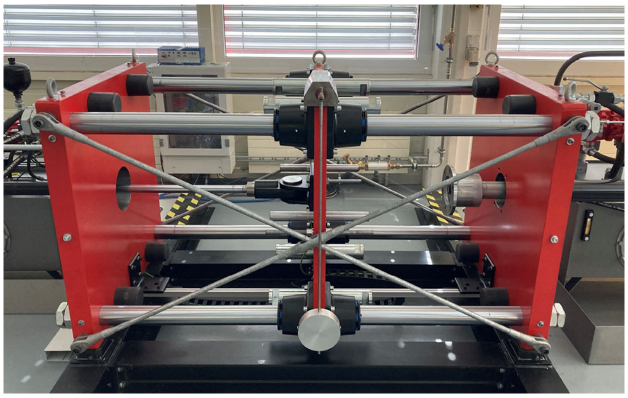

Fig. 1 Inverted pendulum test rig

horizontally along the x-axis using the linear guides. On this cart, a pendulum pole is attached at a pivot point, allowing it to rotate freely. The pendulum has a total length of 0.875 m, measured from the pivot to its tip, where a 1 kg mass is fixed. By applying a force to the cart, it is initiated with a linear motion, which results in a swinging movement of the pendulum.

In contrast to conventional cart-pole systems, typically powered by purely electrical drive trains, the test rig used in this research consists of the HELAX system, an electro-hydraulic system designed by Bucher Hydraulics. A volumetric flow rate is applied to the HELAX system by giving speed inputs to the electric motor. This volumetric flow rate controls the synchronous cylinder, which moves the cart along the x-axis to swing up the pendulum. Since no control valve is used, the required volumetric flow rate is set directly via the motor, making the HELAX a displacement control system. This makes the system particularly efficient, with low friction and minimal leakage even at low speeds. Nevertheless, this test rig introduces significant nonlinearities due to factors such as fluid compressibility and variable flow dynamics. Additionally, frictional effects, as well as temperature-dependent fluid properties, contribute to the system's complex and nonlinear behavior, making accurate modeling and control challenging.

The control task of the PILCO algorithm is to determine the optimal rotational speed n of the electrical motor at every sampling step, which results in the required linear motion of the cart to swing up and balance the pendulum. Since the PILCO algorithm only knows about the state variables of the cart and the pendulum, as well as the rotational speed of the motor, all mechanical and physical properties occurring between the motor and cart, such as nonlinearities due to friction and compression, do not have to be modeled. Instead, the algorithm must learn these behaviors and the resulting influences on the system.

The state variables of the inverted pendulum used to calculate the rotational speed input n of the electric motor are the cart position x (m), the cart velocity $\dot{x}$ (m/s), the pendulum angle θ (rad), and the angular velocity of the pendulum $\dot{\theta}$ (rad/s). Therefore, the state space x_t of the inverse pendulum is given by

$$x_t = [x, \dot{x}, \dot{\theta}, \theta]^T,$$

Since the measured value of the pendulum's angle adds up with each rotation, the state space is augmented with the complex representations of the angle. Therefore, the state space results in:

$$x_t = [x, \dot{x}, \dot{\theta}, \theta, \sin\theta, \cos\theta]^T \tag{1}$$

Accurate modeling of the dynamic system is necessary to perform successful state predictions. This implies the need for precise and coherent data on the state variables. However, in the hydraulic pendulum test rig, only the position data are measured directly at high resolution. A linear encoder measures the cart position x on the linear axis, and the pendulum's angle θ is measured by an angle encoder in the pendulum mount. The cart velocity $\dot{x}$ and the angular velocity $\dot{\theta}$ are computed directly from the position data inputs. The position data signals were first filtered and then derived from this. This is necessary because deriving it without filtering would result in noisy velocity data, despite the negligible noise in the position data. For this purpose, a *Savitzky–Golay* filter is used. The Savitzky–Golay filter is a digital filter designed to smooth a set of data points while preserving the shape and important features of the signal. It performs a local polynomial fitting to the position data over a sliding window and calculates the derivative of the fitted polynomial directly. This approach enables simultaneous noise reduction and differentiation, providing a smooth estimate of velocity without requiring a separate differentiation step. After obtaining the velocity data from the Savitzky–Golay filter, a lowpass *Butterworth* filter is applied to further smooth all signals. The Butterworth filter is a digital filter used to reduce high-frequency noise while preserving the essential low-frequency components of the signal, such as the main trends in the velocity data.

3.2 The PILCO

Most conventional RL control algorithms are impractical for real-world applications due to their data inefficiency and the subsequent prohibitively high number of interactions with the environment [16]. To improve data efficiency, deterministic model-based RL algorithms are a promising approach; however, they come with the drawback of model bias [8]. Using a probabilistic model-based approach that can express its uncertainty, accurate long-term predictions can be made in regions with little or no training data, thereby reducing the amount of required training data [16]. The *Probabilistic Inference for Learning Control* (PILCO) algorithm is a policy search method that utilizes nonparametric *Gaussian processes* (GPs) as probabilistic models to enhance data efficiency and mitigate model bias, one of the primary challenges in model-based RL [8]. A Gaussian process is a probabilistic framework used in machine learning for regression, classification, and dimensional reduction [17]. It is used to describe a distribution over functions $p(f)$ to make predictions with uncertainty [18]. One main advantage of other approaches is that GPs are *nonparametric* models and therefore do not require a predefined set of basis functions or a fixed number of parameters [19]. This makes GPs more flexible in determining the underlying function from data [20]. As an indirect policy search method, the PILCO algorithm is capable of continuous state-action domains and does not require an explicit value function model [8]. This subsection gives an insight into the mathematical foundations of the PILCO algorithm used in this work, based on the original paper by Deisenroth and Rasmussen [8]. For further introduction, the reader is advised to read their work.

The overall idea of using the PILCO algorithm is to model dynamic systems by predicting the next state of the system $\mathbf{x}_t$ from an input, which itself consists of the concatenation of the previous state $\mathbf{x}_{t-1}$ and action $\mathbf{u}_{t-1}$

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}),$$

with $\mathbf{x} \in \mathbb{R}^D$, $\mathbf{u} \in \mathbb{R}^F$, and f being the unknown transition dynamics. From that, the goal is to learn a deterministic policy π, also referred to as a controller, that minimizes the overall expected return

$$J^\pi(\theta) = \sum_{t=0}^{T} \mathbb{E}_{\mathbf{x}_t}\left[c(\mathbf{x}_t)\right], \quad \mathbf{x}_0 \sim \mathcal{N}(\mu_0, \Sigma_0), \tag{2}$$

over the time steps T, where θ are the policy parameters and $c(\mathbf{x}_t)$ is the negative reward, modeled as a loss function. To reach that goal, the PILCO framework consists of three main components. In the first step, the dynamics model is learned from data. With this learned model, the current policy is then evaluated over long-term predictions, using the model to simulate its performance. In a third step, the policy is optimized using analytic gradient computation. [8]

The dynamics model of the PILCO, consisting of GPs, is learned from data obtained from the interaction with the environment. As training inputs, tuples of the previous state and action $(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \in \mathbb{R}^{D+F}$ are used and the resulting state differences $\Delta_t = \mathbf{x}_t - \mathbf{x}_{t-1} + \epsilon, \quad \epsilon \in \mathbb{R}^D, \quad \epsilon \sim \mathcal{N}(0, \Sigma_\epsilon)$ are considered as training targets. The GP makes one-step predictions of the next state, providing both the mean $\boldsymbol{\mu}_t$ and covariance Σ_t to capture the uncertainty in the predictions: [8]

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathcal{N}(\mathbf{x}_t|\mu_t, \Sigma_t), \tag{3}$$

with

$$\mu_t = \mathbf{x}_{t-1} + \mathbb{E}_f[\Delta_t],$$

and

$$\Sigma_t = \mathrm{var}_f[\Delta_t].$$

with $\mathbb{E}_f[\Delta_t]$ being the mean and $\mathrm{var}_f[\Delta_t]$ the variance of the differences.

For a single test input x and with the use of a zero-mean prior, the posterior distribution consisting of mean and variance can be calculated in its general form through

$$m_{\text{post}}(x) = \mathbb{E}[f(x)] = K(x, X)\left(K(X, X) + \sigma_n^2 I\right)^{-1} \mathbf{y} \tag{4}$$

and

$$\mathrm{var}[f(x)] = k(x, x) - K(x, X)\left(K(X, X) + \sigma_n^2 I\right)^{-1} K(X, x) \tag{5}$$

with the kernel function $k(x, x)$, the noise variance σ_n^2, and the kernel matrix $K(X, X)$ defined as

$$K(X, Z) := \begin{bmatrix} k(\mathbf{x}_1, \mathbf{z}_1) & k(\mathbf{x}_1, \mathbf{z}_2) & \cdots & k(\mathbf{x}_1, \mathbf{z}_{\hat{N}}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{z}_1) & k(\mathbf{x}_N, \mathbf{z}_2) & \cdots & k(\mathbf{x}_N, \mathbf{z}_{\hat{N}}) \end{bmatrix} \in \mathbb{R}^{N \times \hat{N}}$$

[8, 19]. A common choice for the kernel function is the squared exponential (SE) kernel

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right),$$

with σ_f being the signal variance and ℓ being the length scale [19]. This form indicates that the resulting mean is directly dependent on the choice of kernel and, respectively, the hyperparameters [19]. In this work, a zero-mean prior as well as an SE kernel is used. By fitting the GP to the observed state-action pairs and state changes, the GPs hyperparameters length scale, signal variance, and noise variance are learned by maximizing the log marginal likelihood of the data under the GP model. This ensures that the GP best fits the data while controlling for model complexity, allowing it to model the system dynamics accurately while accounting for uncertainty in the data. For multivariate targets, independent GPs are trained for each target dimension. [8]

To evaluate the policy in the second step, long-term predictions of the future states of the environment are made over the planning horizon T. To do so, one-step predictions are cascaded through the GPs across multiple time steps to obtain the target state distributions $p(\mathbf{x}_1)$, ..., $p(\mathbf{x}_T)$. To predict $\mathbf{x}_t$ from $p(\mathbf{x}_{t-1})$, a joint distribution of the input state and action $p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$ is needed. To achieve this, the mean $\mu_\mathbf{u}$ and covariance $\Sigma_\mathbf{u}$ of the predictive action distribution $p(\mathbf{u}_{t-1})$ are calculated first. By computing the cross-covariance $\text{cov}[\mathbf{x}_{t-1}, \mathbf{u}_{t-1}]$, the Gaussian joint distribution at time $t-1$ can then be approximated with: [8]

$$p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = p(\tilde{\mathbf{x}}_{t-1}) = \mathcal{N}(\tilde{\mathbf{x}}_{t-1}|\tilde{\mu}_{t-1}, \tilde{\boldsymbol{\Sigma}}_{t-1}) \tag{6}$$

This expression directly illustrates the key challenge of the policy evaluation. Instead of having a fixed, deterministic input as in the model learning phase, the input distributions of states and actions are Gaussian with a mean and a covariance and are therefore uncertain. In other words, the uncertainty of each input has to be propagated through the GPs. When predicting the next state change Δ_t, a single input cannot simply be used but must be integrated over the possible inputs weighted by their probabilities. Since computing the exact predictive distribution $p(\Delta_t)$ is analytically challenging, the PILCO algorithm uses *moment matching*, to approximate the distribution of $p(\Delta_t)$, as Gaussian by its exact mean and covariance. Moment matching is the process of propagating uncertainty through the GP dynamics model. For reference, the detailed calculation steps can be found in [8]. Once the predictive distribution $p(\Delta_t)$ is obtained, the target distribution $p(\mathbf{x}_t)$ is given by

$$p(\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_t|\mu_t, \boldsymbol{\Sigma}_t), \tag{7}$$

with the mean

$$\mu_t = \mu_{t-1} + \mu_\Delta$$

and the covariance

$$\boldsymbol{\Sigma_t} = \boldsymbol{\Sigma_{t-1}} + \boldsymbol{\Sigma_\Delta} + \text{cov}[\mathbf{x}_{t-1}, \Delta_t] + \text{cov}[\Delta_t, \mathbf{x}_{t-1}].$$

The cross-covariance terms in the covariance calculation depend on the policy parameterization θ and are computed as [8] :

$$\text{cov}[\mathbf{x}_{t-1}, \Delta_t] = \text{cov}[\mathbf{x}_{t-1}, \mathbf{u}_{t-1}]\boldsymbol{\Sigma}_\mathbf{u}^{-1}\text{cov}[\mathbf{u}_{t-1}, \Delta_t]$$

In order to evaluate the current policy π, the values of the expected return in Equation 2 dependent on the loss function $c(\mathbf{x}_t)$ can then be computed [8]:

$$\mathbb{E}_{\mathbf{x}_t}\{c(\mathbf{x}_t)\} = \int c(\mathbf{x}_t)\mathcal{N}(\mathbf{x}_t|\mu_t, \boldsymbol{\Sigma}_t)\,\mathrm{d}\mathbf{x}_t. \tag{8}$$

The final step of the PILCO framework is to optimize the policy π, or more precisely, to find the optimal policy parameters θ^* that minimizes the cumulative expected return J^π in Equation 2 [16]. Since the mean μ_t and covariance $\boldsymbol{\Sigma}_t$ are functionally dependent on the mean μ_u and covariance $\boldsymbol{\Sigma}_u$ of the action distribution, which itself depends on the policy parameter θ, respectively, the analytic gradients of the expected return w.r.t the policy parameters θ can be computed, to update the policy using gradient-based optimization methods [8]. This is done under the condition that the expected return in Equation 8 is differentiable w.r.t. mean μ_t and covariance $\boldsymbol{\Sigma}_t$ of the state distribution [16]. Furthermore, it is assumed that the mean μ_u and covariance $\boldsymbol{\Sigma}_u$ of the action distribution are differentiable w.r.t. the policy parameters θ [16]. The derivative $\mathrm{d}J^\pi/\mathrm{d}\theta$ is obtained by repetitive application of the chain rule [8]:

$$\frac{\mathrm{d}J^\pi}{\mathrm{d}\theta} = \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}\theta} = \frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}p(\mathbf{x}_t)}\frac{\mathrm{d}p(\mathbf{x}_t)}{\mathrm{d}\theta} := \frac{\partial\mathcal{E}_t}{\partial\mu_t}\frac{\mathrm{d}\mu_t}{\mathrm{d}\theta} + \frac{\partial\mathcal{E}_t}{\partial\boldsymbol{\Sigma}_t}\frac{\mathrm{d}\boldsymbol{\Sigma}_t}{\mathrm{d}\theta}, \tag{9}$$

with

$$\frac{\mathrm{d}\mathcal{E}_t}{\mathrm{d}p(\mathbf{x}_t)} := \left\{\frac{\partial\mathcal{E}_t}{\partial\mu_t}, \frac{\partial\mathcal{E}_t}{\partial\boldsymbol{\Sigma}_t}\right\}.$$

The derivations $\mathrm{d}\mu_t/\mathrm{d}\theta$ and $\mathrm{d}\boldsymbol{\Sigma}_t/\mathrm{d}\theta$ are then calculated by using the partial derivatives $\partial\mu_u/\partial\theta$ and $\partial\boldsymbol{\Sigma}_\mathbf{u}/\partial\theta$, which itself depend on the parameterization of the policy θ. Since all required derivatives can be calculated analytically, a gradient-based optimization method can then be applied. One optimization method used in the PILCO algorithm is the quasi-Newton method L-BFGS to obtain the optimized policy parameter θ^*. [8]

Once the optimal policy parameters θ^* have been derived in the PILCO algorithm, the action output is computed through the policy function $\pi(x)$. Usually, a nonlinear policy function is used, represented by a deterministic GP that calculates the action based on the current measured state. The deterministic GP states that there is no uncertainty about the policy function $\text{var}_\pi[\pi(x)] = 0$. For the nonlinear policy, several fixed basis functions, nc, must be set in advance, which affects the complexity and flexibility of the policy. [16]

Figure 2 shows a simplified flowchart of the PILCO framework described. The three main components—model learning, policy learning, and policy application—are executed iteratively over multiple cycles. After each policy application to the environment, also referred to as a rollout, the collected data from this rollout are used to update the dynamics model. In the initial step, when there is no policy to collect data, random action inputs are given to the environment to collect the initial data. The process of updating

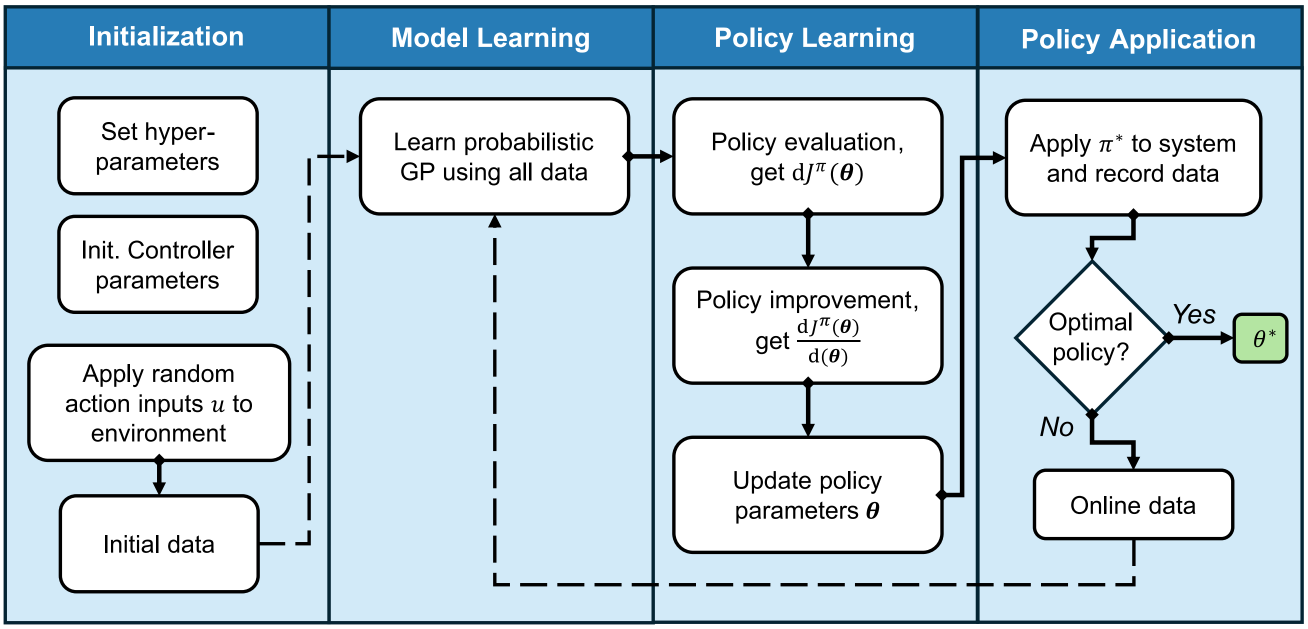
Initialization
Model Learning
Policy Learning
Policy Application
Set hyper-parameters
Init. Controller parameters
Apply random action inputs u to environment
Initial data
Learn probabilistic GP using all data
Policy evaluation, get $\mathrm{d}J^{\pi}(\boldsymbol{\theta})$
Policy improvement, get $\frac{\mathrm{d}J^{\pi}(\boldsymbol{\theta})}{\mathrm{d}(\boldsymbol{\theta})}$
Update policy parameters $\boldsymbol{\theta}$
Apply π^* to system and record data
Optimal policy?
Yes
No
θ^*
Online data

Fig. 2 Flowchart of the PILCO algorithm with its main modules [21]

and learning the dynamics model is cumulative, meaning that all previously gathered data are used during each optimization phase, not just the data from the most recent rollout. This is crucial because PILCO utilizes GPs to model system dynamics, which are incrementally updated with new data. By leveraging the entire dataset from all rollouts, PILCO ensures that understanding system dynamics is constantly improved while considering the older data points. This leads to more accurate predictions and better policy optimization in subsequent rollouts.

3.3 The control loop

When using the PILCO algorithm in a real-world environment, only the policy application (compare Fig. 2) must be performed in real-time [21]. The actual model and policy learning process occurs outside the real-time policy application. In this research, the policy is embedded and used with Simulink® and Speedgoat®, a hardware test system for embedded controllers. Figure 3 shows the workflow used for the control loop of the policy application. First, the optimized policy parameters are loaded into the Simulink® model. After the model is built, it is compiled and loaded onto the Speedgoat®. Once running, the controller interacts with the pendulum plant via PROFINET®. At each time step, the policy calculates the corresponding action output u_t based on the current state space x_t. In this research, the action output u_t represents the set point for the rotational speed n of the motor that actuates the pump. After computing the action output u_t, it is appended to the state variables of the system at time step t. Furthermore, the loss L of the current state is calculated based on the defined loss function $c(x_t)$. Both the state x_t with the appended action u_t and the loss L_t are then saved together as input data. Once the motor has set its value to the computed action output value, the system transitions to its next state at the following sample time step, $t+1$, where the new state variables, x_{t+1}, are observed. These new state variables are then saved as the target data. Subsequently, the latest action output u_{t+1} and the loss L_{t+1} for that time step are computed. This process repeats at each sample time step until the rollout ends. After each rollout, the observed and saved input and target data points are exported and transferred to the MATLAB® code. The dynamics model is learned with the training inputs and targets, followed by optimizing the policy π.

3.4 Rules and conditions of the controller

To successfully apply the PILCO algorithm to the real-world environment, it is necessary to ensure the controller's real-time capabilities. Since the policy calculates the action output u_t for a given state space x_t of the pendulum at each time step t, the output calculation must, in any case, be faster than the selected sample time. Furthermore, the choice of sample time is critical. On the one hand, a fast sample time is necessary to capture the pendulum's dynamics and provide precise control accurately. On the other hand, however, this would generate more data points and, therefore, increase the computational cost of the PILCO algorithm. Additionally, since the PILCO algorithm relies on Gaussian process models to predict system behavior, faster sample times require more prediction steps into the future over the same time span,

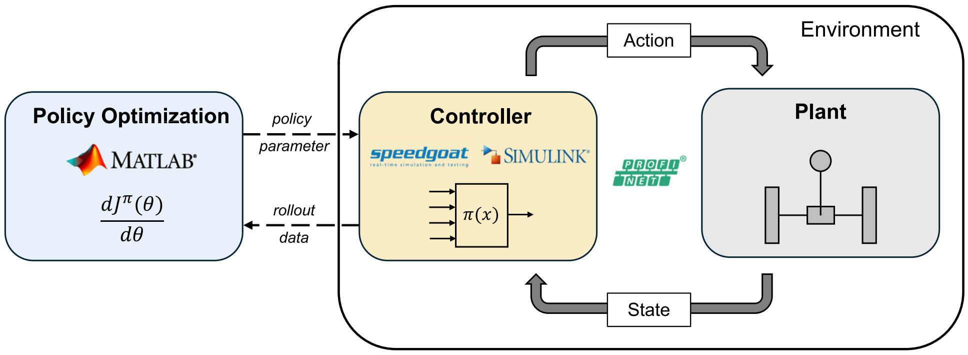

Fig. 3 Workflow of the control loop

thereby increasing computational complexity and reducing predictive accuracy due to the consideration of uncertainty in long-term predictions. Therefore, an optimal balance of the chosen sample time must be found. The sample time should be fast enough to allow adequate control while remaining slow enough to ensure that PILCO's predictions remain computationally feasible.

To cope with this challenge, a strategy was adopted in this work to use a fast sample time during the policy application with the environment while simultaneously keeping the number of prediction steps low during the policy optimization: During the rollouts, where the agent interacts with the environment to balance the pendulum and collect data for further policy optimizations, the fast sample time of $dt_{control} = 0.005$ seconds was chosen. After the data points were collected from the rollouts, the sample time was artificially reduced by downsampling the dataset. Specifically, only every 20$^{\text{th}}$ data point from the original dataset is used for model learning and policy optimization. Following this approach, the number of prediction steps can be effectively reduced while maintaining the overall trend of the system dynamics. Without this downsampling, the model learning and policy optimization of the GPs would be computationally infeasible and therefore beyond the practical scope of this work. Nevertheless, artificially downsampling the data can lead to less accurate system dynamics models and likely result in a less fine-tuned control policy [22]. Taking every 20$^{\text{th}}$ data point, the sample time dt of the policy evaluation and optimization results in $dt_{optimization} = 0.1$ seconds.

3.5 Offline data utilization

In the original implementation of the PILCO algorithm, the agent's first rollout generates data for initial model learning and policy optimization by applying random action outputs to the system. This is suitable for simulated environments with no physical constraints; however, when working in real-world systems, random actions can cause damage to the system, which, in the worst case, can lead to system failure. For example, when examining the movement of the cart along the x-axis, many simulation environments do not impose a limitation on the direction of the x-axis. This means the agent can move the cart as far as possible in each direction without consequences. In contrast, the freedom of movement of the real hydraulic pendulum used in this work is constrained. Therefore, moving the cart at a high speed to one end plate of the hydraulic pendulum could lead to severe damage to the test bench.

Instead of generating data for the first policy optimization through random actions, this work utilizes offline data to optimize the policy initially. The targeted use of offline data within the first rollout avoids the risk of applying random actions. It places the policy directly into a good starting domain on which future optimizations can be developed. In this work, the offline data replicated the swing-up of the pendulum for $T = 8$ seconds, which was recorded in advance by manually operating the motor and moving the cart using a joystick. With the given sampling time of $dt_{optimization} = 0.1$, a total of 80 offline data points were used. The PILCO framework using offline data is shown in Fig. 4. Seeing a perfect swing-up and balancing the pendulum in the offline data are unnecessary. Instead, they should represent a range of state spaces where the policy should be implemented and improved. Ideally, they should show the PILCO states where the reward is high and how to get there.

4 Results

Two experimental test runs with different loss functions were carried out. The two trained policies are compared based on the evaluation metrics presented in Sect. 4.1, and their behavior will be evaluated from different perspectives about the respective loss function used. Thereby, only a selection of rollouts is described. Furthermore, it is to be noted that a supposedly good policy that can swing the pendulum upwards and balance it over. The remaining planning horizon T does not automatically guarantee balancing over an indefinite time. For this reason, assumed good policies are then checked by applying them over an indefinite time. It

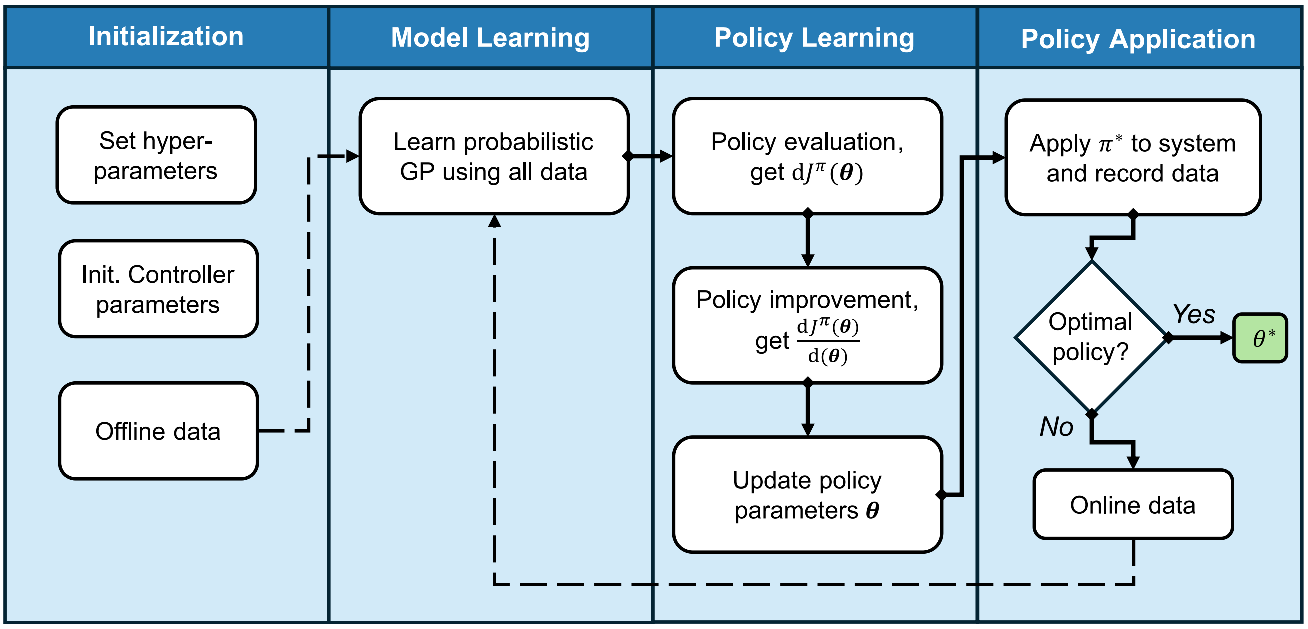
Initialization
Model Learning
Policy Learning
Policy Application
Set hyper-parameters
Init. Controller parameters
Offline data
Learn probabilistic GP using all data
Policy evaluation, get $dJ^{\pi}(\boldsymbol{\theta})$
Policy improvement, get $\frac{dJ^{\pi}(\boldsymbol{\theta})}{d(\boldsymbol{\theta})}$
Update policy parameters $\boldsymbol{\theta}$
Apply π^* to system and record data
Optimal policy?
Yes
No
θ^*
Online data

Fig. 4 Flowchart of the PILCO algorithm with offline data utilization [21]

should be noted that this work conducted a feasibility study with a focus on achieving the goal of a successful swing-up and balancing of the pendulum. For this reason, the termination criterion for policy learning is reached when these policies achieve stable balancing for an indefinite time. Further studies would need to examine whether the policies could be improved even further with continued model and policy learning, and to what extent these improvements are beneficial about the cost-intensive learning.

4.1 Evaluation metrics

Different evaluation metrics are used to compare the conducted runs with one another. The primary goal is to learn the swing-up and balancing of the pendulum over the given planning horizon T. However, even if this goal is achieved, the individual runs with their rollouts may differ significantly from each other, which is why clearly defined evaluation metrics are necessary. Table 1 lists the five evaluation metrics used in this work to capture different aspects of the system's behavior and learning efficiency.

The first metric is the *total loss* ΣL, which represents the sum of the loss accumulated over the whole planning horizon T. This can either be presented as the *function value* ΣL_{pred}, which is the sum of the predicted loss trajectory determined by the model during policy optimization, or by the sum of the real loss trajectory ΣL_{real} of the applied policy of each rollout. The lower the total loss, the supposedly better the optimized policy; however, it is also crucial whether and how the predicted loss trajectory matches the actual applied loss trajectory. If there are significant differences, especially in areas where the predicted loss trajectory has a low variance, it is usually a sign of poor modeling of the system's dynamics. For the second evaluation metric, the *time until balancing* is considered, which measures how quickly the system stabilizes the pendulum in the upright position. A shorter time to balance reflects a more effective and responsive policy.
The third metric evaluates the *interaction time* of the algorithm with the environment, i.e., the accumulated time of policy application (compare Fig. 4), represented by the number of data points needed to achieve balancing. This metric evaluates the data efficiency of the learning process, as fewer interactions or data points indicate that the policy can be learned with less real-world data, a crucial consideration when data collection is costly or time-consuming.
Furthermore, the quality of the policy is considered. On the one hand, the *Integral of Time-weighted Absolute Error* (ITAE) is used to assess the quality of the swing-up and balancing phase of the cart-pole system. The ITAE is calculated by integrating the time-weighted absolute error of the pendulum's angle from the target upright position over the entire rollout:

$$\text{ITAE} = \int_0^T t \cdot |e(t)| \, dt, \tag{10}$$

with $e(t) = \theta(t) - \theta_{\text{target}}$ being the error between the current and target angle of the pendulum. By capturing both the magnitude and duration of deviations, it penalizes unwanted behaviors, such as overshooting the pendulum, providing a comprehensive measure of policy performance. On the other hand, the *quality of the balancing* is evaluated by measur-

Table 1 List of evaluation metrics

Metric	Definition
Total loss	Sum of the loss accumulated over the planning horizon T
Time until balancing	Time until target angle is reached
Interaction time	Quantity of interactions with the environment
ITAE	Time-weighted absolute error between current and target angle
Constraint violations	Number of times the cart is moved into the limits of the test bench
Quality of balancing	Standard deviation of the control input actions

ing the standard deviation of the control input actions once the pendulum reaches the upright position. A lower standard deviation suggests smoother and more stable control during the balancing phase, indicating that the controller applies consistent and measured actions. In contrast, a higher standard deviation would indicate a relatively high degree of dynamic or unstable behavior, suggesting that the policy consistently struggles to maintain an upright position.

An additional essential aspect of the policy is the ability to respect specified safety regulations. Since the policy is applied to a real-world system, specific actions can have serious consequences that damage the system, making it essential to follow these constraints. The metric *constraint violations* indicates how often the respective policy has violated safety specifications in terms of moving the cart into the physical limit of the test bench.

Lastly, another parameter is presented in the results to facilitate comparison among the test runs. The PILCO algorithm learns the noise standard deviation σ_{noise} during the model learning through data [21]. This is an essential parameter in the GP model that reflects the amount of noise, or random variation, present in the data. This noise can result from measurement errors or small, unpredictable changes from the actual system behavior. A higher learned noise standard deviation σ_{noise} means the model sees more noise in the data, making more cautious predictions to avoid overfitting to random variations. On the other hand, a lower noise standard deviation σ_{noise} suggests the data are cleaner and more reliable, allowing the model to make more confident predictions. By learning this noise parameter, PILCO can focus on capturing the actual system dynamics while ignoring irrelevant fluctuations.

4.2 Hyperparameter tuning

Although the PILCO is a self-learning algorithm that learns the underlying mathematical model using data, this does not exclude the need to specify parameters in advance. The choice of suitable *hyperparameters* has a decisive influence on the learning behavior of the algorithm and can significantly improve model performance. However, finding ideal parameters is difficult due to the high degree of possible variations. Standard techniques include a grid or random search, but more complex optimization methods can also be used to find suitable hyperparameters. The following section presents the hyperparameters of the PILCO algorithm for the inverted pendulum tasks. The values are chosen partly based on recommendations, such as the software documentation of the PILCO algorithm [21], and partly through a heuristic grid search.

One of the most essential hyperparameters is the number of prediction steps H that the PILCO algorithm must complete during policy learning. It defines how far the algorithm will look into the future, or rather, how many future states it will predict to evaluate and optimize the policy. The number of prediction steps H is derived through the planning horizon T, which is the interaction time of the agent with the environment, and the sample time dt:

$$H = \frac{T}{dt}$$

However, choosing a suitable number of prediction steps H, respectively, the planning horizon T combined with an appropriate sample time dt, leads to a trade-off: On the one hand, the fewer prediction steps PILCO has to make, the better, since the computational cost increases significantly with more prediction steps. For each additional time step in the prediction horizon, PILCO must perform more forward simulations, in which uncertainties are propagated through the GPs, becoming increasingly computationally expensive as the prediction horizon lengthens. Furthermore, predictions made far in the future tend to be less accurate due to the compounding effect of uncertainties. On the other hand, the planning horizon T should be chosen so that the target state, the upward position of the pendulum, can be reached by the agent at all. Since the system in this work has a certain inertia due to the high masses of the test bench, it can take some time for the pendulum to swing up. Furthermore, a sufficient amount of time should also be provided after the pendulum has swung up to ensure that the agent learns to balance the pendulum successfully. Therefore, the planning horizon T should be sufficiently long to capture the key dynamics of the system, but not so long that it renders the optimization problem computationally intractable or overly uncertain. The planning horizon T in this work is set to $T = 8$ seconds.

A compromise must also be made when choosing the sample time dt. From the perspective of the control strategy when applying the controller to the system, the sample time dt, which is the time interval between each control action, should be fast enough to enable control of the system. However, since the sample time dt is in the denominator, it should be as slow as possible to reduce the number of prediction steps H. For this reason, the strategy in this work involves using artificial downsampling of the dataset (compare Sect. 3.4): During policy application, a fast sample time of $dt_{\text{control}} = 0.005$ seconds was chosen. A fast sample time is necessary to handle the system's fast dynamics, as slow sampling can lead to delayed actions, making stabilization more difficult. A higher sample time was not possible due to the action computing time of the controller. By artificially downsampling, specifically taking only every 20th data point from the original dataset, the sample time dt of the policy evaluation and optimization results in $dt_{\text{optimization}} = 0.1$ seconds. Following this approach, the number of prediction steps H can be effectively reduced while maintaining the overall trend of the system dynamics.

Further hyperparameters are the initial state mean $mu0$ and the initial state covariance $S0$. In this work, the initial state of the cart is at rest, positioned in the middle of the x-axis with the pendulum hanging down at position $\theta = \pi$. With Equation 1, the initial state mean is defined as:

$$mu0 = [0, 0, 0, \pi, 0, -1],$$

The initial state covariance matrix $S0$ describes the uncertainty of the system's initial state mean $mu0$. It has only diagonal elements, each corresponding to the respective state variable. Since the off-diagonal elements are zero, no correlation is assumed between the uncertainties of the state variables. As the starting position of the pendulum is set manually at the beginning of each rollout, the uncertainty of being in that specific initial state mean $mu0$ is very low, which is why the initial state covariance $S0$ is set to:

$$S0 = \text{diag}(0.0001^2, 0.0001^2, 0.0001^2, \\ 0.0001^2, 0.0001^2, 0.0001^2).$$

In this work, a zero-mean prior and an SE kernel are used for the GPs to learn the model dynamics. Regarding the policy, the maximum amplitude of the control value has to be set. A squashing function ensures that the calculated action output is never higher than the set maximum value. The maximum action output is set to $u_{\max} = 3500$ rpm, with $u \in [-u_{\max}, u_{\max}]$. Another policy-specific hyperparameter is the number of controller basis functions nc of the policy π. The higher the number of basic functions nc, the better complex nonlinear behaviors of the system can be captured, and the more flexible the controller becomes. On the other hand, choosing too many increases the computational complexity of the controller and may risk overfitting the underlying noise. The number of controller basis functions nc used in this work is set to $nc = 70$.
Furthermore, the initial values for the policy structure have to be set. Since the policy is represented as a deterministic GP, the policy structure consists of inputs, targets, and hyperparameters. These are adopted unchanged from the software documentation of the PILCO algorithm [21]. The policy inputs correspond to the centers of the policy equation and are initially sampled from the Gaussian distribution of the initial state $p(x_0)$. Similarly, the initial policy targets are set close to zero. The policy hyperparameters consist of the logarithmic values of the length scales, signal variance, and noise variance and are initially set to:

$$policy.hyp = \log([1, 1, 1, 0.7, 0.7, 1, 0.01])'.$$

Finally, the maximum number of iterations, $opt.length$, during policy optimization must be set. This hyperparameter indicates how often the policy is optimized during policy learning. It should be sufficiently high to ensure the policy parameters converge. Still, it can increase computational cost as the policy parameters can hardly be improved beyond a certain point. In this work, the number of iterations is first set to $opt.length = 150$; however, an early-stopping technique is implemented, which stops the optimization as soon as the iterations do not improve anymore. Furthermore, an approach is used to dynamically adjust the maximum number of line searches based on the convergence behavior observed in previous rollouts. This is because during early rollouts, the policy parameters typically exhibit strong convergence over multiple iterations. In contrast, in later rollouts, they are already in regions of good convergence and no longer require numerous iterations. Using this approach, the computational cost can be reduced. All values of the essential hyperparameters used in this work are summarized in Table 2.

4.3 Euclidean distance

To solve a given task, the agent must know whether the particular state it is currently in is good or bad in relation to its goal. Specifically, a policy π_* is optimal if the expected return J^π over all states is minimized. The expected return depends on the loss function, implemented as a negative reward in the PILCO algorithm. The higher the value of the loss function, the worse the agent's state is, and vice versa. By getting this information about a particular state, the agent is guided to the optimal state. Since the loss function does not instruct the agent on how to perform the task, but instead determines

Table 2 List of used hyperparameters of the PILCO algorithm

Hyperparameter	Definition	Value
dt_{control}	Sample time of policy application	0.005 s
$dt_{\text{optimization}}$	Sample time of policy learning	0.1 s
T	Planning horizon	8 s
H	Number of prediction steps	80
$mu0$	Initial state mean	$[0, 0, 0, \pi, 0, -1]$
$S0$	Initial state covariance	$(0.0001^2) \cdot I_6$
nc	Number of controller basis functions	70
$u_{\max}$	Max. amplitude of the action input	3500 rpm
$policy.hyp$	Initial policy hyperparameters	$\log([1, 1, 1, 0.7, 0.7, 1, 0.01])'$
$opt.length$	Number of optimization iterations	150
x_{target}	Target state	$[0, 0, 0, 0, 0, 1]$
$\sigma_{c,\text{sat}}$	Loss width of the saturated loss	0.6
$\sigma_{c,\text{quad}}$	Loss width of the quadratic loss	20
w	Weight factor of the quadratic loss	10
a	Slope of the double-hinge loss	500
b_1, b_2	Corner points of the double-hinge loss	$[-0.42, 0.42]$ m

whether a state is good or bad, it is crucial to define the loss function in a specific way to achieve the desired behavior of the agent. Therefore, the shape of the loss function is always dependent on the task's goal and must be defined individually for each scenario.

In the original PILCO framework for the inverted pendulum, a saturated loss function is used to calculate the Euclidean distance between the pendulum's tip and its target position. Considering the Euclidean distance, this loss function effectively penalizes deviations from the pendulum's target position. This approach is well suited for scenarios with a large or unlimited usable x-axis and a relatively small pendulum length, as in the original PILCO simulation environment. However, considering the Euclidean distance as a loss function presents challenges when applied to the test rig used in this research. With a much longer pendulum length of 0.875 m and a smaller usable range of motion on the x-axis, the loss function would remain nearly constant and close to its maximum value throughout most of the motion outside the desired target state, even if the motion might be feasible for reaching the target state. This would lead to an unlearnable behavior in the PILCO algorithm, as the algorithm lacks a clear transition in the loss function that would indicate incremental improvements in the pendulum's trajectory.

In the original PILCO framework for controlling an inverted pendulum, the loss function is calculated using the Euclidean distance between the pendulum tip and its target position [8]. This approach effectively penalizes deviations from the target state by measuring the straight-line distance in Cartesian coordinates. While this formulation works well in environments with a large or even unlimited x-axis range and a relatively short pendulum, as in the original PILCO simulation, it presents significant challenges when applied to the test rig used in this research.

The real-world setup features a longer pendulum length of 0.875 m and a more restricted range of motion along the x-axis. Under these conditions, the Euclidean distance-based loss function remains nearly constant and close to its maximum value throughout most of the pendulum's motion outside the desired target state, even when the pendulum is moving in a feasible trajectory toward stabilization. As a result, the loss function fails to provide the gradual transitions needed to distinguish between good and bad state changes, making it difficult for PILCO to learn an effective control policy. This is why in this research, a different approach to the loss function was taken.

4.4 Run 1—state space penalization

In this research, the loss function is designed to achieve two primary objectives. The primary goal is to enable the agent to swing up and balance the pendulum in the upright position as quickly as possible. Therefore, the loss function must be defined in a way that transitions from bad to good states and vice versa are not sudden, but rather that a gradual transition is recognizable to the agent. This is necessary for the agent to reach and maintain good states and achieve optimal control performance. The second goal is to ensure safety on the test, so that the agent does not exceed physical limits. In other words, the agent should never apply input actions that move the cart to the physical end plate of the test bench. For this reason, a combination of two loss functions was chosen for the first run/experiment of this research.

The first loss function is responsible for swinging and balancing the pendulum in the upright position. It is implemented as a saturating loss function that compares the current measured state variables with the target state variables [21]. More precisely, the greater the difference between the current state and the target state, the higher the loss value. Therefore, the value of the saturating loss cannot be higher than one. This is convenient because the saturating loss only determines whether a state is far away from the target and not specifically *how* far away [20]. The saturated loss is defined as

$$L_{\text{saturated}} = 1 - \exp\left(-\frac{1}{2}(x_t - x_{\text{target}})^\top \frac{\mathbf{W}_{\text{sat}}}{\sigma^2_{c,\text{sat}}}(x_t - x_{\text{target}})\right) \in [0, 1], \tag{11}$$

with x_t being the current and x_{target} being the target state space at time step t, W a weight matrix and σ_c the width of the loss [21]. The target state is characterized by the pendulum being in the upright position at $\theta = 0$, with the cart positioned at the center of the track, ideally without any movement or velocity of both the pendulum and cart. Therefore, the target state space is given by:

$$x_{\text{target}} = [0, 0, 0, 0, 0, 1]^T$$

To adapt the saturated loss specifically for the given problem, the diagonal entries of the weight matrix $\mathbf{W}_{\text{sat}}$ can be set. These describe the ratios by which the individual state variables are evaluated or taken into account by the saturated loss. If one entry is set to zero, the corresponding state variable is not punished, whereas setting it to one fully penalizes it. Setting the diagonal elements of $\mathbf{W}_{\text{sat}}$ can be challenging, as it is essential to penalize the state variables sufficiently to reach the target state, but not so heavily that the agent perceives no transitions to good states and avoids moving altogether. The agent must have sufficient motivation to explore the environment and make the necessary adjustments, while also being directed towards achieving the upright pendulum position.

In general, the state variable x of the cart's position should not be penalized too strongly since the agent needs to move the cart along the x-axis to swing up the pendulum. However, if not punished at all, the agent would try to swing up and balance the pendulum in an arbitrary position. Since it is desired to balance the pendulum in the middle of the x-axis, the state variable x of the position of the cart is set to a comparatively small value of 0.05. In theory, this causes no influence on the swing-up but ensures the pendulum moves to the center of the test bench once it is swung up. Both the cart velocity $\dot{x}$ and the angular velocity $\dot{\theta}$ are not penalized, as both values must fluctuate greatly to allow swinging up and balancing the pendulum. Since the angle of the pendulum θ adds up with several pendulum turns, it would not represent the loss correctly. Instead, the complex representations of the angle $sin(\theta)$ and $cos(\theta)$ are considered. The weight of the cosine state variable $cos(\theta)$ is set to 1. This is done because the cosine term clearly distinguishes between the pendulum's initial downward position ($\theta = \pi$) and the desired upright position ($\theta = 0$). The cosine function allows for a straightforward evaluation of whether the pendulum is moving toward or away from the upright position. A high weight on the cosine term ensures that the agent receives a substantial penalty when the pendulum is far from the upright position, encouraging it to push the pendulum towards balance. However, when using only the cosine term, the transitions from good to bad states are not sufficiently precise for the agent since the gradient of the cosine is relatively flat for angles close to zero. For instance, when the pendulum is swinging down after being upright, the cosine value could still give an acceptable loss, leading the agent to believe that the state is still acceptable. In reality, it has no chance of counteracting the fall. To address this issue, the sine term $sin(\theta)$ is incorporated with a weight of 0.5. The sine function has a stronger gradient around the zero angle, which helps capture the dynamic changes in the pendulum's motion more effectively. By assigning a moderate weight to the sine term, the agent can better anticipate transitions to poor states, allowing it to react in time and adjust its actions to prevent the pendulum from falling. If only the sine term is considered, the agent would face difficulty because the sine function alone does not provide clear feedback about the position of the pendulum, as both $sin(0)$ and $sin(\pi)$ result in 0. This could lead the agent to believe that the initial downward position is optimal, which is not desirable. By balancing the use of both sine and cosine terms, the agent can recognize the difference between upright and downward positions while also getting early feedback on transitions toward bad states. The overall weight matrix $\mathbf{W}_{\text{sat}}$ for the saturating loss is set to:

$$\mathbf{W}_{\text{sat}} = \begin{bmatrix} 0.05 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In contrast to the weight matrix $\mathbf{W}_{\text{sat}}$, which describes the ratios of the state variables among each other, the loss width $\sigma_{c,\text{sat}}$ defines how quickly the loss approaches its maximum value of 1 [21]. It must be set to ensure an even distribution of the loss function. Choosing a higher value, the saturated loss $L_{\text{saturated}}$ remains closer to 0 and the system is less sensitive to deviations. Setting it too low, the saturated loss $L_{\text{saturated}}$ is more likely to be close to 1, making it difficult for the

Table 3 List of used hyperparameters of the loss functions in the first run

Hyperparameter	Definition	Value
$\mathbf{W}_{\text{sat}}$	Weight matrix of saturated loss	diag(0.05, 0, 0, 0, 0, 0.5, 1)
$\sigma_{c,\text{sat}}$	Loss width of the saturated loss	0.6
a	Slope of the double-hinge loss	500
b_1, b_2	Corner points of the double-hinge loss	$[-0.42, 0.42]$ m

PILCO to learn transitions to good states. The loss width of the saturating loss was determined to $\sigma_{c,\text{sat}} = 0.6$.

The second component of the loss ensures that the state constraints are not exceeded, more precisely that the agent does not move the cart into the end plates of the test bench. Adapted from Hesse et al., it is implemented as a double-hinge function

$$L_{\text{double-hinge}} = \max(0, -a(x - b_1), a(x - b_2)) = \begin{cases} -a(x - b_1) & \text{for } x < b_1 \\ 0 & \text{for } b_1 \le x \le b_2 \\ a(x - b_2) & \text{for } x > b_2 \end{cases}, \quad \in [0, \infty) \tag{12}$$

which only depends on the state variable of the cart position x with slope a and corner points b_1 and b_2 [4]. The loss is zero as long as the cart is between the corner points. After passing the corner points, the loss increases linearly with the slope a.

Setting adequate values for the corner points is challenging, as they must balance both safety and performance within the system. If the corner points are set too close to each other, it increases the *buffer zone* by keeping the cart far from the test bench limits. This minimizes the risk of collisions with the end plates, making the control system more robust. However, a larger *buffer zone* comes at the cost of reducing the usable track length for the cart's motion, resulting in a slower or less efficient swing-up. Since the x-axis is relatively short with 520 mm to each side, the corner points are chosen to $b_1 = -420$ mm and $b_2 = 420$ mm. This results in a *buffer zone* of 100 mm for each side. Therefore, to penalize a violation of exceeding the corner points strongly enough, the slope a is set relatively high to $a = 500$.

To summarize, the overall loss function of the first experimental run results in

$$L_{\text{run1}} = L_{\text{saturated}} + L_{\text{double-hinge}}.$$

with its corresponding hyperparameters shown in Table 3.

The following presents the results of the first experimental run. Figure 5 shows the first rollout of the first run, in which the offline data were used to train the policy. On the left subplot, the function value ΣL_{pred} of the prediction is plotted over the number of optimization iterations $opt.length$. In the right subplot, the predicted immediate loss trajectory, L_{pred}, is plotted along with its mean and uncertainty over the number of prediction steps, H. It describes the predicted behavior of the loss when the policy is applied. The actual immediate loss trajectory, L_{real}, of the applied policy is also plotted. It can be seen that the function value ΣL_{pred} of the predicted loss decreases significantly from about $\Sigma L_{\text{pred}} = 7.6 \times 10^4$ to $\Sigma L_{\text{pred}} = 60.0323$ during the policy optimization. Furthermore, when looking at the predicted immediate loss L_{pred}, it is noticeable that the loss behavior is only realistically estimated up to the prediction step $H = 35$. Afterward, the predicted immediate loss L_{pred} runs in a straight line with high uncertainty. Although the dynamic model is directly able to capture an upward swing of the pendulum through the offline data used, which can be recognized by the wave-shaped trajectory of the predicted loss L_{pred}, the actual immediate loss L_{real} deviates strongly from it. The exact total loss of $\Sigma L_{\text{real}} = 75.8403$ is well above the function value ΣL_{pred} of the predicted loss.

Figure 6 displays the state variables and the input action u of the applied policy of the first rollout. In general, an oscillating behavior of all state variables is observed. The cart moves continuously between the two corner points $b_{1,2} = [-0.42, 0.42]$ m, whereby the cart velocity $\dot{x}$ also remains in a constant range of $\dot{x} = [-1.5, 1.5]$ m/s. The angle θ and the angular velocity $\dot{\theta}$ increase slightly during the rollout, indicating an upswinging of the pendulum; however, the pendulum does not reach the target point in any step. The action input u of the electric motor fluctuates within the given range of $u_{\max} = 3500$ rpm.

In the first run, the PILCO algorithm learns to swing-up and balance the pendulum after the tenth rollout. The duration of model and policy learning ranged from 10 min in the first rollout to around 90 min in the tenth rollout. Overall, the policy demonstrated incremental improvement with each rollout; however, in rollout four, the policy pushed the cart beyond the physical limits of the test bench. When moving into the physical stop, the action output automatically switches to zero via an emergency stop, and the carriage comes to a complete stop. In this case, the policy of the PILCO algorithm still computes the input actions u based on the given state variables; however, the cart has not moved since the emergency stop has been activated. Providing this entire dataset to the next learning iteration would falsify fur-

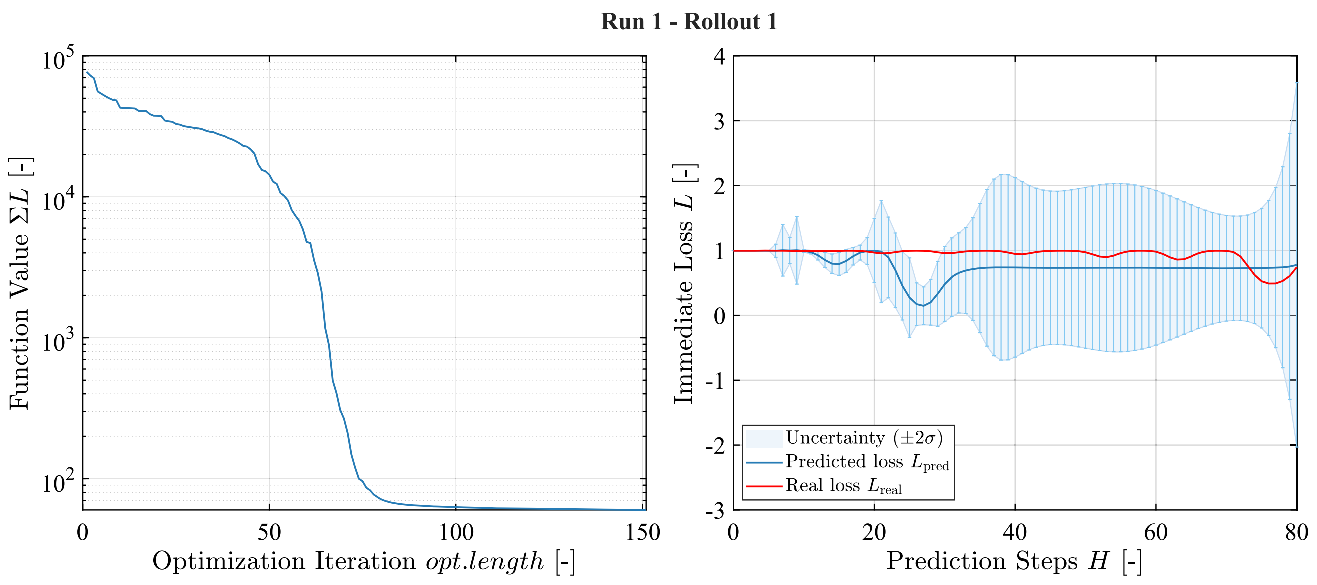

Fig. 5 Function value and immediate loss of the first rollout of run 1

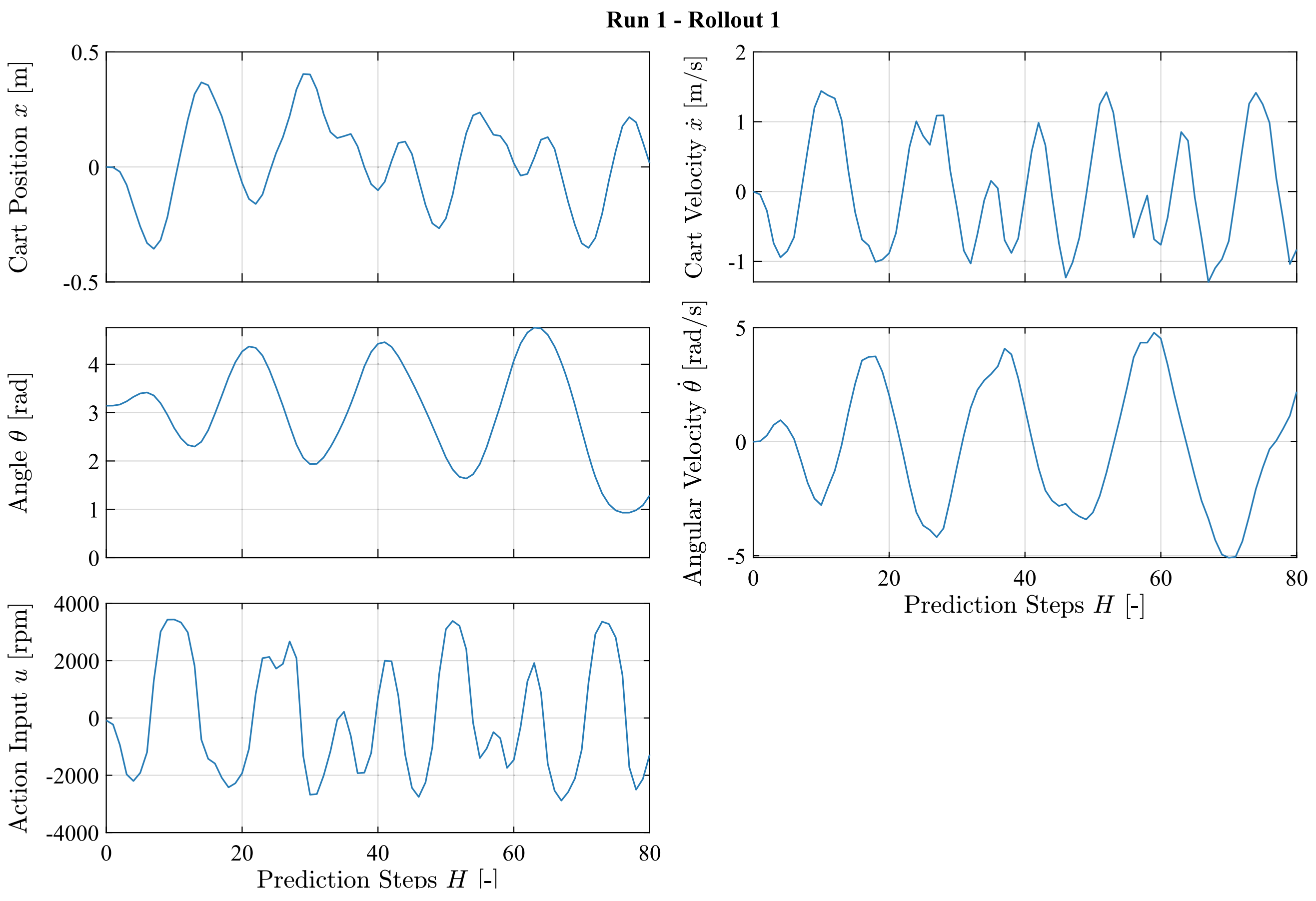

Fig. 6 State variables of the applied policy of the first rollout of run 1

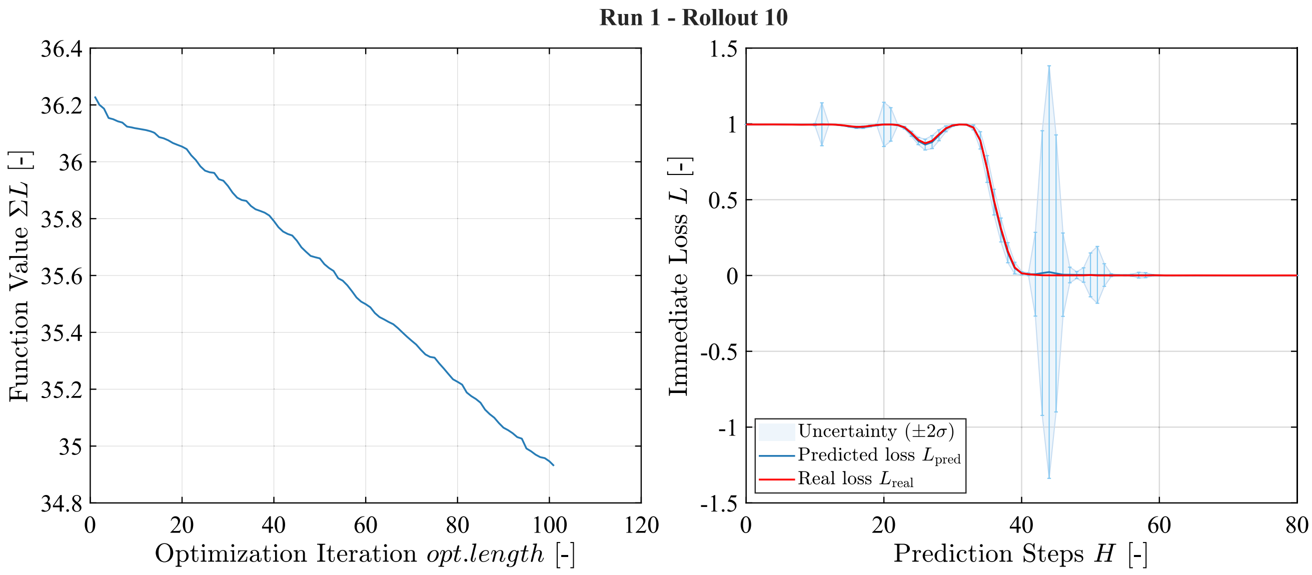

Fig. 7 Function value and immediate loss of the tenth rollout of run 1

ther model learning, as the PILCO algorithm perceives an active input action, but the actual cart movement is fixed. This is why the measured data from the rollout have to be shortened from the prediction step H in which the cart moves to the physical limit. For this reason, moving to the physical limit has two adverse effects: It not only poses potential risks in the real-world environment but also reduces the amount of data passed for model learning, thereby increasing learning time and, consequently, the interaction with the environment. A similar behavior occurred in the fifth, sixth, and seventh rollouts of the first run. In these three further rollouts, the cart reaches its physical limit. This does not happen during the swing-up, but always in the attempt to keep the pendulum in the upper state. In each rollout, the data for the next model and policy learning are cut from the point where the cart reaches the physical limit of the test bench.

Figure 7 clearly shows how the predicted immediate loss in the tenth rollout L_{pred} predicts a balancing of the pendulum from prediction step $H = 42$. Furthermore, the uncertainty over the entire rollout is very low and only increases at the critical moment of transition from swinging up to balancing. The predicted immediate loss trajectory aligns very well with the actual loss trajectory. The function value exhibits a relatively linear progression over the optimization iterations $opt.length$, with a continuing downward trend even after 100 iterations are performed. However, the absolute improvement of the function value is minimal, with a final value of $\Sigma L_{\mathrm{pred}} = 34.9305$. The total loss is $\Sigma L_{\mathrm{real}} = 35.9387$.

In Fig. 8 of the state variables of the tenth rollout, both the swing-up process and the pendulum balancing can be seen very clearly. The angular velocity reaches its highest value of $\dot{\theta} = 6.5$ rad/s. The cart moves alternately to the left and right of the track without passing the corner points $b_{1,2}$ and with a cart velocity of $\dot{x} = [- - 1.5, 1.5]$ m/s. During the balancing, it moves slightly to the left and right, but not entirely around the zero point of the x-axis. Furthermore, the cart maintains a relatively high velocity of $\dot{x} = [-0.5, 0.5]$ m/s during the balancing movement. Once the angle θ reaches its target value, it remains the same with only small deviations over the rollout. The angular velocity ranges between $\dot{\theta} = [-1, 1]$ rad/s during balancing. During the transition from swing-up to balancing, a spike in the $u = 2000$ rpm action output is necessary to stabilize the pendulum. Furthermore, during balancing, the policy requires a rapid change in rotational speed from $u = [-1000, 1000]$ rpm to keep the pendulum upright.

Table 4 summarizes the function values of the predicted loss ΣL_{pred} as well as the actual total loss ΣL_{real} of all rollouts of the first run. Notably, both values in the early rollouts are further apart, particularly in those where the cart moves into its physical limit. During the later rollouts, the values match more closely. In addition, a downward trend in both values over all rollouts can be seen, except for the actual total loss ΣL_{real} in the rollouts where the physical limit is exceeded. Overall, the tenth rollout has the lowest values in both metrics.

In Table 5, the evaluation metrics of the first run are presented. The learned policy can swing up and balance the pendulum in $H = 42$ prediction steps, respectively, in 4.2 seconds. An interaction time of 80 seconds with the real-world environment was necessary to learn the policy. The time-weighted absolute error between the current and target angles is $ITAE = 15.7446$ s·rad. In total, the policy moved the cart 4 times into the physical limit of the test bench.

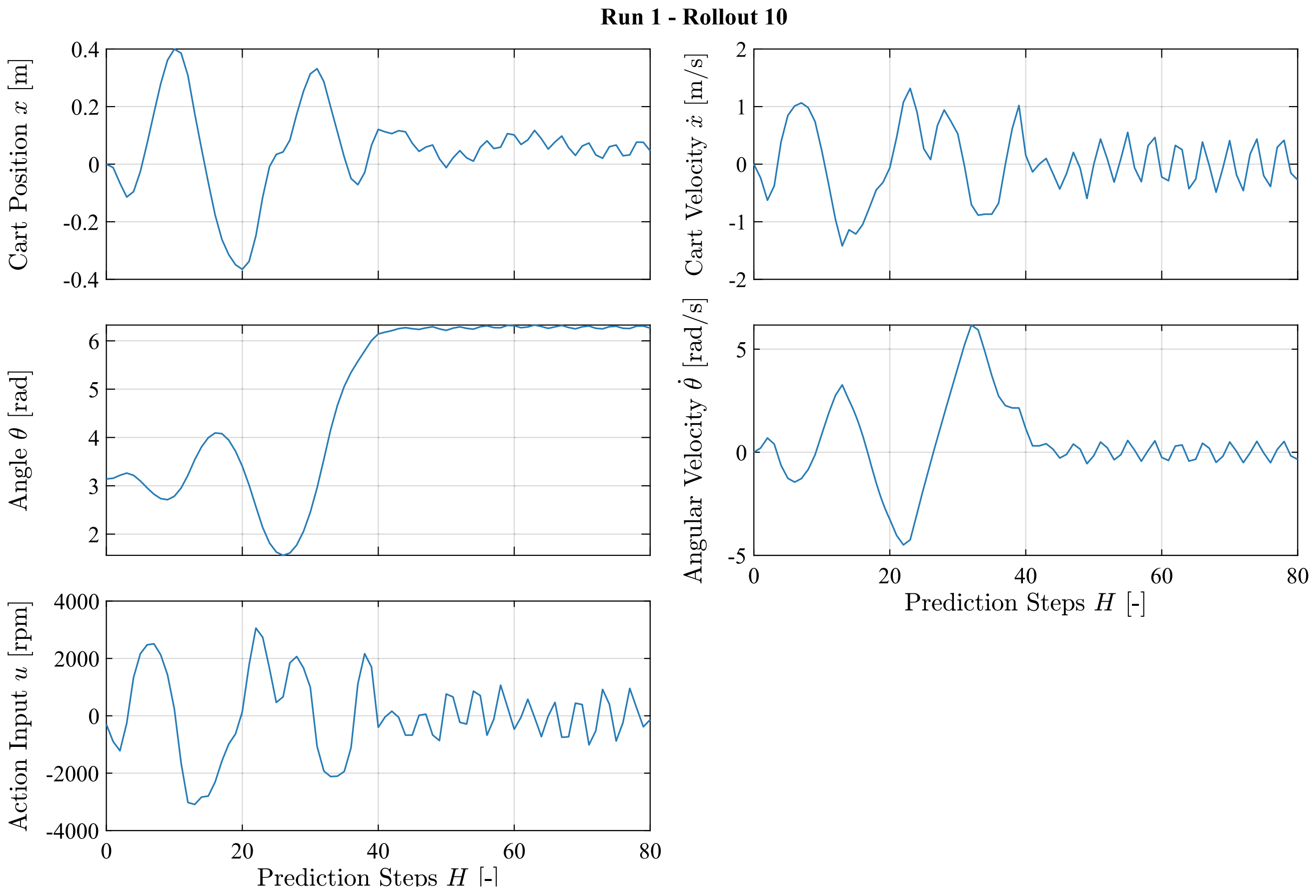

Fig. 8 State variables of the applied policy of the tenth rollout of run 1

Table 4 Function value ΣL_{pred} and real total loss ΣL_{real} of run 1

Rollout	Function value ΣL_{pred}	Real total loss ΣL_{real}
1	60.0323	75.8403
2	61.9303	63.1940
3	60.4830	54.4629
4	58.8970	120.4815
5	56.5022	191.3463
6	53.1686	151.4270
7	48.3419	134.4675
8	48.3418	51.0010
9	36.1816	37.3714
10	34.9305	35.9387

Table 5 Evaluation metrics of the final learned policy of run 1

Metric	Value
Time until balancing	4.2 s
Interaction time	80 s
ITAE	15.7446 s·rad
Constraint violations	4
Quality of balancing	588.9017 rpm

Table 6 Learned noise standard deviations of the final learned policy of run 1

State variable	σ_{noise}
x	0.004539
$\dot{x}$	0.067496
θ	0.0062155
$\dot{\theta}$	0.053278

Lastly, the quality of balancing, as measured by the standard deviation of the input action u, results in 588.9017 rpm.

Lastly, the learned noise standard deviations σ_{noise} of the state variables of the tenth rollout of the first run are presented in Table 6. It can be seen that the learned noise standard deviation of the two position state variables with $\sigma_{\mathrm{noise},x} = 0.004539$ and $\sigma_{\mathrm{noise},\theta} = 0.0062155$ lies in a similar order of magnitude. The two velocity state variables also have similar values to each other. However, they have overall higher learned noise standard deviations with $\sigma_{\mathrm{noise},\dot{x}} = 0.067496$ and $\sigma_{\mathrm{noise},\dot{\theta}} = 0.053278$ compared to the position state variables.

4.5 Run 2—energy consideration

In the first run, a relatively simple loss function composition was chosen, which mainly penalizes the angle of the pendulum and the crossing of the corner points $b_{1,2}$. When looking at the state variables of the applied optimal policy π^* of the tenth rollout in Fig. 8, it can be seen that the pendulum angle θ remains constant after the swing-up with very low deviations. This results from the specified loss function, which primarily penalizes the angle θ of the pendulum in the first run, leading the policy to optimize for this behavior. However, this also means that other behaviors are left unpunished, allowing the policy to act more freely in those areas. Consequently, the optimal policy π^* from the first run achieves a fast swing-up within 4.2 seconds. Still, it behaves abruptly with decisive alternating control input actions u, potentially damaging the motor and test bench. Further rollouts and optimization iterations will not improve this behavior, as the immediate loss L is already zero during the balancing due to the achieved target angle. Furthermore, it can be seen that the cart crossed the corner points $b_{1,2}$ as well as the physical limit of the test bench multiple times in rollouts four to seven of the first run. This never happened during the swing-up process, but always occurred when stabilizing the pendulum in its upper position. Although the double-hinge loss helped to keep the cart within a specific section of the track during the swing-up, it was not sufficient to entirely prevent the cart from reaching the physical limits of the test bench during balancing.

For these reasons, the loss function of the second experimental run was expanded by a third component. The third component of the loss structure aims to ensure a smooth swing-up and balancing by minimizing the total energy in the system. This makes it easier for the PILCO to control the pendulum successfully, as rapid movements favor a possible overshooting of the pendulum and also, in general, create more noise in the system. Furthermore, it reduces the amount of electrical energy consumed. An approach was chosen that considers the total energy in the cart-pole system. The sum of the kinetic and potential energy gives the total energy:

$$E_{\text{total}} = E_{\text{kinetic}} + E_{\text{potential}} \tag{13}$$

The total kinetic energy $E_{kinetic}$ is composed of the translational kinetic energy of the cart mass M and the translational and rotational kinetic energy of the mass m of the pendulum tip [23]:

$$E_{\text{kinetic}} = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m\left(v_x^2 + v_y^2\right) \tag{14}$$

Due to the small diameter, the mass of the pendulum pole is neglected. The velocities v_x and v_y of the pendulum tip can be obtained by deriving the displacement coordinates of the pendulum tip. Figure 9 shows the cart-pole system with its coordinates.

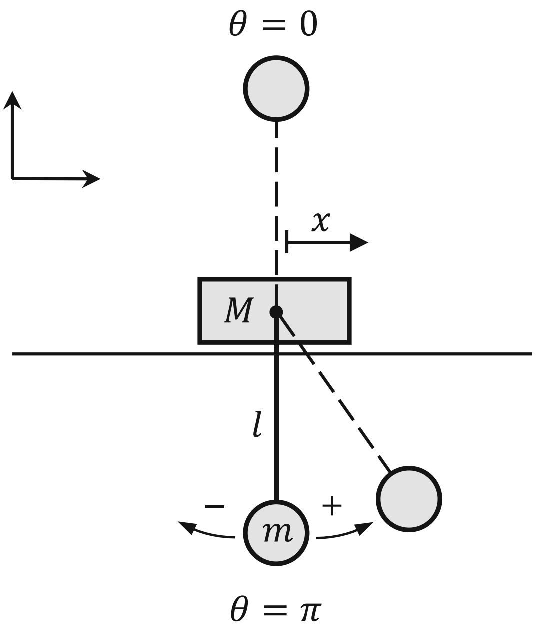

Fig. 9 Schematic of the cart-pole system

Based on the definition of the angle $\theta = \pi$ in the hanging state of the pendulum, the displacement coordinates of the pendulum tip in the horizontal and vertical direction result in

$$x_p = x - lsin(\theta),$$
$$y_p = lcos(\theta).$$

By derivation, the velocities of the pendulum tip result in

$$\dot{x}_p = \dot{x} - \dot{\theta}lcos(\theta),$$
$$\dot{y}_p = -\dot{\theta}lsin(\theta).$$

By substituting the velocities into Equation 14, the total kinetic energy is obtained:

$$E_{\text{kinetic}} = \frac{1}{2}(M+m)\dot{x}^2 - ml\dot{x}\dot{\theta}\cos(\theta) + \frac{1}{2}ml^2\dot{\theta}^2.$$

The potential energy of the pendulum tip is given by

$$E_{\text{potential}} = mgl(cos(\theta) + 1),$$

where the potential energy is at maximum in the upright position and at zero when it is hanging down. Thus, the total energy in the cart-pole system by Equation 13 results in:

$$E_{\text{total}} = \frac{1}{2}(M+m)\dot{x}^2 - ml\dot{x}\dot{\theta}\cos(\theta) + \frac{1}{2}ml^2\dot{\theta}^2 + mgl(cos(\theta)+1). \tag{15}$$

This expression considers the total energy in the system, respectively, the conversion of kinetic and potential energy. While it accurately measures the system's physical state, using the total energy directly as a loss function for control design can be challenging. This is due to the strong nonlinearity of the total energy, where small changes in the state variables can cause large, non-intuitive changes in the energy, leading to unpredictable and potentially unstable control actions. Additionally, the loss function must be differentiable with respect to the Gaussian input distribution for policy optimization. However, the fact that the loss function contains three state variables in one term makes computing the derivatives quite complex. Furthermore, when using the real total energy in the system, it would be necessary to maximize the potential energy to guide the agent to the desired target, since the upper state of the pendulum assumes the maximum energy in the system. This makes using the system's real total energy unsuitable for the control task with the PILCO algorithm. Instead, this work approximates the system's total energy with the help of the quadratic loss function. Since the quadratic loss is easier to differentiate with respect to the mean and covariance of the state variables, it provides a more practical and stable approach to guide the system to the desired target state. The quadratic loss implemented by Deisenroth is defined as [21]:

$$L_{\text{quadratic}} = (x_t - x_{\text{target}})^\top \frac{\mathbf{W}_{\text{quad}}}{\sigma^2_{c,\text{quad}}}(x_t - x_{\text{target}}). \tag{16}$$

With the given target state space x_{target} and the weight matrix $\mathbf{W}_{\text{quad}}$ set to

$$\mathbf{W}_{\text{quad}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{w}\frac{1}{2}(M+m) & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2}ml^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & mgl \end{bmatrix},$$

the quadratic loss $L_{\text{quadratic}}$ results in:

$$L_{\text{quadratic}} = \frac{1}{w}\frac{1}{2}(M+m)\dot{x}^2 + \frac{1}{2}ml^2\dot{\theta}^2 + mgl(cos(\theta)-1)^2. \tag{17}$$

By comparing the quadratic loss with the total energy in the cart-pole system in Equation 15, it is noticeable that the cross-term of the linear and rotational movement of the pendulum is no longer present. Additionally, the term of the potential energy has changed. This is because the loss function is minimized regarding the target state, defined as the pendulum pointing upwards with the angle $\theta = 0$. The expression $(cos(\theta) - 1)$ indicates that the optimized minimum energy state corresponds to the upright position. Although this is not physically accurate, it enables the PILCO algorithm to achieve its control objective. Instead of fully modeling the system's exact kinetic and potential energy, the focus is shifted to penalizing deviations from the desired energy state. This approach simplifies the key dynamics, such as the interaction between the cart's velocity and the pendulum's angular velocity, making it more intuitive for control purposes. Furthermore, the influence of kinetic energy is reduced in comparison to the other terms by the weight factor w, as this would have a much stronger influence on the energy reduction than the other terms due to the high cart mass M. Using this quadratic loss approximation, simplified physical relationships are still considered, while ensuring a smooth and controlled swing-up, which leads to more stable learning toward the desired target state.

The choice of suitable parameters for the loss width $\sigma_{c,\text{quad}}$ and weight factor w requires careful consideration. The loss width $\sigma_{c,\text{quad}}$ is selected to ensure that the quadratic loss $L_{\text{quadratic}}$ is within the same range as the saturated loss $L_{\text{saturated}}$. Furthermore, the weight factor w has to be chosen so that the cart's kinetic energy is not disproportionately weighted compared to that of the pendulum. If the kinetic energy is weighted too vigorously, the cart might hardly move due to the high energy loss. In this work, the loss width is set to $\sigma_{c,\text{quad}} = 20$ and the weight factor is set to $w = 10$.

To summarize, the overall loss function of the second experimental run results in

$$L_{\text{run2}} = L_{\text{saturated}} + L_{\text{double-hinge}} + L_{\text{quadratic}}.$$

with its corresponding hyperparameters shown in Table 7.

The following presents the results of the first experimental run. Figure 10 shows the function value ΣL_{pred} and the immediate loss L of the first rollout of the second run. The function value drops significantly during the first optimization iterations, resulting in $\Sigma L_{\text{pred}} = 67.8797$ after $opt.length = 150$. The predicted immediate loss, L_{pred}, exhibits similar behavior to the first rollout of the first run. With the help of the offline data, the algorithm can recognize the pattern of a swing-up; however, after the prediction step $H = 35$, the mean of the predicted immediate loss L_{pred} remains almost unchanged, accompanied by high uncertainty. The real immediate loss, L_{real}, exhibits a good swinging behavior over the rollout, but the pendulum never moves close to the target angle. The cart remains between the corner points $b_{1,2}$ at all times. Furthermore, it is noticeable that the actual immediate loss, L_{real}, deviates significantly from the predicted trajectory, even in areas with low uncertainty. The total loss is $\Sigma L_{\text{real}} = 80.3347$.

The state variables of the first rollout of run two are displayed in Fig. 11. The plot of the angle θ indicates an upward swinging of the pendulum, but the target angle is again not

Table 7 List of used hyperparameters of the loss functions in the first run

Hyperparameter	Definition	Value
$\mathbf{W}_{\text{sat}}$	Weight matrix of saturated loss	diag(0.05, 0, 0, 0, 0.5, 1)
$\mathbf{W}_{\text{quad}}$	Weight matrix of quadratic loss	$\text{diag}(0, \frac{1}{w}\frac{1}{2}(M+m), \frac{1}{2}ml^2, 0, 0, mgl)$
$\sigma_{c,\text{sat}}$	Loss width of the saturated loss	0.6
$\sigma_{c,\text{quad}}$	Loss width of the quadratic loss	20
w	Weight factor of the quadratic loss	10
a	Slope of the double-hinge loss	500
b_1, b_2	Corner points of the double-hinge loss	$[-0.42, 0.42]$ m

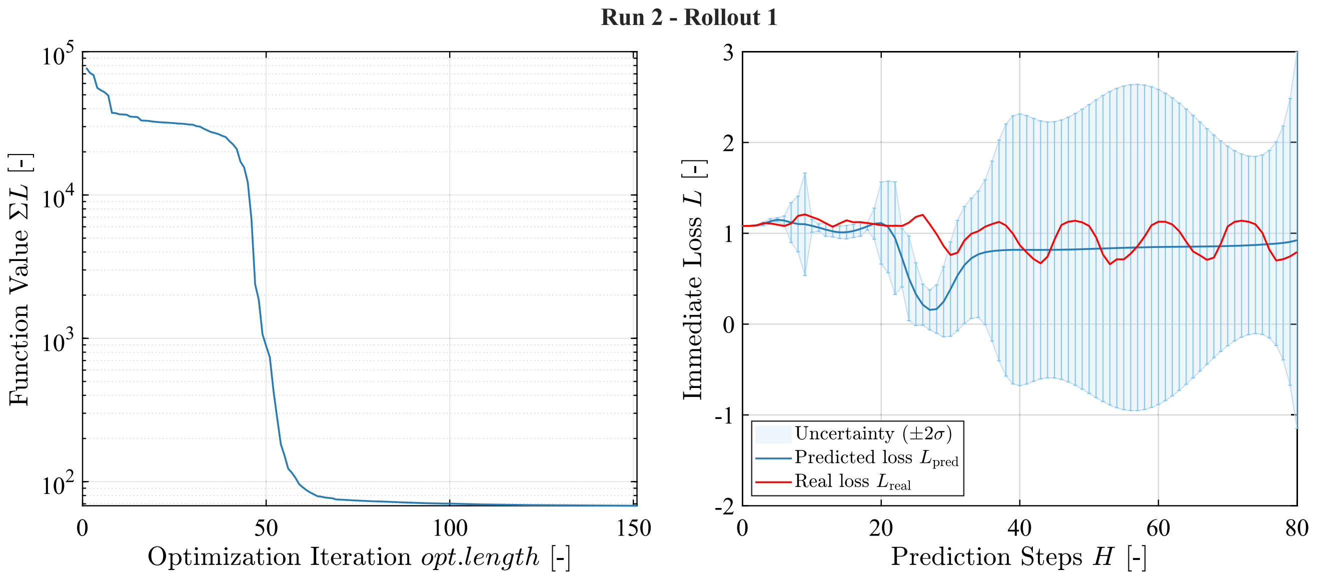

Fig. 10 Function value and immediate loss of the first rollout of run 2

reached in the first rollout. The cart moves between the corner points $b_{1,2}$ with a cart velocity of $\dot{x} = [-1.2, 1.5]$ m/s. The angular velocity reaches values between $\dot{\theta} = [-5, 5]$ rad/s, whereas the input action u ranges between $u = [-2500, 3200]$ rpm.

A successful swing-up and balancing of the pendulum over the planning horizon $T = 8$ s were first learned in rollout seven. However, when applying this trained policy for validation over an extended period, it exhibited unstable behavior in balancing the pendulum. This is why another rollout was performed, which could balance the pendulum over an infinite period. Figure 12 shows the function value ΣL_{pred} and immediate loss L of the final eighth rollout. The function value improves slightly to $\Sigma L_{\text{pred}} = 52.0526$ with an ongoing linear downward trend. The predicted immediate loss estimates a $L_{\text{pred}} = 0$ from prediction step $H = 60$ onward. Overall, the uncertainty of the predicted immediate loss, L_{pred}, is very low, except during the transition from the swing-up process to balancing. The actual immediate loss L_{real} matches the predicted loss trajectory well and results in a total actual loss of $\Sigma L_{\text{real}} = 53.3960$. Similar to the first run, the duration time of model and policy learning varied from 10 min in the first rollout to 60 min in the eighth rollout of the second run.

In Fig. 13 of the state variables of the eighth rollout of run two, both the upswing and the balancing of the pendulum are recognizable. The angle θ first moves alternately before reaching the target value at prediction step $H = 57$. After that, it remains constant over the entire rollout. This behavior is also visible in the angular velocity $\dot{\theta}$, which remains around zero after the swing-up. The swing-up process can be seen in the subplot of the cart position x. The cart moves between $x = [-0.3, 0.4]$ m. It is noticeable that the cart moves toward the left corner point b_2 during the transition from swing-up to balancing. It then continuously moves toward the center after the pendulum has been swung up. The cart velocity reaches values between $\dot{x} = [-1, 1.5]$ m/s during the swing-up and lowers significantly during the balancing process. This behavior can also be seen in the action output u. Whereas high alternating action inputs $u = [-2200, 3100]$ were set from the policy at the beginning, the input actions u remain very low after the swing-up process. During the transition from

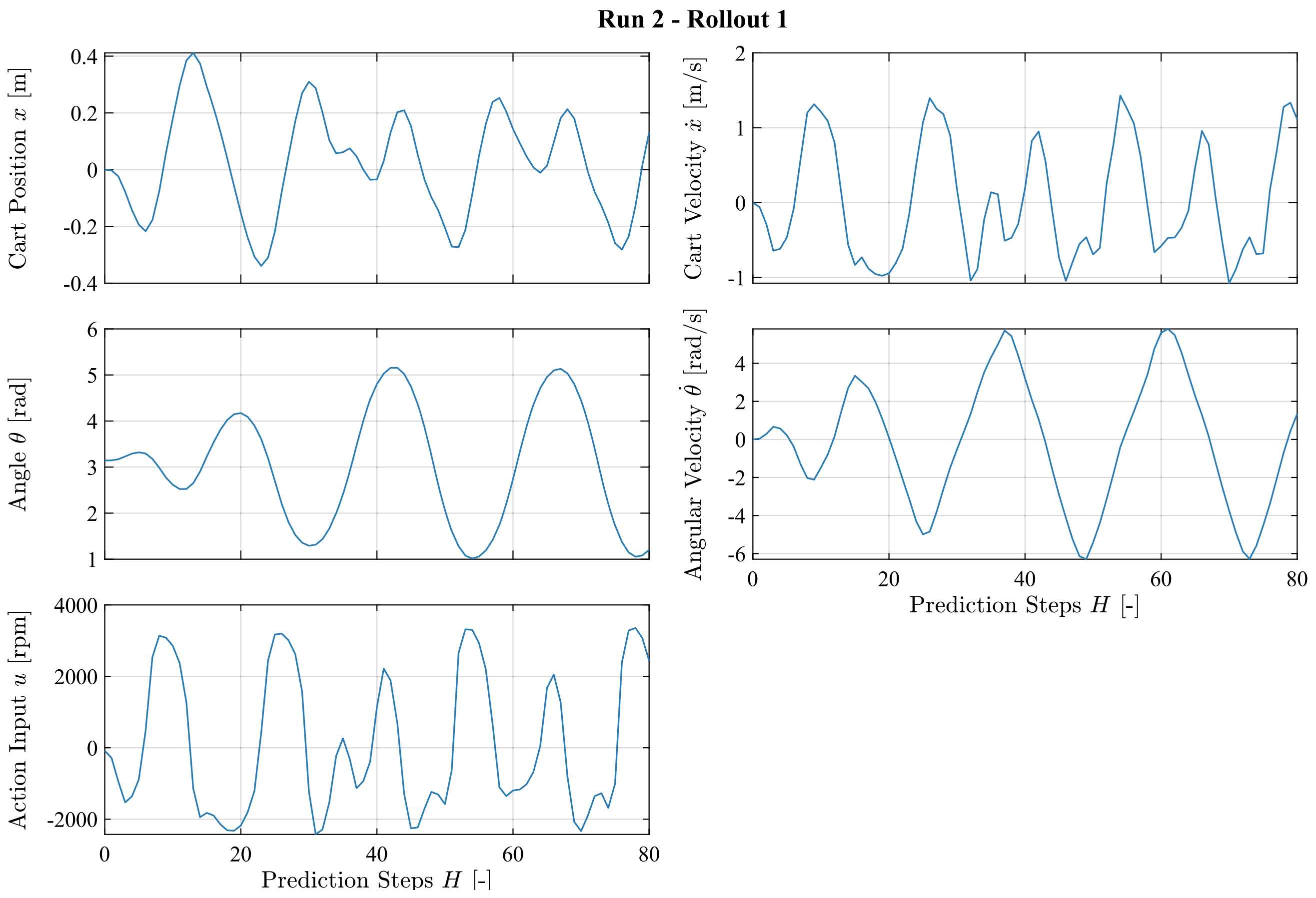

Fig. 11 State variables of the applied policy of the first rollout of run 2

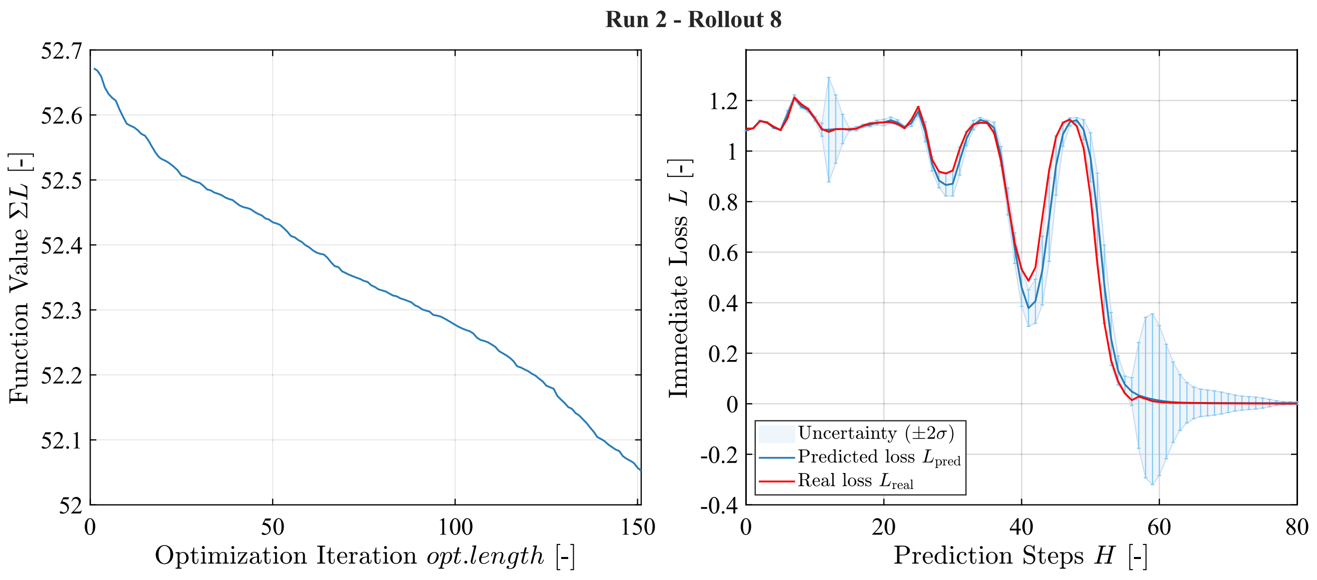

Fig. 12 Function value and immediate loss of the eighth rollout of run 2

Table 8 Function value ΣL_{pred} and real total loss ΣL_{real} of run 2

Rollout	Function value ΣL_{pred}	real total loss ΣL_{real}
1	67.8797	80.3347
2	67.0344	63.2509
3	61.6460	62.6369
4	61.2624	61.1813
5	59.3122	55.5645
6	53.8033	63.1657
7	52.6347	53.6424
8	52.0526	53.3960

Table 9 Evaluation metrics of the final learned policy of run 2

Metric	Value
Time until balancing	5.7 s
Interaction time	64 s
ITAE	27.7551 s·rad
Constraint violations	0
Quality of balancing	75.6069 rpm

swing-up to balancing, an action output of around $u = 1000$ rpm at prediction step $H = 58$ is performed to stabilize the pendulum.

In Table 8, both the function values of the predicted loss ΣL_{pred} and the actual total loss ΣL_{real} of all rollouts of the second run are summarized. Both values decrease with the number of rollouts, except the actual total loss ΣL_{real} of rollout six. This is because the policy of rollout six predicted a balancing of the pendulum, whereas the applied policy failed to do so. Compared to the first run, the values of both metrics are generally higher, as long as the increase in the actual total loss ΣL_{real} due to the double-hinge loss in run one is not considered. Again, both values match more closely in the last two rollouts. Also, the eighth rollout has the lowest values in both metrics overall.

Table 9 shows the evaluation metrics of the final rollout of the second run. Compared to the first run, with 5.7 seconds, the policy takes longer to bring the pendulum upright. However, it only needed 64 seconds of interaction time with the real-world environment, which is 16 seconds less than the first run. The ITAE results in 27.7551 s·rad, implying a higher deviation from the target angle. Unlike the first run, the cart always moves within the physical limits of the test bench. The quality of balancing, as measured by the standard deviation of the action input after the pendulum is swung up, is 75.6096 rpm, and therefore significantly lower than in the first run.

The learned noise standard deviations σ_{noise} of the state variables of the final rollout of the second run are presented in Table 10. Overall, all learned noise values are

Table 10 Learned noise standard deviations of the final learned policy of run 2

State variable	σ_{noise}
x	0.0025373
$\dot{x}$	0.036417
θ	0.0045536
$\dot{\theta}$	0.046904

lower than those in the first run. Again, the learned noise standard deviation of the two position state variables with $\sigma_{\text{noise},x} = 0.0025373$ and $\sigma_{\text{noise},\theta} = 0.0045536$ lies in a similar range. Compared to that, the two velocity state variables have a higher learned noise standard deviation with $\sigma_{\text{noise},\dot{x}} = 0.036417$ and $\sigma_{\text{noise},\dot{\theta}} = 0.046904$.

5 Discussion

Firstly, the PILCO algorithm successfully learned a policy that could swing up and balance the hydraulic pendulum upright in both test runs. However, due to the varying parameterization of the loss functions, both runs exhibit different characteristics in their control strategies.

In the first run, a relatively simple loss function composition was chosen, which mainly penalizes the angle of the pendulum and the crossing of the corner points $b_{1,2}$. When looking at the state variables of the applied optimal policy π^* of the tenth rollout in Fig. 8, it can be seen that the pendulum angle θ remains constant after the swing-up with very low deviations. This results from the specified loss function, which primarily penalizes the angle θ of the pendulum in the first run, leading the policy to optimize for this behavior. However, this also means that other behaviors are left unpunished, allowing the policy to act more freely in those areas. Consequently, the optimal policy π^* from the first run achieves a fast swing-up within 4.2 seconds. Still, it behaves abruptly with decisive alternating control input actions u, potentially damaging the motor and test bench. Further rollouts and optimization iterations will not improve this behavior, as the immediate loss L is already zero during the balancing due to the achieved target angle. Furthermore, it can be seen that the cart crossed the corner points $b_{1,2}$ as well as the physical limit of the test bench multiple times in rollouts four to seven of the first run. This never happened during the swing-up process, but always occurred when stabilizing the pendulum in its upper position. Although the double-hinge loss helped to keep the cart within a specific section of the track during the swing-up, it was not sufficient to entirely prevent the cart from reaching the physical limits of the test bench during balancing.

For these reasons, the loss function was expanded in the second run. By adding the quadratic loss function, the energy

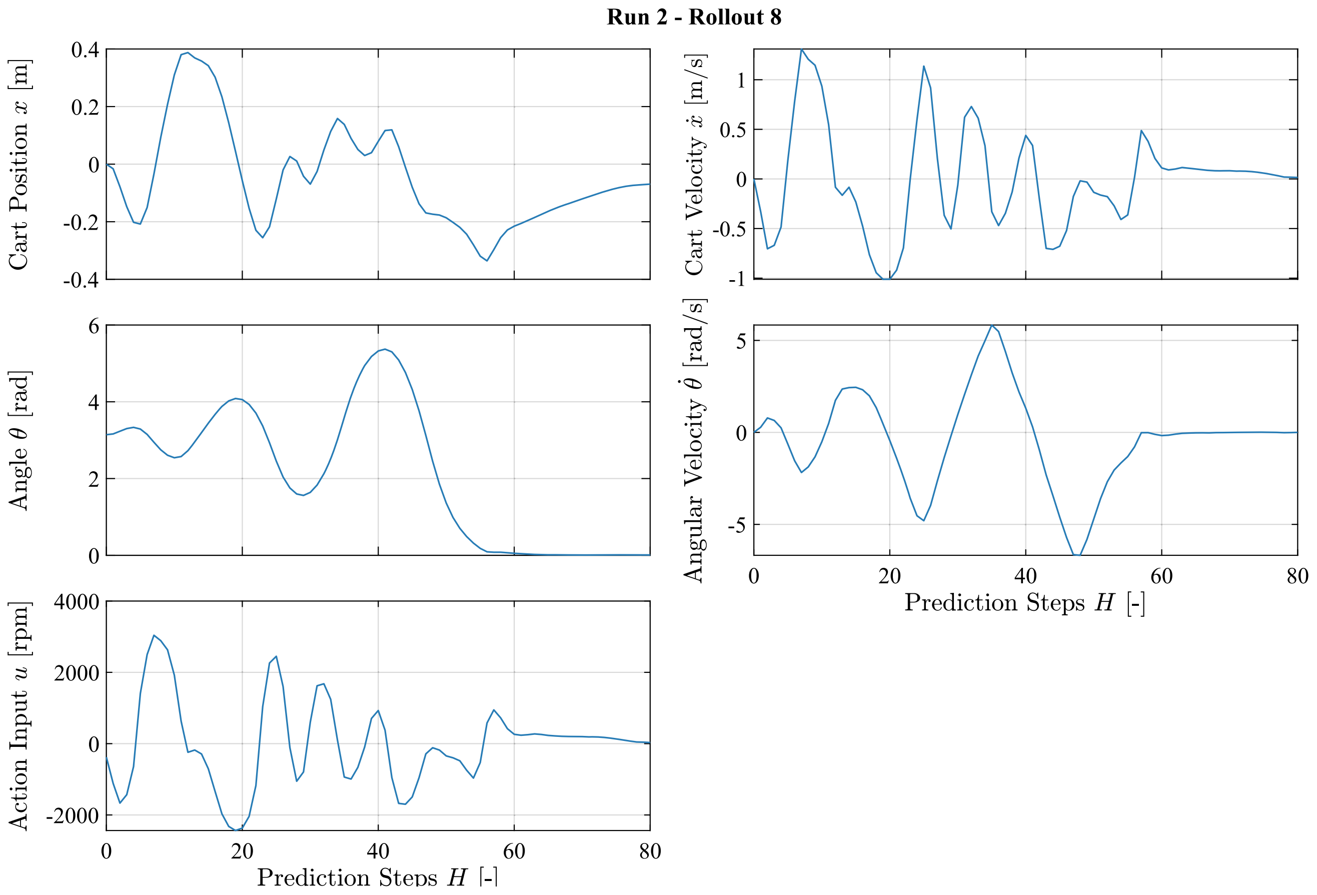

Fig. 13 State variables of the applied policy of the eighth rollout of run 2

in the system is penalized additionally. The positive influence of the quadratic loss is evident from the results of the second run. At no point did the cart move beyond the corner points $b_{1,2}$ or the physical limit of the test bench. Additionally, the algorithm was able to model the behavior of the loss over the prediction steps H more quickly, requiring a total interaction time of only 64 seconds to learn the optimal policy π^*, compared to the first run, which took 80 seconds. However, this may also be because, unlike in the first run, none of the datasets needed to be reduced due to the cart exceeding the physical limit of the test bench. Overall, the behavior of the optimal policy π^* of the second run is steadier and smoother, leading to a more gentle swing-up and balancing process. This can be seen in the metric quality of balancing. The standard deviation of the input action u during balancing is significantly lower at 75.6069 rpm than in the first run. Furthermore, the input action u spike during the transition from swing-up to balancing with $u = 1000$ rpm is only half as substantial as the first run. This indicates that the pendulum is moved more gently to the upright position, with less need for strong countersteering of the cart. The learned noise standard deviations σ_{noise} further confirm this behavior since all four state variables in the second run showed lower learned noise values. This noise reduction is likely due to the decreased dynamics in the system, as smoother movements reduce measurement variability and improve data quality. In summary, the additional penalization of system energy led to a smoother policy, especially during balancing. However, it also resulted in a slower overall swing-up time of 5.7 seconds. The ITAE with 27.7551 s·rad is also higher than in the first run due to the more extended swing-up period of the policy.

Furthermore, it can be observed that swinging up the pendulum was a much easier task for PILCO, while learning to balance the pendulum required considerably more optimization. This is partly because the offline dataset included only the pendulum swing-up, without any data on balancing. Additionally, balancing the pendulum, especially the transition phase, is challenging for the algorithm to learn. On one hand, stabilizing the pendulum requires quick and precise movements of the cart, which is a complex control task. On the other hand, this balancing phase occurs very late in the rollout, meaning that the propagated uncertainty in the system becomes relatively high at later prediction steps, making accurate prediction more difficult. Additionally, because the transition to balancing happens quickly, far fewer data points are available to PILCO for this phase compared to the swing-up process.

Overall, the behavior observed in both runs with differently defined loss functions demonstrates that reward shaping has a substantial impact on the learning process and the resulting policy. To develop an effective policy, reward shaping must be carefully designed to guide the learning process precisely. In this case, the loss function establishes the boundaries within which PILCO can explore freely, continuously searching for an optimal path. However, since the algorithm iteratively optimizes itself based on data obtained from previous policies, the resulting policy may vary significantly and can be somewhat unpredictable. If the loss function is not well-defined, the resulting policy can show fluctuations or appear random, particularly in the absence of offline data. Once a policy has been learned, the behavior remains fixed, limited by the constraints initially set by the loss function. This highlights the critical role of reward shaping in ensuring consistent and reliable policy performance.

5.1 Responsiveness and adaptability

When using the optimal policies π^* of both runs over a longer time and considering the influence of disturbing forces, respectively, manual deflections of the pendulum, the optimal policy of the first run proved more responsive to external influences. This is because the policy behaves more dynamically and can quickly stabilize the pendulum after deflections. Although the optimal policy of the second run can also do this, this counteracting ability was observed to be more pronounced on one side than the other. This can be explained by the fact that the swing-up of the second run is slower and smoother, which is why, during the transition moment, the balancing movement of the cart against the angular speed of the pendulum is also gentle and only happens in one direction. This can be seen in Fig. 13. Instead of quickly applying alternating input actions to stabilize the pendulum, the action input is very low and only in the positive direction to catch the pendulum. After that, the rollout ends, resulting in the algorithm only seeing this data and, therefore, only learning how to stabilize the pendulum on one side, leading to better performance on that side. However, the successful, albeit limited, counteraction on the other side, without explicitly visible data points, also shows that the PILCO can successfully learn and model states that have not yet been observed. Furthermore, this insight also highlights the importance of setting the hyperparameters. To better understand the balancing process in the second run, a longer planning horizon, T, would be preferable. Extending the planning horizon to T enables the algorithm to observe the balancing process over a more extended period, allowing it to learn how to actively balance the pendulum rather than just holding it upright for a short time. This behavior was observed in previously conducted runs, where the algorithm could only see the balancing process for one second due to a shorter planning horizon T. In this run, the policy maintained the pendulum's upright position by moving the cart in one direction, rather than actively balancing it. Although this approach yielded a sufficiently low loss, it did not achieve accurate balance control.

5.2 Modeling accuracy

In both test runs, an initial upswing of the pendulum was learned effectively with the help of offline data. However, when examining the immediate loss, L, in Figs. 5 and 10, it is noticeable that the two loss trajectories do not align with each other. This suggests poor system dynamics modeling due to the limited number of data points observed so far. Figure 14 summarizes the modeling accuracy of both conducted test runs against each other by considering the differences between actual and predicted loss measured by the L2 norm. It is noticeable that the second test run has an overall better loss prediction than the first run, reaching good modeling accuracy in earlier rollouts compared to test run 1. This also explains the almost consistent function values ΣL_{pred} of the first three rollouts in the first run. Ideally, each rollout should improve the function value ΣL_{pred}. However, the function value was lower in the first rollout because the dynamic model was not well modeled. With better modeling from the third rollout onwards, the dynamic model could predict the loss trajectory more realistically, resulting in a higher but more realistic loss trajectory. In general, it can be said that the best-predicted loss trajectory or the highest-performing policy is not necessarily the most desirable. Much more critical is a realistically learned dynamic model of the system. For example, in the sixth rollout of the second test run, the dynamic model predicts a balance of the pendulum with low uncertainty over all prediction steps, but the policy fails to keep the pendulum upright, hence the increasing difference in Fig. 14. It would be much more efficient if the policy were predicted correctly, even with poorer performance. However, the advantage of the PILCO algorithm is that it can compensate for these incorrect modeling errors by obtaining new data from the rollouts.

Another important finding on the ability to model the dynamic system comes from the learned noise standard deviation σ_{noise}. The results showed that the learned noise values for the velocity state variables are consistently one order of magnitude higher than those for the position state variables. This difference is due to how the data are collected. Since the position values are measured directly, they have very low noise. In contrast, the velocity values are not measured directly but are calculated from the position data, resulting in much higher noise. This difference in noise levels has a strong impact on the accuracy of the dynamic model and its ability to make reliable predictions about the loss. Therefore,

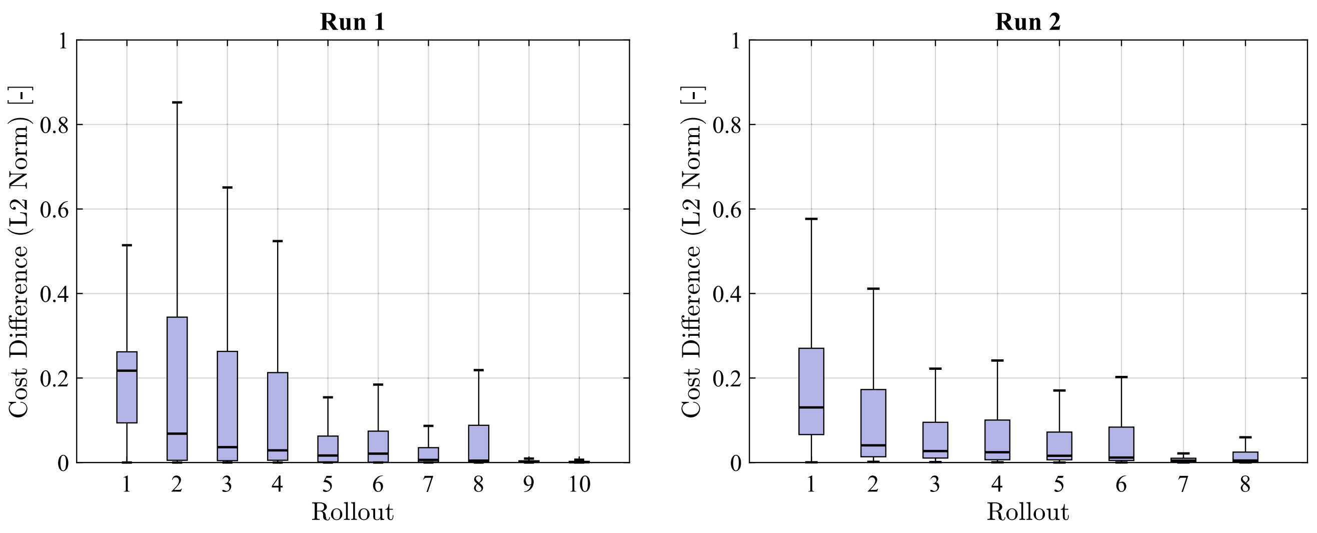

Fig. 14 Differences between predicted and actual loss for all rollouts of Run 1 and Run 2

directly measuring velocity values and thus reducing noise could potentially further decrease the required interaction time with the environment.

5.3 Offline data usage

The results showed that initial learning of the pendulum's upswing using offline data was possible. In both runs, the policy of the first rollout generated an upswing-like behavior of the cart without crossing the specified corner points $b_{1,2}$, providing a solid foundation for subsequent improvements. This resulted in a significant reduction in the interaction time with the environment in both runs. Without the offline data, the first rollout would have had to optimize data from random action inputs of the electric motor, which would not have been strong enough to swing up the pendulum. This would have resulted in the loss trajectory having a constant high loss without showing gradients to a lower loss, making it much more complex to optimize the policy and requiring more rollouts to reach the optimal policy. Nevertheless, the offline data only included the pendulum's swing-up. In the best case, a dataset that included both the swinging and balancing of the pendulum would have been available. However, this would not lead to direct learning on balancing the pendulum in the first rollout, but it could further reduce the interaction time with the environment.

5.4 Practical application and robustness

When using the PILCO algorithm in real-world applications, it's important to distinguish between learning and long-term application. During the learning application, PILCO collects data through multiple rollouts. Each rollout must start from the same initial position and have the same planning horizon T. These conditions are required for the algorithm to train the model correctly. Once the policy is trained, it can be used in a real-world application setting for an indefinite period. However, while the policy is running continuously, new data cannot be sampled for further model and policy learning, since the system no longer resets to the original starting point. If the policy starts to perform worse over time, for example, due to wear and tear, the learning process can be repeated under the same controlled conditions used during the initial training.

Another important aspect is the robustness of the learned policy. Since training always starts from the same initial state, the policy may perform poorly if started from a different initial position. This behavior is influenced by the parametrization of the hyperparameters initial state mean $mu0$ and the initial state covariance $S0$. Since in this work the pendulum always started from the same initial position before each rollout, a very low initial state covariance $S0$ was used, in order to make learning easier. To improve generalization and increase robustness of the PILCO regarding the ability to succeed from different initial conditions, a higher initial state covariance $S0$ could be used. However, this approach would introduce more uncertainty and increase the complexity of the learning task. Investigating the trade-off between learning efficiency and policy robustness under different initial state distributions presents a promising direction for future work.

6 Conclusion and outlook

This work demonstrates the practical application of a PILCO controller for controlling a real-world inverted hydraulic pendulum. After outlining the role of RL in fluid power and describing the test rig, control loop, and PILCO framework, the investigated control task and results were presented.

The experiments highlight PILCO's ability to learn precise control policies with minimal interaction. The pendulum was successfully swung up and stabilized after only 64 seconds of interaction. Comparative tests showed that reward shaping strongly influences behavior, with the best performance achieved when penalizing system energy. Offline data for policy initialization further accelerated learning, reduced wear, and lowered operational costs. A key advantage of PILCO is its data efficiency, which makes it especially suitable for fluid power systems where extensive physical testing is costly or impractical. Unlike many RL algorithms that depend on simulations or large datasets, PILCO can leverage small amounts of offline data together with a few real-world trials. Nonetheless, PILCO is not an off-the-shelf solution: careful hyperparameter tuning, reward design, and significant computational effort are still required. Future work could compare its performance to model-free RL under identical rollout and data constraints. However, such approaches usually require much more data. Additional directions include integrating sparse Gaussian processes to reduce computational cost and employing informative mean priors to exploit domain knowledge better. Prior studies by Bischoff et al. [24] and Cutler and How [25] suggest these enhancements could further improve data efficiency and adaptability. Sparse GPs reduce computational costs by approximating the full GP model, enabling the possibility of using larger datasets or decreasing the computational time for currently used datasets. However, the use of sparse GPs results in a decrease in the model accuracy, thus resulting in a trade-off, which needs to be examined. The usage of informative mean prior instead of zero-mean assumptions in the GP model could accelerate the learning process, especially in areas where no data points have been provided to the model, leading to faster convergence and potentially reducing the necessary interaction time with the environment. Another promising approach is to investigate hybrid strategies that combine simulation pre-training with real-world fine-tuning. In contrast, the direct real-world learning applied in this work avoids sim-to-real transfer challenges. It underscores the potential for immediate application to real-world systems.

Appendix

A Comparison to conventional LQR control

In previous work, a conventional control strategy using LQR control was designed to swing the pendulum upwards and balance it. Figure 15 shows the state variables of the hydraulic pendulum during LQR control. It can be seen that the LQR control takes significantly longer with 11.3 seconds to swing up the pendulum, compared to both test runs performed with the PILCO algorithm. Furthermore, a speed impulse of about $u = 2000$ rpm is required to stabilize the pendulum during the transition from swinging to balancing. This impulse magnitude is similar to the first PILCO test run but greater than that of the second run. The evaluation metrics for the conventional LQR control are shown in Table 11. The quality of balancing for LQR is 255.3992 rpm, which falls between the two PILCO test runs. Because the conventional control strategy is not learned using data, the interaction time with the environment cannot be measured. Similarly, a comparison of the ITAE metric is not meaningful, since this metric weights the angular error of the angle θ over time, and the LQR control takes significantly longer to swing up the pendulum. In terms of performance, the PILCO-based control system achieves a faster and smoother swing-up overall. However, directly comparing the two control strategies is challenging, as they follow entirely different approaches. Conventional control strategies, such as LQR or PID controllers, require extensive and precise parameterization, which can be complex and time-consuming. The PILCO algorithm, on the other hand, does not require this level of manual parameter tuning. However, the critical and complex aspect of defining the loss function requires careful consideration. Additionally, the two approaches differ fundamentally in their dependence on prior knowledge. PILCO learns from data and iteratively optimizes its policy, while LQR control requires a detailed simulation model of the system. Once parameterized, LQR control can be applied almost immediately, whereas PILCO demands substantial computation time for policy optimization. Conventional control strategies are also more modular, allowing

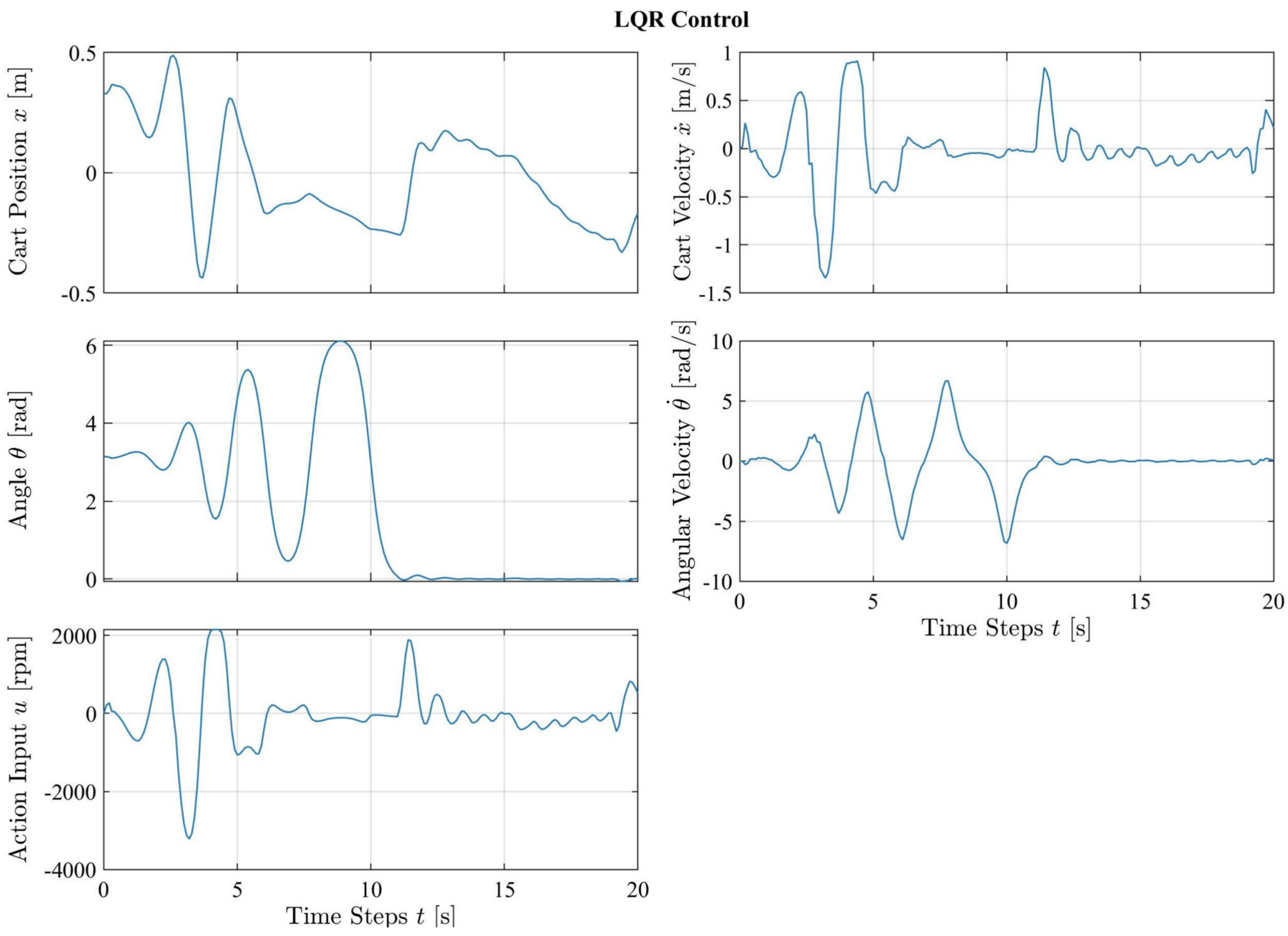

Fig. 15 State variables of the applied LQR control

Table 11 Evaluation metrics of the conventional LQR control

Metric	Value
Time until balancing	11.3 s
Interaction time	–
ITAE	–
Constraint violations	0
Quality of balancing	255.3992 rpm

for specific adjustments to control behaviors. In contrast, with PILCO, desired behaviors must be implicitly specified through the loss function. Once the policy is learned, behavior cannot be modified without re-training.

In conclusion, it is challenging to state which approach is superior, as each has strengths and limitations that make it better suited to specific contexts. Factors such as the intended use case, computational resources, and the need for flexibility in control behavior should be carefully considered when choosing between these methods.

Author Contributions F.B. and P.W. contributed to conceptualization, methodology, formal analysis, and visualization and provided software. P.W. performed the experiments and data curation. F.B. and P.W. wrote the original draft. T.G. and K.S. supervised the project and reviewed and edited the manuscript. All authors reviewed the manuscript. T.G. acquired funding and resources. F.B., P.W., and T.G. handled project administration.

Funding Open Access funding enabled and organized by Projekt DEAL. This work did not receive any direct funding.

Data Availability No datasets were generated or analyzed during the current study.

Code Availability The code used to support the findings of this study is confidential. It cannot be shared publicly due to data protection restrictions.

Declarations

Conflict of interest The authors declare no conflict of interest.

Ethical Approval Not applicable.

References

1. Hubertus M (2012) Servohydraulik - Geregelte hydraulische Antriebe (Institut für fluidtechnische Antriebe und Steuerungen)
2. Brumand-Poor F, Kauderer L, Matthiesen G, Schmitz K (2023) Application of deep reinforcement learning control of an inverted hydraulic pendulum. Int J Fluid Power
3. Brumand-Poor F, Matthiesen G, Schmitz K. Control of a hydromechanical pendulum with a reinforcement learning agent; 2nd revised edition
4. Hesse M, Timmermann J, Hüllermeier E, Trächtler A (2018) A reinforcement learning strategy for the swing-up of the double pendulum on a cart. Procedia Manufacturing 24
5. Adam S, Busoniu L, Babuska R (2012) Experience replay for real-time reinforcement learning control. Syst Man Cybernet Part C Appl Rev IEEE Trans 42:201–212
6. Sutton RS, Barto AG (2018) Reinforcement learning - an introduction. MIT Press, Cambridge
7. Levine S, Kumar A (2020) George Tucker & Justin Fu. Tutorial, review, and perspectives on open problems, Offline reinforcement learning
8. Deisenroth MP, Rasmussen CE (2011) Pilco: a model-based and data-efficient approach to policy search. Int Conf Mach Learn
9. Andersson J, Bodin K, Lindmark D, Servin M, Wallin E (2021) Reinforcement learning control of a forestry crane manipulator. IEEE/RSJ Int Conf Intell Robots Syst (IROS)
10. Wyrwał D, Lindner T, Nowak P, Białek M (2020) Control strategy of hydraulic cylinder based on deep reinforcement learning. In: 2020 international conference mechatronic systems and materials (MSM)
11. Yao Z, Liang X, Jiang G-P, Yao J (2023) Model-based reinforcement learning control of electrohydraulic position servo systems. IEEE/ASME Trans Mech
12. Bécsi T, Szabó Á, Kővári B, Aradi S, Gáspár P (2020) Reinforcement learning based control design for a floating piston pneumatic gearbox actuator. IEEE Access
13. Berglund D, Larsson N (2021) Controlling a hydraulic system using reinforcement learning – implementation and validation of a DQN-agent on a hydraulic Multi Chamber cylinder system. Ph.D. thesis, Linköping University — Department of Management and Engineering
14. Yuan X et al (2022) Reinforcement learning control of hydraulic servo system based on td3 algorithm. Machines 10:1244
15. Gao L, Zhao J, Duan J, Liu S, Ma F (2024) Reinforcement learning based motion control method for electrical-hydraulic valve-controlled system. IEEE/ASME Trans Mechatron
16. Marc PD (2015) Dieter Fox & Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control, IEEE Trans Pattern Anal Mach Intell
17. Liu M, Chowdhary G, da Silva BC, Liu S-Y, How JP (2018) Gaussian processes for learning and control. IEEE Control Syst Mag
18. Rasmussen CE, Williams CKI (2006) Gaussian processes for machine learning. MIT Press, Cambridge
19. Prof. Dr. Trimpe Sebastian (2022) Fundamentals of Machine Learning (Institute for Data Science in Mechanical Engineering (DSME))
20. Deisenroth MP (2010) Efficient reinforcement learning using gaussian processes. Ph.D. thesis, Karlsruher Institut für Technologie (KIT)
21. Deisenroth MP, McHutchon A, Hall J, Rasmussen CE (2013) Pilco code documentation v0.9
22. Rontsis N, Polymenakos K (2020) Github repository: Probabilistic inference for learning control (pilco). https://github.com/nrontsis/PILCO
23. Żegleń J (2019) The application of an adaptive controller combined with the lqr controller for the inverted pendulum. Pomiary Automatyka Robotyka 23:47–54
24. Bischoff B et al (2014) Policy search for learning robot control using sparse data. In: 2014 IEEE International Conference on Robotics and Automation (ICRA) 3882–3887
25. Cutler M, How J (2015) Efficient reinforcement learning for robots using informative simulated priors. Proc IEEE Int Conf Robot Autom 2015:2605–2612