Application of the REINFORCE Algorithm in Real-Time Inverted Pendulum Control

Faculty of Technology
Master's thesis

Kimmo Paldanius

2.7.2026
Turku

Master's thesis

Subject: Mechanical Engineering

Author(s): Kimmo Paldanius

Title: Application of the REINFORCE Algorithm in Real-Time Inverted Pendulum Control

Supervisor(s): Wallace Moreira Bessa, Gabriel Da Silva Lima

Number of pages: 61 pages

Date: 2.7.2026

This thesis investigates the application of the REINFORCE policy-gradient algorithm to upright stabilization of the Quanser Qube-Servo 2 rotary inverted pendulum. Since the balancing task is approximately linear near the upright equilibrium, classical PD control already provides an effective solution. The aim is therefore not to show that REINFORCE outperforms classical control, but to examine whether a simple model-free reinforcement learning method can learn a local balancing policy, how the resulting controller compares with a PD reference controller, and how it behaves outside the training simulator.

A Gym-compatible simulation environment was developed for the Furuta pendulum and used as the training environment. A continuous-action Gaussian policy was trained using a Monte Carlo REINFORCE update without a value-function baseline or critic network. The policy receives a compact two-dimensional feature representation formed from the arm and pendulum angles and angular velocities, and outputs the mean control action and state-dependent standard deviation. A simple survival reward encourages the policy to maximize balancing time. Both the learned policy and the PD reference controller were evaluated in the custom simulation, the Quanser virtual environment, and on the physical Qube-Servo 2 hardware.

The results show that REINFORCE learns a local stabilizing policy that works in the virtual environment and on the physical device without retraining. Settled-state angular precision was broadly comparable to that of the PD controller, but the learned controller used a more active voltage signal and was less consistently reliable. The work demonstrates the feasibility of REINFORCE for this real-time control task while highlighting its sensitivity to task formulation and the limitations of transferring from simulation to the physical device.

Keywords: reinforcement learning, REINFORCE, policy gradient, inverted pendulum, Furuta pendulum, sim-to-real transfer

Generative artificial intelligence was used for language refinement and for improving the structure and clarity of the text.

Diplomityö

Oppiaine: Konetekniikka

Tekijä(t): Kimmo Paldanius

Otsikko: REINFORCE-algoritmin soveltaminen reaaliaikaiseen käänteisheilurin säätöön

Ohjaaja(t): Wallace Moreira Bessa, Gabriel Da Silva Lima

Sivumäärä: 61 sivua

Päiväys: 2.7.2026

Tässä diplomityössä tutkitaan, voidaanko REINFORCE-vahvistusoppimisalgoritmia hyödyntää Quanser Qube-Servo 2 -pyörivän käänteisheilurin pystyasennon tasapainottamiseen. Koska tasapainotustehtävä on likimain lineaarinen pystyasennon tasapainopisteen lähellä, klassinen PD-säätö tarjoaa siihen jo tehokkaan ratkaisun. Työn tavoitteena ei siten ole osoittaa REINFORCE-algoritmin paremmuutta klassiseen säätöön nähden, vaan selvittää, voiko yksinkertainen mallivapaa vahvistusoppimismenetelmä oppia paikalliseen tasapainotukseen soveltuvan ohjauspolitiikan. Lisäksi työssä tarkastellaan, miten opittu säädin vertautuu PD-vertailusäätimeen ja miten se toimii koulutussimulaattorin ulkopuolella.

Työtä varten kehitettiin Gym-yhteensopiva simulaatioympäristö pyörivälle käänteisheilurille. Säädin toteutettiin jatkuva-arvoisena stokastisena ohjauspolitiikkana, jossa ohjaustoiminto kuvataan gaussisen todennäköisyysjakauman avulla. Ohjauspolitiikka koulutettiin Monte Carlo -REINFORCE-päivityksellä ilman arvofunktioon perustuvaa vertailutasoa tai kriitikkoverkkoa. Ohjauspolitiikka saa syötteenään kaksi piirrettä, jotka muodostetaan varren ja heilurin kulmista sekä kulmanopeuksista, ja tuottaa ohjaustoiminnon keskiarvon sekä tilasta riippuvan keskihajonnan. Yksinkertainen tasapainossa pysymiseen perustuva palkkio ohjaa oppimista kohti mahdollisimman pitkää tasapainotusaikaa. Opittua säädintä ja PD-vertailusäädintä arvioitiin tätä työtä varten kehitetyssä simulaatioympäristössä, Quanserin virtuaaliympäristössä sekä fyysisellä Qube-Servo 2 -laitteella.

Tulokset osoittavat, että REINFORCE oppii paikalliseen stabilointiin soveltuvan ohjauspolitiikan, joka toimii virtuaaliympäristössä ja fyysisellä laitteella ilman uudelleenkoulutusta. Vakiintuneessa tilassa kulmatarkkuus oli verrattavissa PD-säätimeen, mutta opitun säätimen ohjausjännite vaihteli enemmän ja sen toimintavarmuus oli heikompi. Työ osoittaa, että REINFORCE-algoritmia voidaan soveltaa tähän reaaliaikaiseen säätötehtävään, mutta tuo samalla esiin menetelmän herkkyyden oppimistehtävän määrittelylle sekä simulaatiosta fyysiselle laitteelle siirrettävyyden rajoitteet.

Avainsanat: vahvistusoppiminen, REINFORCE, politiikkagradientti, käänteisheiluri, Furuta-heiluri, siirtyminen simulaatiosta todellisuuteen

Generatiivista tekoälyä käytettiin kielen hiomiseen sekä tekstin rakenteen ja selkeyden parantamiseen.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

Symbol	Description
Physical system and control	
$\phi,\ \dot{\phi}$	Rotary arm angle and its angular velocity
$\alpha,\ \dot{\alpha}$	Pendulum angle ($\alpha = 0$ upright) and its angular velocity
x_t	Environment state vector $[\phi, \dot{\phi}, \alpha, \dot{\alpha}]^T$
$u_t,\ u_{\max}$	Motor voltage command (action) and its safety limit
$\tau,\ \tau_c$	Motor torque and smoothed Coulomb-friction torque
$K_t,\ R_m$	Motor torque constant and armature resistance
$J_r,\ J_p$	Rotary-arm and pendulum moments of inertia
$m_r,\ m_p$	Rotary-arm and pendulum masses
$r,\ l,\ L_p$	Arm length, pendulum centre-of-mass distance, pendulum length
$b_r,\ b_p$	Viscous damping coefficients (arm and pendulum)
g	Gravitational acceleration
$\Delta t,\ \Delta t_s$	Simulation timestep and internal integration substep
$K_P,\ K_D$	Proportional and derivative gains (basic PD law)
K_α, $K_{\dot{\alpha}}$, K_ϕ, $K_{\dot{\phi}}$	Gains of the four-term PD baseline controller
$e_\alpha,\ e_\phi$	Pendulum-angle and arm-angle errors
Reinforcement learning	
$s_t,\ a_t,\ r_t$	State, action and reward at timestep t
o_t	Policy observation (feature vector)
$e_1,\ e_2$	Policy features supplied to the neural network
$\lambda_1,\ \lambda_2$	Feature coefficients
π_θ	Stochastic policy with parameters θ
θ	Policy (neural-network) parameters
$\mu_\theta(o),\ \sigma_\theta(o)$	Mean and standard deviation of the Gaussian policy
a_{scale}	Action scaling applied to the policy mean

Symbol	Description
G_t	Discounted return from timestep t
γ	Discount factor
$J(\theta),\ L(\theta)$	Expected-return objective and training loss
∇_θ	Gradient with respect to the policy parameters
η	Learning rate
b	Return baseline
T	Episode length (number of timesteps)
$\alpha_{\max},\ \phi_{\max}$	Pendulum-angle and arm-angle termination thresholds
Abbreviations	
RL	Reinforcement learning
MDP	Markov decision process
PD	Proportional–derivative control
PID	Proportional–integral–derivative control
ReLU	Rectified linear unit
DC	Direct current
RMS	Root mean square

1 Introduction

1.1 Motivation

Reinforcement learning (RL) controls dynamical systems by learning from interaction rather than from analytical controller design alone [1]. It is now widely applied to the control of nonlinear and uncertain systems [2], with recent applications ranging from spacecraft attitude tracking under external disturbances [3] to depth control of underwater robots [4]. RL has also been used to synthesize feedback laws directly from data, including derivative-feedback controllers [5]. Applying these methods to real-time physical control is, however, more demanding. Traditional controllers such as the proportional–derivative (PD) law offer predictable behaviour and well-established design principles [6], whereas RL typically requires large amounts of interaction data and remains sensitive to noise, modelling errors, reward design and hyperparameter selection.

The rotary inverted pendulum is a classical benchmark for studying the control of nonlinear, underactuated and unstable systems, combining a simple mechanical structure with challenging stabilization dynamics [7], [8]. Many classical and intelligent controllers have been applied to it, including PD control with adaptive fuzzy compensation [9]. These properties make it a demanding but informative test case for a learning-based controller. The physical system used in this thesis is shown in Figure 1.1.

The Quanser Qube-Servo 2 consists of a DC motor driving a horizontal rotary arm and a pendulum that rotates freely in the vertical plane [10]. The platform provides accurate sensing, fast actuation and a corresponding virtual environment, which makes it suitable for investigating reinforcement learning methods in a controlled and repeatable setting.

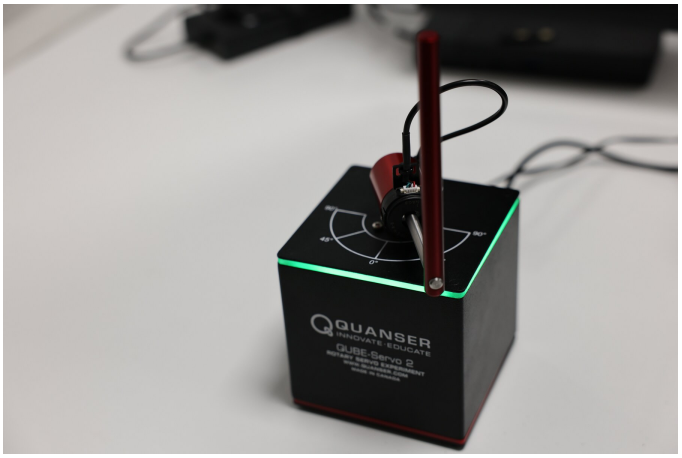

Figure 1.1: Quanser Qube-Servo 2 rotary inverted pendulum used in this thesis, with the pendulum balanced at the upright equilibrium.

The rotary inverted pendulum also provides a useful setting for studying simulation-to-real-system transfer. A controller can first be developed safely in simulation, then evaluated in a more realistic virtual environment, and finally tested on hardware. This staged workflow is particularly attractive in reinforcement learning, where direct exploration on the physical device may be unsafe, time-consuming or impractical.

Near the upright equilibrium, the rotary inverted pendulum can be approximated by a linear model, and a classical PD controller stabilizes it effectively in this region. This makes the system a useful testbed: a learning-based controller can be evaluated against a well-understood classical solution on a problem whose desired local behaviour is known. The aim of this thesis is therefore to use the pendulum as a controlled setting in which to study REINFORCE, focusing on its training dynamics, its sensitivity to the task formulation, and its practical limitations.

1.2 Research Problem

This thesis investigates the use of the REINFORCE algorithm for learning a stabilizing control policy for the rotary inverted pendulum. The purpose is to study the practical behaviour of a basic model-free policy-gradient algorithm on an unstable real-time control task; classical control methods are already well suited to local upright stabilization. REINFORCE provides a simple and transparent algorithmic starting point for this kind of study because it relies on Monte Carlo estimates of the return and does not require a critic network or learned value-function approximation.

The main challenge is that successful reinforcement learning depends not only on the learning algorithm itself, but also on how the control task is formulated. The choice of feature representation, action interpretation, reward function, termination conditions and simulation assumptions directly affects what kind of policy is learned. In this thesis, these choices define the implemented training pipeline and therefore affect how the results should be interpreted.

To study this problem, the work is organized as a staged development and evaluation process. The policy is trained in a Gym-compatible simulation environment, then evaluated in the Quanser virtual environment, and finally tested on the physical Qube-Servo 2 hardware. Figure 1.2 summarizes this progression.

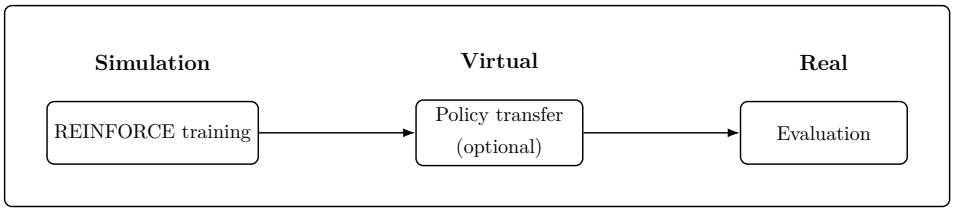

Figure 1.2: Simulation-to-hardware workflow used in this thesis.

The custom simulation is used for policy learning and initial deterministic evaluation during training. The Quanser virtual environment and the real

device are then used to test whether the learned behaviour remains useful outside the training simulator. This distinction is important because strong performance in the training environment alone is not sufficient evidence of practical controller quality.

1.3 Objectives and Research Questions

The objective of this thesis is to study how the REINFORCE policy-gradient algorithm behaves on a classical control benchmark. A neural-network policy is implemented to generate continuous-action motor voltage commands in real time, and learning takes place through sampled trajectories collected from interaction with a Gym-compatible simulation environment. The policy receives a compact two-dimensional feature representation formed from linear combinations of the arm and pendulum angles and angular velocities, and the controller is trained using a survival-based reward formulation. Because the upright equilibrium is approximately linear, a classical PD controller is also implemented and serves as a reference baseline for comparison. The aim is not to demonstrate that REINFORCE is superior to classical control, but to study its learning behaviour, the consequences of the selected task formulation, and its behaviour outside the training simulator.

The thesis addresses the following research questions:

1. **RQ1.** Can a continuous-action REINFORCE policy learn to stabilize the rotary inverted pendulum in simulation from sampled interaction data?

2. **RQ2.** How do the selected modelling and algorithmic choices, including the feature representation, reward function, action scaling and termination conditions, affect the interpretation and limitations of the learned controller?

3. **RQ3.** How does the learned REINFORCE controller compare with a classical PD baseline in terms of stabilization precision, control effort and behaviour near the upright equilibrium?

4. **RQ4.** How well does a policy trained in simulation transfer to the Quanser virtual environment and, where applicable, to the physical Qube-Servo 2 hardware?

The questions are arranged so that each subsequent question builds on the previous one. RQ1 establishes whether the chosen algorithm is viable in the training simulation. RQ2 clarifies how the selected implementation choices shape the interpretation of the results. RQ3 puts the learned controller in context by comparing it against a classical baseline. RQ4 then examines whether the resulting controller remains useful outside the training simulator. Taken together, the questions are intended to clarify how REINFORCE behaves when applied to inverted-pendulum balance, rather than to argue that it is the right tool for the task.

The main contribution of the thesis is a complete implementation and evaluation pipeline for applying continuous-action REINFORCE to the Quanser Qube rotary inverted pendulum. This includes the custom simulation environment, the policy-gradient training implementation, evaluation in the Quanser virtual environment and on hardware, and the comparison against a classical PD reference controller.

1.4 Scope and Delimitations

The study focuses on model-free policy-gradient reinforcement learning applied to a single benchmark system. More advanced actor–critic algorithms, model-based reinforcement learning and safe reinforcement learning methods are beyond the scope of this thesis. Similarly, the work does not aim to produce an optimal controller, but rather to develop and evaluate a complete

REINFORCE-based controller pipeline. Particular emphasis is placed on practical implementation, empirical evaluation and how the selected simulation and training choices affect the interpretation of the results.

The experiments report a single representative training run for the selected configuration rather than multi-seed statistics. The experiments focus on the complete implemented controller pipeline; detailed ablation studies over individual design choices, for example reward weights, action ranges or policy architectures, are left as future work.

The experiments are also restricted to the rotary, or Furuta, configuration of the Quanser Qube-Servo 2. Other benchmark variants of the inverted pendulum, such as the cart–pole or double pendulum systems, are not considered. Hardware experiments, when performed, are limited in duration and use conservative safety settings; the work does not attempt to characterize long-term controller robustness or wear-related behaviour of the physical device.

1.5 Structure of the Thesis

The remainder of this thesis is organized as follows. Chapter 2 reviews the theoretical background, including reinforcement learning, policy-gradient methods, the REINFORCE algorithm, neural-network policy representation, classical PD control and the rotary inverted pendulum. Chapter 3 describes the Quanser Qube-Servo 2 hardware, the Quanser virtual environment and the custom Gym-compatible simulation environment. Chapter 4 presents the controller design, reward formulation, REINFORCE implementation, training procedure and evaluation protocol. Chapter 5 reports the training behaviour of the learned policy and compares its settled-state performance with the PD reference controller. The results are then discussed in relation to the research questions. Finally, Chapter 6 summarizes the main findings, limitations and directions for future work.

2 Background

This chapter reviews the theoretical concepts needed for the rest of the thesis. Section 2.1 introduces reinforcement learning and the Markov decision process formalism. Sections 2.2 and 2.3 present policy-gradient methods and the REINFORCE algorithm. Section 2.4 covers classical proportional–derivative (PD) control, which serves as a reference baseline in this thesis. Section 2.5 discusses neural-network policy representation, and Section 2.6 introduces the rotary inverted pendulum as the physical system under study.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a framework for optimizing sequential decision making through interaction with an environment. An RL agent observes the state of a system, selects actions and receives numerical rewards that evaluate the quality of its behaviour [1]. Over repeated interactions, the agent adjusts its policy so that actions leading to higher long-term return become more likely.

Many control systems, especially those that are nonlinear, underactuated or difficult to model accurately, pose challenges for traditional analytical controller design. In such cases, RL is attractive because it can learn a control policy from interaction data rather than from an explicit closed-form model of the dynamics. This does not remove the need for engineering judgement: the state representation, reward function, action limits, sampling time and termination criteria still strongly influence what is learned. These design choices are especially important for unstable physical systems, where unsafe exploration can quickly lead to failure.

Control problems in RL are commonly described as Markov decision processes (MDPs), defined by the tuple

$$(\mathcal{S},\, \mathcal{A},\, P,\, R,\, \gamma), \tag{2.1}$$

where $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ is the set of admissible actions, $P(s' \mid s, a)$ is the transition probability from state s to state s' after applying action a, $R(s, a)$ is the reward function and $\gamma \in [0, 1)$ is the discount factor. The MDP formalism (2.1) provides a mathematical foundation for sequential decision making [11]. The Markov property means that the next-state distribution depends on the current state and action, but not on the earlier history, provided that the state contains all relevant information.

At each timestep t, the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to a policy $\pi(a_t \mid s_t)$, transitions to a new state s_{t+1} and receives a reward r_t. Figure 2.1 illustrates this interaction loop.

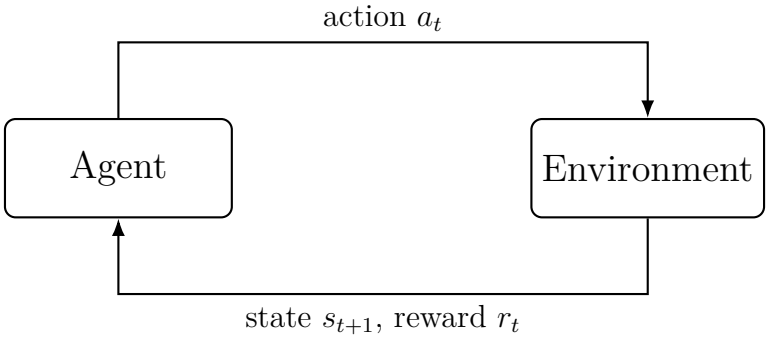

Figure 2.1: Agent–environment interaction loop in reinforcement learning.

In this loop, the policy determines how the agent acts, while the reward signal determines which behaviours are reinforced. For an inverted pendulum, the state may contain angular positions and velocities, the action may correspond to motor voltage, and the reward may encourage the pendulum to remain close to the upright equilibrium.

The objective is to find a policy π that maximizes the expected discounted return

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]. \tag{2.2}$$

The expectation in (2.2) is taken over the trajectories generated by the policy and the environment dynamics. In episodic control tasks, the infinite sum is replaced in practice by a finite episode that ends either after a maximum number of timesteps or when a termination condition is reached.

Continuous control tasks, such as torque or voltage regulation in physical actuators, require the policy to output continuous-valued actions at a fixed sampling rate. Such tasks are challenging because the controller must act smoothly and quickly despite sensor noise, delays, actuator saturation and modelling error. These challenges motivate the use of differentiable function approximators, such as neural networks, to represent policies that can be optimized from data. Value-based methods such as Q-learning [12] learn an action-value function rather than a policy directly, but they do not extend straightforwardly to continuous action spaces; policy-gradient methods are therefore more natural for the voltage-command problem studied here. Asynchronous actor–critic architectures have further demonstrated the scalability of policy-gradient approaches [13].

Despite its flexibility, RL in control presents several practical limitations. Exploration on unstable physical systems can be risky, hardware data collection is slow, reward design is non-trivial, and Monte Carlo return estimates can have high variance. Deep neural networks have enabled RL to scale to high-dimensional state spaces, as demonstrated by the Atari game-playing results of [14]. Applications of RL to physical robotic systems involve additional challenges such as sample efficiency and safety; a comprehensive overview is provided by [15]. The methodological choices used to address these issues in this thesis are presented in Chapter 4.

2.2 Policy Gradient Methods

In many continuous control problems it is natural to optimize the policy directly. Policy-gradient methods represent the policy as a parameterized function $\pi_\theta(a \mid s)$, where θ denotes the trainable parameters, and update those parameters in the direction that increases the expected return [1]. This is well suited to continuous action spaces because the policy can represent a probability distribution over real-valued actions.

For a stochastic policy, the objective can be written as

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[G(\tau)]\,, \tag{2.3}$$

where τ denotes a trajectory generated by the current policy and $G(\tau)$ is the return collected along that trajectory. Using the likelihood-ratio identity [16], the gradient of objective (2.3) can be written in the commonly used form

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t]\,, \tag{2.4}$$

where

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k \tag{2.5}$$

is the discounted return from timestep t until the end of the episode. A derivation of the policy-gradient identity (2.4) is given in Appendix A.

The term $\nabla_\theta \log \pi_\theta(a_t \mid s_t)$ in (2.4) points in the direction that would make the sampled action more likely under the current policy. Multiplying it by G_t strengthens actions that were followed by high return and weakens actions that were followed by low return. Since the expectation cannot be computed exactly in most practical problems, it is estimated from sampled trajectories.

The parameters are commonly updated by gradient ascent,

$$\theta \leftarrow \theta + \eta\, \nabla_\theta J(\theta), \tag{2.6}$$

where η is the learning rate. In software implementations based on loss minimization, the same update is often implemented by minimizing the negative policy-gradient objective. Policy-gradient methods naturally handle

stochastic policies and nonlinear function approximators, but their gradient estimates may have high variance. Variance-reduction techniques such as baselines, advantage estimation and normalization are therefore often used in more advanced algorithms [17], [18]. Actor–critic methods that combine a policy with a learned value function have been applied successfully to continuous control tasks including those with high-dimensional action spaces [19]. Model-based policy search methods such as PILCO [20] offer an alternative by learning a probabilistic dynamics model, which can dramatically reduce the number of real-world interactions required. The present thesis uses a model-free policy-gradient approach, which avoids the need for an explicit dynamics model at the cost of higher sample complexity.

2.3 The REINFORCE Algorithm

REINFORCE [16] is the original Monte Carlo policy gradient algorithm and a fundamental reference point for policy-based reinforcement learning. It generates complete episodes using the current policy and updates the policy parameters after the returns for the sampled trajectory have been computed.

For one episode of length T, the REINFORCE update follows directly from the gradient estimator (2.4):

$$\theta \leftarrow \theta + \eta \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t, \tag{2.7}$$

where G_t is the discounted return (2.5) following timestep t. The update therefore uses the return observed after each sampled action as the learning signal for that action. If an action was followed by high return, the update (2.7) increases the probability of selecting similar actions in similar states. If the return was low, the update has the opposite effect.

Algorithm 1 gives a pseudocode summary of the episodic REINFORCE procedure.

Algorithm 1 REINFORCE (Monte Carlo Policy Gradient)

Require: Differentiable policy π_θ, learning rate η, discount factor γ

Initialise the policy parameters θ
repeat
    Generate an episode $(s_0, a_0, r_0, \ldots, s_T, a_T, r_T)$ by following π_θ
    for $t = 0, 1, \ldots, T$ **do**
        $G_t \leftarrow \sum_{k=t}^{T} \gamma^{k-t} r_k$ *(Eq. 2.5)*
    end for
    $\theta \leftarrow \theta + \eta \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t$ *(Eq. 2.7)*
until convergence
return θ

This direct Monte Carlo update makes REINFORCE conceptually simple, but also sensitive to randomness in individual trajectories. For this reason, many policy-gradient methods use variance-reduction techniques, such as subtracting a value baseline, estimating advantages, normalizing returns or learning a critic network. These additions can make learning more stable and sample efficient, but they also introduce extra components beyond the basic REINFORCE algorithm.

In this thesis, REINFORCE is used deliberately because of its simplicity. The implemented controller is trained without a learned critic or value-function baseline, which keeps the algorithm easier to inspect and explain. At the same time, this choice exposes the practical weaknesses of basic Monte Carlo policy-gradient learning: sample inefficiency, high-variance updates and sensitivity to reward scaling, exploration and initialization.

2.4 Classical PD Control

Classical feedback control provides a well-established reference point for stabilizing dynamical systems such as inverted pendulums [6], [21]. Among these methods, the proportional–derivative (PD) controller is commonly used because it has low computational cost, predictable local behaviour and a direct physical interpretation. In this thesis, PD control is used as the classical baseline against which the learned REINFORCE policy is compared.

The PD control law can be written as

$$u(t) = K_P e(t) + K_D \frac{de(t)}{dt}, \tag{2.8}$$

where $u(t)$ is the control input, $e(t)$ is the error between the desired and measured variable, and K_P and K_D are the proportional and derivative gains. The proportional term produces a correction based on the current error, while the derivative term provides damping based on the rate of change of the error.

For upright pendulum balancing, the controller is usually designed for operation near the upright equilibrium, where the nonlinear dynamics can be approximated by a linear model. In this local region, (2.8) can stabilize the pendulum effectively when the gains are tuned appropriately and the actuator limits are respected. Its limitations become more visible when the pendulum moves far from the upright region, when the model is strongly nonlinear, or when the actuator saturates.

A full PID controller also includes an integral term. The integral term can remove steady-state error in many regulation problems, but it may cause windup when the actuator saturates. Since the balancing task studied here focuses on local upright stabilization rather than tracking a slowly varying setpoint, the integral term is not used. This keeps the classical baseline simple and makes the comparison with the learned policy easier to interpret.

2.5 Neural Networks for Continuous Control Policies

Artificial neural networks are widely used as function approximators in machine learning and reinforcement learning because they can represent nonlinear mappings from inputs to outputs [22]. In this thesis, a neural network is used to represent the policy that maps measured pendulum features to a continuous motor command. The rectified linear unit (ReLU) activation function is used in the hidden layer because it avoids the vanishing-gradient problem and trains efficiently in practice [23].

For stochastic continuous control, a common choice is a Gaussian policy. Instead of outputting a single deterministic action directly, the policy defines a probability distribution over possible actions [24]. This can be written as

$$\pi_\theta(a \mid o) = \mathcal{N}\left(\mu_\theta(o), \sigma_\theta^2(o)\right), \tag{2.9}$$

where o denotes the input to the policy network, $\mu_\theta(o)$ is the mean action predicted by the network and $\sigma_\theta(o)$ controls the amount of exploration. The standard deviation may be fixed, learned as a global parameter, or predicted from the policy input. Chapter 4 describes the specific parameterization used in this work.

During training, an action is sampled from the distribution (2.9). During deterministic evaluation, the mean action $\mu_\theta(o)$ can be used directly. The log probability of the sampled action is needed for the REINFORCE update (2.7). For a one-dimensional Gaussian policy, it is

$$\log \pi_\theta(a \mid o) = -\frac{1}{2}\left[\frac{(a - \mu_\theta(o))^2}{\sigma_\theta^2(o)} + \log\left(2\pi\sigma_\theta^2(o)\right)\right]. \tag{2.10}$$

Equation (2.10) links the neural network output to the policy-gradient update: backpropagation computes how changing the network parameters would change the log probability of the sampled action, and the return scales the direction and magnitude of the update.

The policy network used in this thesis is intentionally small. This is appropriate because the task is local upright stabilization rather than image-based perception or high-dimensional manipulation. A compact network also reduces computational cost and simplifies deployment to the Quanser Qube-Servo 2 control loop.

2.6 The Rotary Inverted Pendulum

The inverted pendulum is a classical benchmark system in control theory because it is unstable, nonlinear and underactuated [8]. A comprehensive survey of control methods applied to the inverted pendulum benchmark is provided by Boubaker [25]. The rotary inverted pendulum, also known as the Furuta pendulum, is named after Katsuhisa Furuta, who introduced it at the Tokyo Institute of Technology [7]. It consists of a horizontally rotating arm driven by a motor and a pendulum mounted at the end of the arm. The pendulum rotates in a vertical plane, while the arm rotates in the horizontal plane. This coupling creates nonlinear dynamics that are simple enough to simulate but challenging enough to test control algorithms.

The state of the rotary inverted pendulum is typically described by the rotary arm angle ϕ, the pendulum angle α and their angular velocities $\dot{\phi}$ and $\dot{\alpha}$. These variables can be collected into the state vector

$$x = \begin{bmatrix} \phi \\ \dot{\phi} \\ \alpha \\ \dot{\alpha} \end{bmatrix}. \tag{2.11}$$

The motor input, denoted by u, produces torque on the rotary arm. In the Quanser Qube-Servo 2 implementation used in this thesis, the command sent to the actuator is expressed as a motor voltage.

The continuous-time system dynamics can be written abstractly as

$$\dot{x} = f(x, u), \tag{2.12}$$

where f represents the nonlinear equations of motion. For reinforcement learning, the dynamics (2.12) are implemented in discrete time as

$$x_{t+1} = f_d(x_t, a_t), \tag{2.13}$$

where a_t is the action selected by the policy and f_d denotes the discrete-time transition over one sampling interval. The upright position of the pendulum is an unstable equilibrium: without active control, small deviations grow and the pendulum falls away from the balanced position.

These properties make the rotary inverted pendulum a useful testbed for this thesis. The system is physically meaningful, fast enough to require real-time control, and sensitive to modelling and implementation choices. At the same time, its structure is simple enough that classical PD control can provide a clear reference point for evaluating the learned REINFORCE policy.

3 System Description

This chapter describes the three system environments used in this thesis: the physical Quanser Qube-Servo 2 hardware, the Quanser virtual environment and the custom Gym-compatible simulation environment. The custom simulation environment is used for reinforcement learning training, the Quanser virtual environment is used as an intermediate test environment, and the physical Qube-Servo 2 is used for hardware evaluation. The chapter also introduces the state representation, motor voltage action and modelling assumptions needed to interpret the simulation, virtual-environment and hardware results. Figure 3.1 shows the complete experimental setup, which corresponds to the three-stage simulation–virtual–real workflow introduced in Chapter 1.

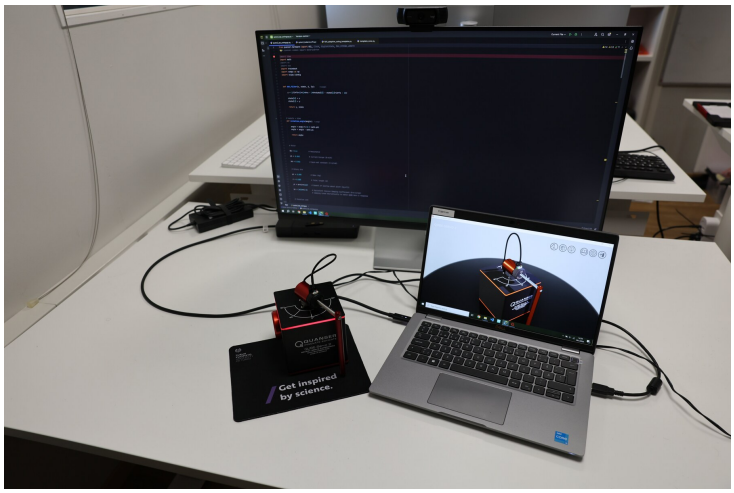

Figure 3.1: Complete experimental setup: the physical Qube-Servo 2 in the foreground, the Quanser virtual environment on the laptop, and the Python control code on the external monitor.

3.1 Quanser Qube-Servo 2 Hardware

The Quanser Qube-Servo 2 is a compact laboratory platform for real-time control experiments [10]. In the rotary inverted pendulum configuration, the system consists of a rotary arm driven by a DC motor and a pendulum attached to the end of the arm. The motor actuates the horizontal rotary arm directly, while the pendulum rotates freely in the vertical plane. The system is therefore underactuated: the pendulum is not actuated directly, but must be stabilized through the motion of the rotary arm.

The main generalized coordinates are the rotary arm angle ϕ and the pendulum angle α. The arm angle ϕ describes the rotation of the motor shaft and arm in the horizontal plane, measured from a fixed reference direction. The pendulum angle α describes the rotation of the pendulum relative to the upright balancing configuration, with $\alpha = 0$ defined as the upright position after angle conversion and wrapping. These coordinates and sign conventions are illustrated in Figure 3.2.

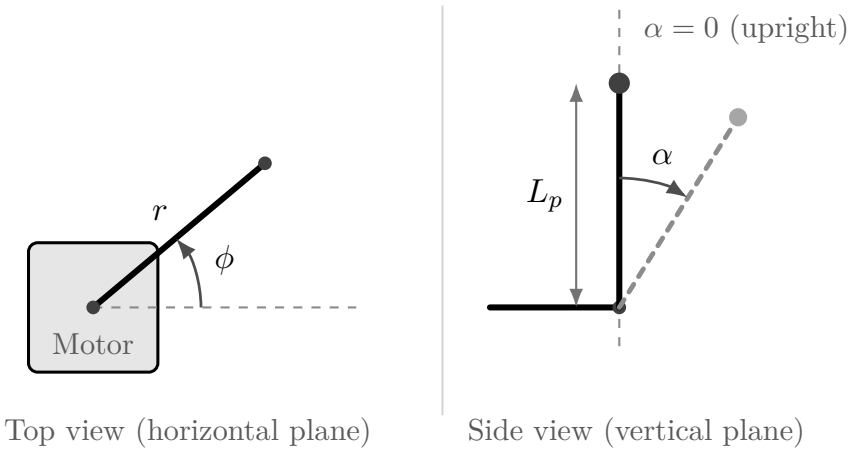

Figure 3.2: Coordinate and sign conventions of the rotary inverted pendulum, shown in top view (arm angle ϕ) and side view (pendulum angle α).

As Figure 3.2 shows, the DC motor drives the rotary arm of length r in the horizontal plane, and the arm angle ϕ is measured from a fixed reference

direction. The pendulum of length L_p is attached at the arm tip and swings in the vertical plane, with $\alpha = 0$ at the upright equilibrium; the dashed pendulum indicates a non-zero angle α.

The Qube-Servo 2 measures the arm and pendulum angles using optical encoders. Angular velocities are not measured directly, so they are estimated from consecutive angle measurements using finite differences and filtering. The actuator input is a motor voltage command; before the command is sent to the motor it is clipped to a predefined safety range in software. This voltage limit is important because the reinforcement learning policy can otherwise produce aggressive actions during exploration or during unstable deployment trials.

At each control timestep, the hardware control loop performs the same basic sequence of operations:

1. read the arm and pendulum encoder measurements,
2. convert encoder counts to angles,
3. estimate angular velocities,
4. construct the controller input,
5. compute the voltage command,
6. clip the voltage command to the allowed range,
7. write the voltage command to the motor.

This structure is used both for the classical PD controller and for the trained reinforcement learning policy. Keeping the hardware interface consistent makes it possible to compare the two controllers under similar measurement, timing and voltage-limiting conditions.

3.2 Quanser Virtual Environment

The Quanser virtual environment is a software-based representation of the Qube-Servo 2 system provided by Quanser, the manufacturer of the hardware. It provides a hardware-like interface without requiring direct interaction with the physical device. In this thesis, it is used as an intermediate test environment between the custom training simulation and the physical Qube-Servo 2. Because it is provided by the device manufacturer, it is expected to represent the behaviour of the hardware more closely than the simplified custom simulation, although the present work does not attempt to quantify this correspondence.

The virtual environment has a different role from the custom simulation environment. The custom simulation is designed for repeated reinforcement learning interaction and policy training. The Quanser virtual environment is used mainly for testing whether a controller trained or developed elsewhere still behaves correctly when executed through an interface closer to the real Qube-Servo 2. This makes it useful for detecting implementation errors before hardware experiments.

Typical issues that can be identified in the virtual environment include incorrect angle sign conventions, incorrect encoder scaling, mismatched voltage scaling, action clipping mistakes and timing differences. These errors may not be visible in the custom training simulation if the same incorrect convention is used consistently inside that simulator. Therefore, stable performance in the custom simulation alone is not treated as sufficient evidence that a controller is ready for hardware testing.

In the experimental workflow, the virtual environment is used after policy training and before real-device evaluation. A controller that fails in the virtual environment is not tested on the physical hardware before further debugging or redesign. A controller that remains stable in the virtual environment can

then be tested more carefully on the physical Qube-Servo 2 using conservative voltage limits and short evaluation runs.

3.3 Custom Gym-Compatible Simulation Environment

The custom simulation environment is the main training environment for the reinforcement learning experiments. It was implemented as a numerical model of the Furuta pendulum and wrapped in a Gym-compatible software interface [26]. Physics engines such as MuJoCo [27] provide similar simulation-to-deployment workflows for continuous control. The Gym interface provides the standard `reset` and `step` functions used by the learning algorithm, but the physical behaviour of the environment is determined by the implemented dynamics, motor-torque conversion, integration method, reward logic and termination conditions.

The simulated state is

$$x_t = \begin{bmatrix} \phi_t & \dot{\phi}_t & \alpha_t & \dot{\alpha}_t \end{bmatrix}^T,$$

where ϕ_t is the rotary arm angle, α_t is the pendulum angle relative to the upright position, and $\dot{\phi}_t$ and $\dot{\alpha}_t$ are the corresponding angular velocities. The implementation uses the convention $\alpha = 0$ for the upright balancing position (as shown in Figure 3.2), and the pendulum angle is wrapped to the interval $[-\pi, \pi]$ after each integration step.

The simulator follows the standard structure of rotary inverted pendulum dynamics [28]. It first computes the inertia matrix as a function of the pendulum angle,

$$I(\alpha) = \begin{bmatrix} J_r + J_p \sin^2 \alpha & m_p l r \cos \alpha \\ m_p l r \cos \alpha & J_p \end{bmatrix},$$

where J_r is the rotary-arm inertia, J_p is the pendulum inertia, m_p is the pendulum mass, l is the distance from the pendulum pivot to its centre of mass, and r is the rotary arm length. In the implementation, $l = L_p/2$, $J_r = m_r r^2/3$ and $J_p = m_p L_p^2/3$.

The angular accelerations are obtained by solving

$$I(\alpha) \begin{bmatrix} \ddot{\phi} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} -2J_p \sin\alpha \cos\alpha \, \dot{\phi}\dot{\alpha} - m_p l r \sin\alpha \, \dot{\alpha}^2 + \tau - b_r \dot{\phi} - \tau_c \\ J_p \sin\alpha \cos\alpha \, \dot{\phi}^2 + m_p l g \sin\alpha - b_p \dot{\alpha} \end{bmatrix}.$$

Here b_r and b_p are viscous damping coefficients and τ_c is an optional smoothed Coulomb-friction term for the rotary arm, set to zero in the reported experiments.

The motor voltage command is clipped to $u_t \in [-u_{\max}, u_{\max}]$ before being applied. The motor-torque conversion is

$$\tau = \frac{K_t}{R_m} \left(u_t - K_t \dot{\phi}_t \right),$$

where K_t is the motor torque constant and R_m is the motor resistance. The environment advances the continuous-time model using fourth-order Runge–Kutta integration [29] with main timestep $\Delta t = 0.002$ s and internal substeps of $\Delta t_s = 0.0005$ s.

3.4 Modelling Assumptions and Limitations

The custom simulation environment is a deliberately simplified model of the physical Qube-Servo 2. Its purpose is to support reinforcement learning training rather than to reproduce every hardware detail: the rotary arm and pendulum are treated as rigid bodies, the motor is described by the simplified voltage-to-torque relation given above, and the dynamics are advanced with fixed-step integration. The model does not explicitly represent encoder quantization, sensor noise, communication delay, backlash or the full nonlinear friction of the real device.

These simplifications make the simulation fast and repeatable, but they also create a gap between the custom simulator and the real hardware. Bridging the simulation-to-real gap is an active research challenge; domain randomisation is one widely used approach to improve transfer [30], and has been scaled to highly dexterous manipulation tasks [31]. A broader survey of sim-to-real transfer methods for deep reinforcement learning is provided by [32].

The Quanser virtual environment is expected to approximate the real device more closely than the custom simulator, which is why it serves as the intermediate test stage in the workflow of Figure 1.2. Results are therefore reported separately for the three environments. The comparison between REINFORCE and PD control indicates how well the learned policy behaves under the tested conditions, but does not establish global optimality or robustness over all possible initial states and disturbances.

4 Methods

This chapter describes the methodology used to design, train and evaluate the reinforcement learning controller for the Quanser Qube-Servo 2 rotary inverted pendulum. The chapter introduces the overall experimental approach, the PD reference baseline, the policy network and its parameterization, the reward formulation, the REINFORCE training procedure, and the evaluation protocol.

4.1 Overview of the Experimental Approach

The experimental work follows the staged simulation–virtual–real workflow introduced in Chapter 1. The reinforcement learning controller is first trained in the custom Gym-compatible Furuta-pendulum simulation environment described in Chapter 3. This allows many training episodes to be collected without risk to the physical device. The trained policy is then evaluated in the Quanser virtual environment, which provides a more hardware-like interface, and finally deployed on the physical Qube-Servo 2 device under conservative safety conditions.

A classical PD controller is used as the reference baseline, to place the learned controller in context rather than as a competitor to be outperformed. The upright stabilization task is approximately linear near the balanced position, and PD control is therefore expected to perform well. Policy-gradient methods have previously been applied to physical balancing tasks [33] and more broadly to robot motor skill learning [15]; the present work uses the same class of methods on the Furuta pendulum. Comparing REINFORCE against this baseline makes it possible to assess whether the learned controller produces meaningful stabilizing behaviour and how much control effort it requires.

The reported experiments evaluate one selected training formulation. This formulation includes the compact feature representation, the continuous Gaussian policy, the survival reward, voltage scaling and clipping, and fixed termination thresholds. These choices are described because they define the controller being evaluated.

4.2 PD Reference Controller

A proportional–derivative controller is implemented as a reference baseline for upright stabilization. The pendulum-angle error is defined as

$$e_\alpha(t) = \alpha_{\text{ref}} - \alpha(t), \tag{4.1}$$

where $\alpha_{\text{ref}} = 0$ denotes the upright reference. Using the error (4.1), a basic PD control law is

$$u(t) = K_P e_\alpha(t) + K_D \frac{de_\alpha(t)}{dt}, \tag{4.2}$$

where K_P and K_D are the proportional and derivative gains. In practice, derivative feedback is computed from filtered angular velocity estimates to reduce the effect of encoder noise.

The PD controller used for comparison also includes terms for the rotary arm angle and angular velocity:

$$u(t) = K_\alpha e_\alpha(t) + K_{\dot\alpha}\dot e_\alpha(t) + K_\phi e_\phi(t) + K_{\dot\phi}\dot e_\phi(t), \tag{4.3}$$

where $e_\phi(t)$ is the rotary arm angle error. The pendulum terms in (4.3) stabilize the upright position, while the arm terms prevent the rotary arm from drifting unnecessarily away from the centre. The resulting voltage command is clipped to the same safety limits used by the reinforcement learning policy.

The PD controller is only intended for local balancing. If the pendulum moves too far from the upright region, the linear control law is no longer expected

to produce useful actions. For this reason, a guard region is used during evaluation: outside the allowed pendulum-angle region the controller is not treated as a valid swing-up controller, and the trial is considered to have failed the balancing task.

4.3 Policy Network Architecture

The reinforcement learning controller is represented by a feedforward neural network policy. Although the custom simulation environment returns the four-dimensional physical state

$$x_t = \begin{bmatrix} \phi_t & \dot{\phi}_t & \alpha_t & \dot{\alpha}_t \end{bmatrix}^T, \tag{4.4}$$

the final policy input is a compact two-dimensional feature vector inspired by PD control. The features combine position and velocity for the rotary arm and the pendulum:

$$o_t = \begin{bmatrix} e_{1,t} \\ e_{2,t} \end{bmatrix} = \begin{bmatrix} \dot{\phi}_t + \lambda_1 \phi_t \\ \dot{\alpha}_t + \lambda_2 \alpha_t \end{bmatrix}. \tag{4.5}$$

In the implementation, $\lambda_1 = 1.0$ and $\lambda_2 = 5.0$. The representation (4.5) gives the policy information similar to a PD controller while reducing the dimensionality of the neural-network input. It also makes the distinction clear between the environment state x_t (4.4) and the observation o_t actually used by the policy.

The policy network is a multilayer perceptron with one hidden layer of 128 ReLU units. The output layer has two units. These are interpreted as the mean and logarithmic standard deviation of a one-dimensional Gaussian policy (see (2.9)):

$$\pi_\theta(a_t \mid o_t) = \mathcal{N}\Big(\mu_\theta(o_t), \sigma_\theta^2(o_t)\Big). \tag{4.6}$$

The first output gives the mean action $\mu_\theta(o_t)$ and the second output gives $\log \sigma_\theta(o_t)$. The logarithmic standard deviation is clamped to a fixed interval for numerical stability before being exponentiated.

During training, the controller samples a normalized action from the distribution (4.6). The sampled value is converted to a voltage command using the fixed action scale and clipped to the admissible motor-voltage range. During deterministic evaluation, the policy mean is used instead of a sampled action:

$$a_t = a_{\text{scale}}\, \mu_\theta(o_t), \tag{4.7}$$

followed by voltage clipping. Using the deterministic mean (4.7) during evaluation removes exploration noise and corresponds to how the learned policy is deployed in the virtual environment and on the physical device.

4.4 Reward Function

The reward used for the reported REINFORCE training is a simple survival reward. At each timestep, the agent receives a reward of +1 while the episode remains inside the allowed balancing region. If the pendulum or rotary arm leaves this region, the episode terminates and the terminal reward is set to zero:

$$r_t = \begin{cases} 1, & \text{if the episode continues,} \\ 0, & \text{if the episode terminates.} \end{cases} \tag{4.8}$$

The episode terminates when

$$|\alpha_t| > \alpha_{\max} \quad \text{or} \quad |\phi_t| > \phi_{\max}, \tag{4.9}$$

where $\alpha_{\max}$ is the pendulum-angle limit and $\phi_{\max}$ is the rotary-arm limit. Thus, maximizing the return (2.2) under the reward (4.8) is equivalent to maximizing the time for which the controller keeps the system inside the balancing region defined by (4.9).

This reward was selected because it gives a direct and simple learning signal for local stabilization. More shaped reward formulations were considered during development, but the final implementation uses the survival-based formulation

in order to keep the learning objective simple and directly connected to the balancing task. Since this reward does not directly penalize voltage use or small oscillations, the final evaluation also reports control effort and settled-state angle error.

4.5 REINFORCE Implementation

The REINFORCE algorithm (Algorithm 1) is used to optimize the neural network policy directly from sampled trajectories. For one episode, the policy generates a trajectory

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T),$$

and the discounted return from timestep t is computed as in (2.5):

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k, \tag{4.10}$$

where γ is the discount factor.

For the Gaussian policy (4.6), the log probability of the sampled action follows from (2.10):

$$\log \pi_\theta(a_t \mid o_t) = -\frac{1}{2}\left[\frac{(a_t - \mu_\theta(o_t))^2}{\sigma_\theta^2(o_t)} + \log\Big(2\pi\sigma_\theta^2(o_t)\Big)\right]. \tag{4.11}$$

This quantity is differentiable with respect to the neural-network parameters, so standard backpropagation can be used to compute the policy-gradient update (2.7).

The Monte Carlo policy-gradient estimator (2.4) is approximated over a batch of collected timesteps:

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid o_t)\, G_t. \tag{4.12}$$

In the implementation, this update is expressed as minimization of the negative policy-gradient objective:

$$L(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log \pi_\theta(a_i \mid o_i)\, \tilde{G}_i, \tag{4.13}$$

where N is the number of collected timestep samples in the batch and $\tilde{G}_i$ denotes the normalized return used for weighting that sample. The returns are standardized before being used in loss (4.13). This reduces the scale variation of the gradient estimates and improves numerical stability.

The policy is updated after collecting a batch of episodes rather than after every single timestep. This keeps the implementation close to the episodic REINFORCE algorithm (Algorithm 1) while reducing the effect of individual high-variance trajectories. The Adam optimizer is used for the gradient updates [34]. The learning rate, discount factor, batch size and gradient-clipping settings are reported with the training results in Chapter 5.

4.6 Training Procedure

Training is performed in episodes with a fixed maximum length. Each episode begins close to the upright equilibrium. The reset distribution samples the rotary arm angle, pendulum angle and angular velocities from narrow intervals around zero. This restricts the problem to local stabilization rather than full swing-up control. The exact reset ranges used in the reported training run are listed in Chapter 5.

At each timestep, the current environment state is converted to the policy observation (4.5), the policy samples an action from (4.6), the action is clipped to the voltage limits, and the simulator advances one timestep. The resulting reward (4.8) and termination flag (4.9) are stored together with the state, action and log probability (4.11) needed for the REINFORCE update. The episode ends when the maximum episode length is reached or when the termination condition (4.9) is triggered.

After each batch of episodes, discounted returns (4.10) are computed and used to update the policy via loss (4.13). No learned critic or value-function baseline is used. This preserves the simplicity of REINFORCE and makes the

effect of Monte Carlo policy-gradient learning easier to analyse. However, it also means that the method remains sensitive to reward scaling, exploration variance, initialization and random trajectory outcomes.

During training, the policy is also evaluated periodically in a separate deterministic evaluation run using the mean action (4.7). Training logs, evaluation scores and the best-performing policy weights are saved for later analysis. The best policy weights are selected according to this deterministic evaluation because deployment in the virtual environment and on hardware also uses the deterministic mean action rather than exploratory samples.

4.7 Evaluation Protocol

Trained controllers are evaluated without further policy updates. The REINFORCE controller is evaluated using the deterministic mean action (4.7), while the PD controller (4.3) is evaluated using the same voltage limits and safety thresholds. The comparison focuses on local upright balancing, not swing-up from the downward position.

The controllers compared are:

- the classical PD reference controller (4.3), and
- the policy trained with REINFORCE (Algorithm 1).

In simulation, controllers are evaluated from initial conditions sampled near the upright equilibrium. In the Quanser virtual environment and on the real hardware, the pendulum is first brought close to the upright region before the balancing controller is engaged. Hardware tests are performed conservatively, with voltage limits, limited episode duration and active operator supervision.

For each evaluation run, performance is assessed using settled-state metrics. The settled-state window is defined as the part of the episode after the initial

transient has decayed and the controller is actively balancing the pendulum. The main metrics are:

- root mean square (RMS) pendulum-angle error,
- mean and standard deviation of the pendulum angle,
- RMS motor voltage, and
- peak absolute motor voltage.

These metrics are computed for both controllers, allowing the learned controller to be compared against the classical baseline in terms of balancing precision and control effort.

5 Results

This chapter presents the experimental results for the REINFORCE controller and the PD reference controller. The chapter first reports the training setup and hyperparameters, then analyses the REINFORCE training behaviour, and finally compares the trained policy with the PD baseline in the Quanser virtual environment and on the real Qube-Servo 2. The chapter ends by relating the results to the research questions stated in Chapter 1.

5.1 Experimental Setup

The training hyperparameters are listed in Table 5.1. The feature coefficients $\lambda_1 = 1.0$ and $\lambda_2 = 5.0$ define the two policy-network inputs, $e_1 = \dot{\phi} + \lambda_1 \phi$ and $e_2 = \dot{\alpha} + \lambda_2 \alpha$, which combine angle and angular-velocity information. Initial conditions at every episode reset are sampled uniformly from $|\phi_0| \leq 5°$, $|\alpha_0| \leq 8°$, $|\dot{\phi}_0| \leq 0.5\,\text{rad/s}$ and $|\dot{\alpha}_0| \leq 1.0\,\text{rad/s}$.

The PD reference controller uses the four-term law described in Section 4.2. Its gains were tuned manually in the virtual environment until reliable upright stabilization was obtained, and the same gains were used for the virtual and hardware experiments. The tuned four-term law of (4.3) corresponds to the implemented voltage command $u = 2.0\,\phi + 30.0\,\alpha + 2.5\,\dot{\alpha} + 2.0\,\dot{\phi}$, in volts, with angles in radians and angular velocities in radians per second, which is clipped to $\pm 10\,\text{V}$. The derivative terms are computed from angular velocities obtained through a first-order filtered finite difference with a cutoff of approximately 8 Hz. The controller output is set to zero whenever the pendulum angle leaves the guard region $|\alpha| \leq 10°$. Evaluation follows Section 4.7: the pendulum is first lifted close to upright, control is engaged once the pendulum angle first

drops below 12°, and the controller is then run for 60 s. The settled-state window used for the deployment metrics is introduced in Section 5.3.

Table 5.1: Training hyperparameters for the REINFORCE controller.

Hidden units (single hidden layer)	128
Activation	ReLU
Action scale	10 V
Voltage limit	±10 V
$\log \sigma$ clamp range	$[-20, 0.1]$
Discount factor γ	0.9995
Learning rate (Adam)	1×10^{-3}
Gradient-norm clip	5.0
Episodes per gradient update (N_b)	5
Maximum episode length	2500 steps (5 s)
Maximum training episodes	5000
Pendulum termination threshold	$\lvert\alpha\rvert > 12°$
Arm termination threshold	$\lvert\phi\rvert > 60°$
Deterministic evaluation period	every 100 episodes
Deterministic evaluation episodes	20
Simulation timestep dt	2 ms

5.2 REINFORCE Training Behaviour

The training run that produced the deployed policy is summarized in Figure 5.1. The top panel shows the stochastic episode return (essentially equal to the surviving episode length under the survival reward), the 50-episode moving average of the stochastic return, and the deterministic evaluation score computed periodically with the mean action. The middle panel shows the episode length. The bottom panel shows the per-episode mean of the state-dependent exploration standard deviation.

For roughly the first 1000 episodes the policy fails quickly, with episode lengths typically in the range of a few tens to a few hundred simulation steps. The

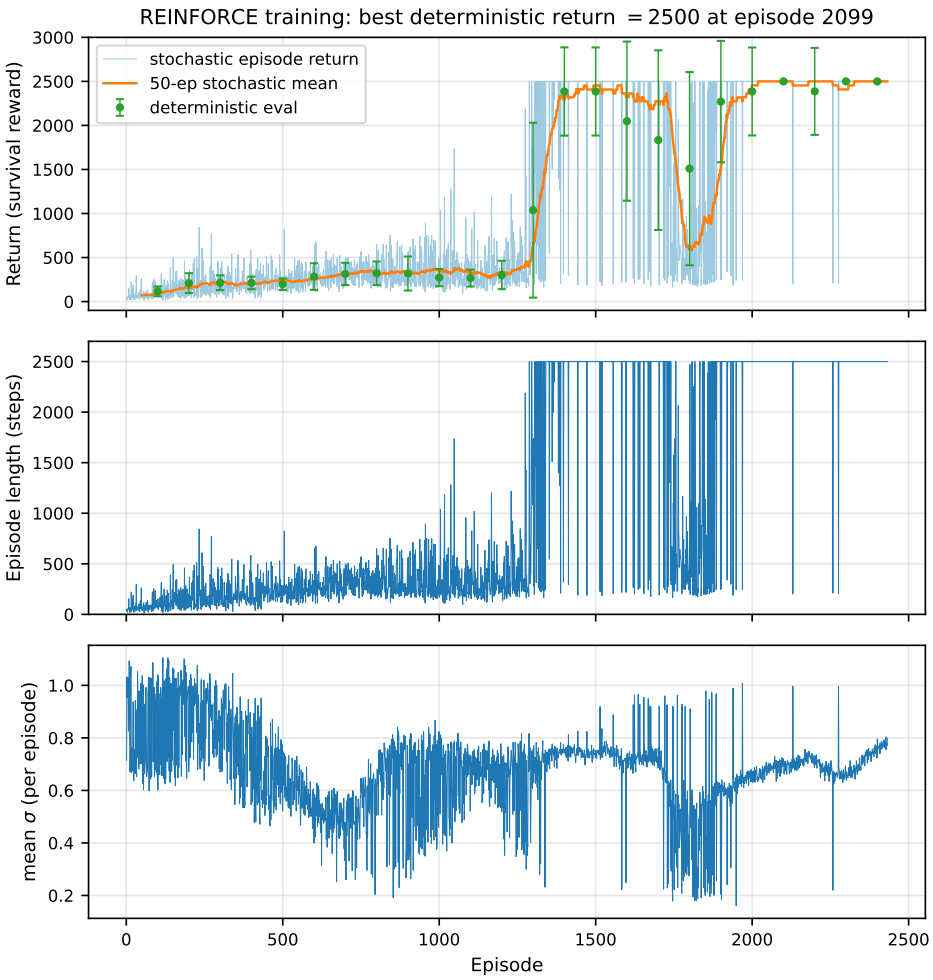

Figure 5.1: REINFORCE training behaviour. Top: stochastic return, moving-average return and deterministic evaluation score. Middle: episode length. Bottom: mean exploration standard deviation.

deterministic evaluation score in this phase rises gradually from just over 100 steps to a few hundred steps, with relatively low variance across evaluation episodes. From around episode 1100 the per-episode return becomes less stable, but the underlying trend continues upward. Around episode 1300 the policy achieves its first full-length episodes, and by around episode 1400 the deterministic evaluation score is close to the maximum episode length. This

indicates that the policy is able to keep the pendulum balanced for the full episode length in most evaluation episodes.

Between approximately episodes 1600 and 2000 the deterministic-evaluation score temporarily decreases, reaching a local minimum around episode 1800 where many evaluation episodes again end early. This corresponds to a visible dip in the moving-average return in Figure 5.1. The policy then recovers, and the best checkpoint, around episode 2100, reaches the maximum possible episode length in all 20 deterministic evaluation episodes. This best policy is the one used in the deployment experiments of Section 5.3. Training was stopped early once the recent-return mean had remained at the maximum for several deterministic evaluations.

The exploration standard deviation σ is not externally scheduled but emerges from training. Across the run it stays in a broad range around 0.5–1.0, decreasing somewhat in the first 1000 episodes, varying during the convergence phase, and ending the run around 0.7–0.8. The policy therefore retains a non-trivial amount of stochastic exploration even at the end of training; the clean settled-state behaviour observed at deployment time is recovered by using the deterministic mean action rather than samples from the trained Gaussian policy.

Two practical observations follow from this run. First, the convergence is not monotonic: the deterministic-evaluation score both increases and decreases over the course of training, even with fixed hyperparameters. Second, the thesis reports a single representative training run with this configuration. Alternative configurations explored during development did not always converge to a stabilizing policy within the same number of episodes, and a systematic characterization of run-to-run variability is left as future work.

5.3 Deployment Results

The best policy obtained from the training run of Section 5.2 and the PD reference controller were each engaged on the virtual Qube-Servo 2 and on the real device for a single 60 s balance run, recorded at 500 Hz. In the simulation environment in which the policy was trained, the deterministic policy holds the pendulum close to the upright reference and uses negligible voltage at steady state. The deployment results therefore focus on the virtual environment and the real device, where the policy was not trained.

On the real device, the pendulum-angle signal is reported after subtracting the constant encoder offset that remains after homing, so that $\alpha = 0$ corresponds to the actual settled-state pendulum position rather than to the sensor zero. The same correction is applied to both controllers and only affects the reported mean, not the RMS or standard deviation of the deviation from that mean.

The time-domain plots are presented before the numerical table because they show how the controllers behave, not only how large the final RMS values are. Figure 5.2 shows the settled-state behaviour in the Quanser virtual environment over the window $t \in [20, 30]$ s.

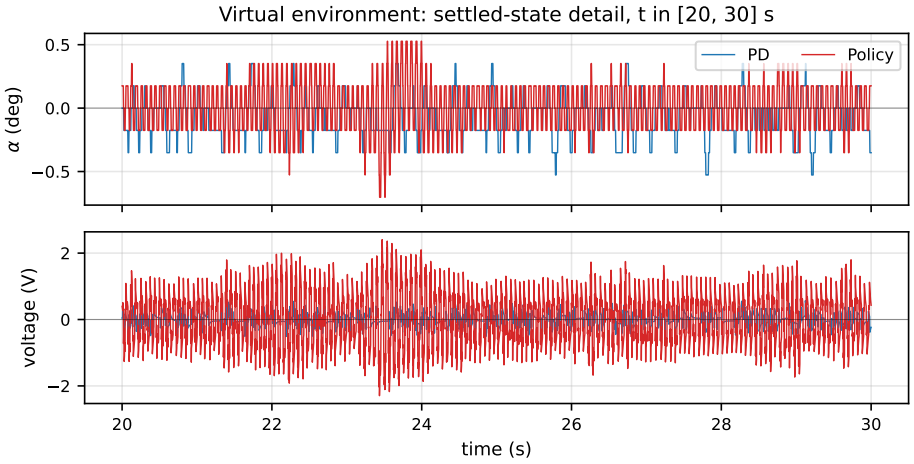

Figure 5.2: Settled-state behaviour in the virtual environment.

In the virtual environment, both controllers keep the pendulum close to upright. The clearer difference is the voltage signal: the PD controller remains close to zero for much of the settled-state window, whereas the REINFORCE policy produces a more varying control voltage.

Figure 5.3 shows the same comparison on the physical Qube-Servo 2. The discrete steps visible in the pendulum-angle traces are caused by encoder quantization rather than by abrupt physical jumps of the pendulum.

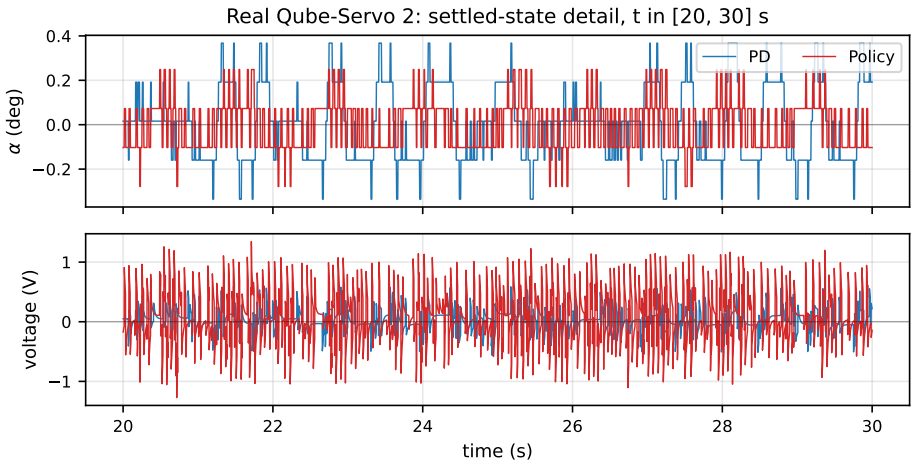

Figure 5.3: Settled-state behaviour on the real Qube-Servo 2.

The real-device run shows the same qualitative pattern as the virtual run, although the voltage difference is smaller. The learned policy stabilizes the pendulum, but it does so with a more varying control voltage than the PD controller. To quantify these observations, Table 5.2 summarizes the settled-state metrics over the longer window $t \in [10, 60]$ s, excluding the initial engagement transient.

The table confirms the visual impression from the time traces. In the virtual environment, the policy's settled-state voltage RMS is roughly four times that of the PD controller (0.85 V versus 0.19 V), and its peak voltage is substantially larger (6.27 V versus 0.62 V). On the real device the difference is

Table 5.2: Settled-state performance comparison.

	Virtual		Real	
	Policy	PD	Policy	PD
α RMS (deg)	0.22	0.18	0.11	0.18
α mean (deg)	0.01	−0.08	0.00	0.00
α std (deg)	0.22	0.17	0.11	0.18
Voltage RMS (V)	0.85	0.19	0.44	0.17
Voltage peak (V)	6.27	0.62	1.34	0.81

Window: $t \in [10, 60]$ s. Values are reported for the successful runs shown in Figures 5.2 and 5.3.

smaller but in the same direction, with the policy using 0.44 V RMS against the PD controller's 0.17 V.

In contrast, the settled-state pendulum-angle precision of the two controllers is broadly comparable. The PD controller is slightly more precise in the virtual environment, whereas the learned policy has a smaller angle standard deviation on the real device. These angle figures are on the order of the pendulum-encoder resolution (about 0.18° per count, which also produces the discrete steps visible in Figure 5.3), so differences of this size should not be over-interpreted. The main practical difference between the controllers is therefore not the achieved angle error, but the amount and smoothness of the voltage used to maintain balance.

The deployment runs reported in this section are representative of successful trials. During development, not every deployment attempt produced an equally clean balance run: some attempts ended in the pendulum leaving the controllable region shortly after engagement, or in oscillations that grew rather than decayed. A more systematic characterization of deployment reliability and disturbance rejection is left as future work.

5.4 Discussion

The findings can now be related to the research questions stated in Chapter 1.

RQ1: Can the REINFORCE algorithm learn a stabilizing controller in simulation? The training results in Section 5.2 show that this is the case. The initial policy usually failed early, but training produced a policy that kept the pendulum balanced for the full episode length at the best checkpoint. This supports the use of REINFORCE as a working baseline for the simulated local stabilization task.

RQ2: How do the selected modelling and algorithmic choices affect the interpretation and limitations of the learned controller? Unlike the other research questions, RQ2 is answered by interpretation rather than by a separate measurement. The results describe the behaviour of the implemented pipeline as a whole, and the design choices of Sections 4.3–4.4 bound what that behaviour can be. The two-dimensional feature representation makes the policy a local controller rather than a general one over the full state space, and the survival reward, which does not penalize voltage use, is the most direct explanation for the more varying control voltage reported above. The initialization range and termination thresholds fix the task as local stabilization rather than swing-up or global control.

RQ3: How does the learned REINFORCE controller compare with a classical PD baseline in terms of stabilization precision, control effort and behaviour near the upright equilibrium? On settled-state pendulum-angle precision, the two controllers are broadly comparable. The clearest difference is in control effort. The REINFORCE policy uses a more varying control voltage than the PD controller in both virtual and real environments, with a roughly four-fold higher voltage RMS in the virtual

environment and a substantially larger peak voltage during the run. The PD controller, by contrast, remains close to zero voltage when the pendulum is near upright. This is consistent with the fact that the local balancing task can be handled by a small set of tuned gains, whereas the learned policy is a nonlinear function approximator without an explicit notion of "do nothing when already balanced."

RQ4: How well does a policy trained in simulation transfer to the Quanser virtual environment and to the physical Qube-Servo 2 hardware? The transfer is successful in the limited sense that the same trained policy weights, without additional training in the target environment, can stabilize the pendulum in the Quanser virtual environment and on the physical Qube-Servo 2 in the reported trials. However, the transfer should not be interpreted as fully robust deployment. Some attempts during development failed shortly after engagement or produced growing oscillations. The result is therefore best described as successful local stabilization under favourable initial conditions, rather than as a deployment-ready controller with reliability comparable to the PD baseline.

Summary. Table 5.3 summarizes the main answer to each research question.

Table 5.3: Summary of the results in relation to the research questions.

RQ	Main finding
RQ1	REINFORCE learned a local balancing policy in simulation and kept the pendulum balanced for the full episode length at the best checkpoint.
RQ2	The selected feature representation, reward, action scaling and termination limits define a local stabilization task and shape how the results should be interpreted.
RQ3	The learned policy and PD controller achieved comparable angle precision in the reported runs, but the learned policy used more voltage.
RQ4	The trained policy transferred to the virtual environment and real hardware in successful trials, but deployment robustness was not fully characterized.

Interpretability. A further qualitative difference between the two controllers, not captured by any of the numerical metrics, is interpretability. The PD reference controller is fully described by four scalar gains with a clear physical role each, and its behaviour can be predicted from those gains alone. The trained policy, by contrast, is a neural network mapping the two combined features into a voltage command, and the way it produces a given control action is not directly inspectable. This is an inherent characteristic of learned controllers of this type rather than a property of this particular implementation, but it is worth stating explicitly: the precision-versus-effort tradeoff documented above is observable from the experimental data, whereas the internal mechanism that produces it is not.

6 Conclusions

6.1 Summary

This thesis investigated the application of the REINFORCE policy-gradient algorithm to upright stabilization of the Quanser Qube-Servo 2 rotary inverted pendulum. A custom Gym-compatible simulation environment for the Furuta pendulum was implemented, and a continuous-action Gaussian policy was trained using the REINFORCE algorithm. The policy used a two-dimensional feature representation formed from the arm and pendulum angles and angular velocities, a survival-based reward, and a small multilayer perceptron with a state-dependent exploration standard deviation. A classical proportional–derivative controller was implemented in parallel as a reference baseline.

The trained policy and the PD reference controller were evaluated under the same general protocol in the custom simulation, the Quanser virtual environment, and the physical Qube-Servo 2 device. The work examined how a simple policy-gradient method behaves on a real-time inverted-pendulum balancing task and how its behaviour compares with a conventional local controller.

6.2 Main Findings

The implemented REINFORCE controller learned a stabilizing policy in the simulation environment. The initial policy usually failed early, but training produced a policy that kept the pendulum balanced for the full episode length. This shows that REINFORCE, despite its high-variance Monte Carlo update, was sufficient to learn local upright balancing in the selected simulation task.

These design choices define the scope of this result. The two-dimensional feature representation makes the learning problem local and compact by combining angle and angular-velocity information before it is given to the neural network. The survival-based reward directly encourages the policy to remain inside the allowed balancing region, but it does not explicitly penalize voltage use or small residual oscillations. The action scaling, voltage clipping, initialization range and termination thresholds further define the task as local stabilization rather than swing-up or global control.

In the deployment experiments, the learned policy stabilized the pendulum in both the Quanser virtual environment and the physical Qube-Servo 2 in the reported successful trials. Its settled-state pendulum-angle precision was broadly comparable with that of the PD reference controller. The clearest difference was in the motor voltage signal: the learned policy produced a more varying control voltage, whereas the PD controller remained quieter near the upright equilibrium. This indicates that the REINFORCE policy learned a working balancing behaviour, but not one that matched the efficiency or simplicity of the classical controller.

6.3 Limitations

The results should be interpreted in light of the experimental scope of the thesis. The thesis reports a single representative training run and a small number of deployment runs per environment rather than multi-seed statistics or a systematic reliability study. The reported deployment results should therefore be interpreted as successful trials, not as a complete characterization of robustness. During development, some deployment attempts failed shortly after engagement or produced growing oscillations, indicating that the learned policy was less reliable than the PD reference controller.

The experiments focus on one selected controller pipeline. The feature representation, reward function, action scaling, initialization range and termination

thresholds were chosen as part of the implemented solution, but the thesis does not isolate the effect of each individual choice through ablation studies. The results therefore describe the behaviour of the complete implementation rather than proving that any single design choice is optimal.

The simulation environment is also simplified. It captures the dominant Furuta-pendulum dynamics used for training, but it does not reproduce all effects present in the physical device, such as encoder quantization, measurement noise, communication delays, backlash and nonlinear friction effects. The simulation-to-real gap is addressed empirically through evaluation in the Quanser virtual environment and on the physical device, rather than by constructing a fully identified hardware model.

The trained controller is also less interpretable than the PD reference controller, as discussed in Section 5.4: its behaviour can be evaluated experimentally, but its internal decision mechanism is far less transparent than four physically meaningful gains.

Finally, the work is limited to upright balancing of the rotary, or Furuta, configuration of the Qube-Servo 2. Swing-up from the downward position and other inverted-pendulum configurations are outside the scope of the learned controller.

6.4 Future Work

Several extensions follow naturally from the present work. First, training should be repeated over multiple random seeds to characterize run-to-run variability and to determine how reliably REINFORCE finds a stabilizing policy under the selected configuration; the importance of such reproducibility studies for deep RL is discussed by Henderson et al. [35]. Second, ablation studies over the feature representation, reward formulation, action scaling

and termination thresholds would make it possible to evaluate the effect of individual design choices more systematically.

Future work could also compare REINFORCE with more advanced policy-gradient methods, such as actor–critic algorithms, proximal policy optimization [36], or soft actor–critic [37]. These methods are designed to reduce the high variance and sample inefficiency associated with basic Monte Carlo policy-gradient learning. Another useful extension would be to initialize the policy from the PD controller through imitation learning, for example behavioural cloning or apprenticeship learning [38], before applying reinforcement learning; this could reduce the number of early failed episodes.

The simulation environment could be improved by adding more realistic friction, encoder quantization, measurement noise, delay and parameter randomization. This would allow the policy to experience a wider range of dynamics during training and could improve transfer to the physical device. Finally, safe reinforcement-learning methods or carefully constrained hardware fine-tuning could be investigated to improve real-device performance without exposing the Qube-Servo 2 to unsafe exploration; a comprehensive overview of safe RL approaches is provided by García and Fernández [39].

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.

[2] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018. DOI: 10.1016/j.arcontrol.2018.09.005.

[3] J. Zhao, Y. Chang, and J. Zhao, "Reinforcement learning-based optimal attitude tracking control for rigid spacecraft with external disturbances," *Communications in Nonlinear Science and Numerical Simulation*, 2026, In press. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1007570426002066.

[4] L. S. Cadengue, G. S. Lima, and W. M. Bessa, "Intelligent depth control of underwater robots using artificial neural networks and reinforcement learning," in *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*, IEEE, 2020, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/9306984.

[5] M. H. Zaheer, S. Y. Yoon, and S. A. A. Rizvi, "Derivative feedback control using reinforcement learning," in *2023 62nd IEEE Conference on Decision and Control (CDC)*, IEEE, 2023, pp. 7350–7355. [Online]. Available: https://ieeexplore.ieee.org/document/10383943.

[6] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ: Princeton University Press, 2008.

[7] K. Furuta, M. Yamakita, and S. Kobayashi, "Swing-up control of inverted pendulum using pseudo-state feedback," in *Proceedings of the*

Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering, vol. 206, 1992, pp. 263–269.

[8] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Hoboken, NJ: John Wiley & Sons, 2006.

[9] X. Wang, H. Dong, X. Sun, and X. Yao, "PD control of inverted pendulum based on adaptive fuzzy compensation," *Journal of Intelligent & Fuzzy Systems*, vol. 31, no. 6, 2016. DOI: 10.3233/JIFS-169186.

[10] Quanser Inc., *QUBE-Servo 2 User Manual*, Quanser Inc., 2016.

[11] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.

[12] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, 1992. DOI: 10.1007/BF00992698.

[13] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, PMLR, 2016, pp. 1928–1937.

[14] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. DOI: 10.1038/nature14236.

[15] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013. DOI: 10.1177/0278364913495721.

[16] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3–4, pp. 229–256, 1992. DOI: 10.1007/BF00992696.

[17] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *International Conference on Learning Representations (ICLR)*, 2016.

[18] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *Journal of Machine Learning Research*, vol. 5, pp. 1471–1530, 2004.

[19] T. P. Lillicrap et al., *Continuous control with deep reinforcement learning*, 2016. arXiv: 1509.02971 [cs.LG].

[20] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, Omnipress, 2011, pp. 465–472.

[21] K. Ogata, *Modern Control Engineering*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 2010.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org.

[23] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, Omnipress, 2010, pp. 807–814.

[24] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008. DOI: 10.1016/j.neunet.2008.02.003.

[25] O. Boubaker, "The inverted pendulum benchmark in nonlinear control theory: A survey," *International Journal of Advanced Robotic Systems*, vol. 10, no. 5, p. 233, 2013. DOI: 10.5772/55058.

[26] G. Brockman et al., *Openai gym*, 2016. arXiv: 1606.01540 [cs.LG].

[27] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[28] B. S. Cazzolato and Z. Prime, "On the dynamics of the furuta pendulum," *Journal of Control Science and Engineering*, 2011. DOI: 10.1155/2011/528341.

[29] R. L. Burden and J. D. Faires, *Numerical Analysis*, 9th ed. Boston, MA: Brooks/Cole, Cengage Learning, 2010.

[30] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 3803–3810. DOI: 10.1109/ICRA.2018.8460528.

[31] M. Andrychowicz et al., "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020. DOI: 10.1177/0278364919887447.

[32] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744. DOI: 10.1109/SSCI47803.2020.9308468.

[33] R. Tedrake, T. W. Zhang, and H. S. Seung, "Stochastic policy gradient reinforcement learning on a simple 3d biped," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2849–2854. DOI: 10.1109/IROS.2004.1389844.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[35] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, AAAI Press, 2018, pp. 3207–3214.

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. arXiv: 1707.06347 [cs.LG].

[37] T. Haarnoja et al., *Soft actor-critic algorithms and applications*, 2018. arXiv: 1812.05905 [cs.LG].

[38] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the 21st International Conference on Machine Learning (ICML)*, 2004, pp. 1–8. DOI: 10.1145/1015330.1015430.

[39] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

A Derivation of the Policy Gradient Theorem

This appendix derives the policy-gradient identity used in (2.4) of Chapter 2. The derivation follows the likelihood-ratio (REINFORCE) approach due to Williams [16] and the treatment in Sutton and Barto [1].

Setup

Let $\pi_\theta(a \mid s)$ be a differentiable stochastic policy parameterized by θ, and let the objective be the expected discounted return

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[G(\tau)]\,,$$

where $\tau = (s_0, a_0, r_0, s_1, \ldots, s_T)$ is a trajectory generated by the policy and the environment. The probability of a trajectory τ under policy π_θ factorizes as

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t)\, P(s_{t+1} \mid s_t, a_t),$$

where $p(s_0)$ is the initial state distribution and $P(s_{t+1} \mid s_t, a_t)$ is the environment transition probability.

Log-derivative trick

Writing the objective as an expectation over trajectory probabilities,

$$J(\theta) = \int p_\theta(\tau)\, G(\tau)\, d\tau,$$

and differentiating under the integral sign,

$$\begin{aligned}\nabla_\theta J(\theta) &= \int \nabla_\theta p_\theta(\tau)\, G(\tau)\, d\tau \\ &= \int p_\theta(\tau)\, \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)}\, G(\tau)\, d\tau \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log p_\theta(\tau)\, G(\tau)]\,. \end{aligned} \tag{A.1}$$

The step from line 1 to line 2 uses the identity $\nabla_\theta p = p\,\nabla_\theta \log p$, which holds whenever $p > 0$.

Cancellation of environment terms

Taking the log of the trajectory probability,

$$\log p_\theta(\tau) = \log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t \mid s_t) + \sum_{t=0}^{T-1} \log P(s_{t+1} \mid s_t, a_t).$$

Differentiating with respect to θ,

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t),$$

because $p(s_0)$ and $P(s_{t+1} \mid s_t, a_t)$ do not depend on θ.

Final form

Substituting back into (A.1),

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G(\tau)\right]. \tag{A.2}$$

Causality reduction

Equation (A.2) uses the full trajectory return $G(\tau)$. Since the action a_t cannot affect rewards received before time t, the gradient can be written more

efficiently using the *reward-to-go* (the discounted return from time t):

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k,$$

giving the final policy-gradient theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t\right]. \tag{A.3}$$

This is Equation (2.4) of the main text. The REINFORCE algorithm (Algorithm 1) estimates the expectation in (A.3) using Monte Carlo samples, yielding the parameter update (2.7).

Variance and baselines

The estimator (A.3) is unbiased but can have high variance. A common variance-reduction technique is to subtract a *baseline* $b(s_t)$ that does not depend on the action:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\Big(G_t - b(s_t)\Big)\right].$$

Subtracting $b(s_t)$ does not introduce bias because $\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s)] = 0$ for any fixed s [18]. In this thesis the baseline is omitted in order to preserve the simplicity of the basic REINFORCE algorithm.