Reinforcement learning based control of an underactuated double pendulum system

Master's Thesis Nr. 0183

Scientific Thesis for Acquiring the Master of Science Degree
in the study program Mechatronic und Robotics at the School of Engineering and Design of the Technical University of Munich.

Thesis Advisor	Laboratory for Product Development and Lightweight Design Prof. Dr. Markus Zimmermann
Supervisor	Laboratory for Product Development and Lightweight Design Akhil Sathuluri, Felix Wiebe, Prof.Dr. Shivesh Kumar Prof.Dr. Frank Kirchner (Second corrector)
Submitted by	Chi Zhang Karl Köglsperger Straße 9, 80939, München Matriculation number: 03735807 chi97.zhang@mytum.de
Submitted on	Garching, 15.11.2023

Declaration

I assure that I have written this work autonomously and with the aid of no other than the sources and additives indicated.

Garching, 15.11.2023

Chi Zhang

Project Definition (1/2)

Initial Situation

Add your Project Brief here. If you don't need it, comment out the creation of this Project Brief in the main document `Thesis.tex`.

Goals

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Project Definition (2/2)

Contents of this Thesis

- Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
- Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
- Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
- Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet
  - Lorem ipsum dolor sit amet

Project Note

Master's Thesis	Nr. 0183
Supervisor	Akhil Sathuluri, Felix Wiebe, Prof.Dr. Shivesh Kumar
Partners in industry/research	DFKI GmbH, Robotics Innovation Center
Time period	15.05.2023 - 15.11.2023

The dissertation project of Akhil Sathuluri, Felix Wiebe, Prof.Dr. Shivesh Kumar set the context for the work presented. My supersivor Akhil Sathuluri, Felix Wiebe, Prof.Dr. Shivesh Kumar mentored me during the compilation of the work and gave continuous input. We exchanged and coordinated approaches and results monthly.

An accurate elaboration, a comprehensible and complete documentation of all steps and applied methods, and a good collaboration with industrial partners are of particular importance.

Publication
I consent to the laboratory and its staff members using content from my thesis for publications, project reports, lectures, seminars, dissertations and postdoctoral lecture qualifications.

The work remains a property of the Laboratory for Product Development and Lightweight Design.

Garching, 15.11.2023

_____
Chi Zhang

_____
Akhil Sathuluri, Felix Wiebe, Prof.Dr. Shivesh Kumar

Contents

1 Introduction

Nonlinear systems, as their name suggests, don't adhere to linear relationships between inputs and outputs. This lack of linearity means the system's response to input changes is intricate and often unpredictable. In the real world, nearly all systems exhibit some form of nonlinearity. This nonlinearity manifests in various phenomena. For instance, in systems with multiple inputs and multiple outputs, interdependencies between variables become complex, posing a coupling problem. Chaotic behavior, often referred to as the butterfly effect, is another common issue. Even a slight alteration in initial conditions can drastically alter the system's outcome. The classic butterfly effect example illustrates this sensitivity — a butterfly's wings in Brazil potentially triggering a tornado in Texas. When dealing with control tasks, especially in many real-world systems, accounting for this nonlinearity is essential.

Gaining proficient control over nonlinear systems has been a primary objective in the field of control theory for a substantial period. Throughout history, control engineers have developed a diverse range of approaches to handle these intricate systems. The advent of robotics in recent times has brought forth new methodologies specifically designed to tackle the challenges posed by nonlinearities.

1.1 Motivation

Robots, which are programmable mechanical entities, are purposefully designed for autonomous or semi-autonomous task execution, showcasing mobility, manipulation, and interaction with their surroundings.

In the realm of modern robotics, these machines exemplify intricate, highly nonlinear mechanical systems. Some well-known instances include quadruped robotics, autonomous vehicles, quadcopters, and humanoid robots.

Figure 1.1: Caption for Image 1

Figure 1.2: Caption for Image 2

Figure 1.3: Caption for Image 3

Figure 1.4: Caption for Image 4

1.1.1 Trajectory planning and tracking

To enhance their motion capabilities, reliable nonlinear control methods are essential. Traditionally, for planning complex motion in systems like those mentioned above, a two-step approach is followed, namely trajectory planning and trajectory tracking.

Trajectory planning involves computing a smooth and feasible path that a robot should follow to reach a specific target position or work point. A common method employed is trajectory optimization, aiming to minimize a cost function that factors in travel time, energy consumption, and smoothness of motion. Techniques such as gradient descent or genetic algorithms are often used for this optimization. Adhering to constraints such as maximum velocity and accelerations is crucial during this process.

Once the trajectory is successfully planned, trajectory tracking involves implementing control algorithms to guide the robot along the planned trajectory. Typically, a feedback control approach is utilized, continuously monitoring the robot's position and adjusting control inputs. However, in real-world systems, even with a well-planned trajectory, external disturbances, uncertainties, or system limitations may cause significant deviations. Hence, accurate state estimation and robust control are crucial in the realm of feedback control.

[maybe there a better example is to explain how atlas works] In the domain of industrial robot control, it's typical to separate the process into distinct planning and execution phases. This division arises from the fact that real-time responsiveness is not a stringent necessity in this context. When a task is defined, the planning phase kicks in, utilizing algorithms like linear interpolation and A* for trajectory generation. Following this, a steady and reliable control policy is implemented to ensure precise tracking of the generated trajectory during the execution phase.

Yet, in dynamic domains like automotive and flight control, the demand for real-time responsiveness takes center stage. Model Predictive Control (MPC) stands as a prime illustration of this critical requirement. MPC seamlessly integrates trajectory planning and execution into a unified framework. The process commences by projecting a sequence of control actions into the future as part of the planning stage. Subsequently, the calculated control inputs are meticulously fine-tuned to minimize the deviation between the actual system state and the planned trajectory. This strategic implementation effectively guides the system to closely track the intended trajectory.

MPC's brilliance lies in its ability to concurrently devise and optimize trajectories while swiftly adapting in real time to stay closely aligned with the planned trajectory, even when facing

various disturbances and uncertainties. This amalgamation plays a pivotal role in achieving precise control and adaptability, especially in rapidly changing and intricate environments.

Though the traditional approach of trajectory generation and tracking has proven effective for many systems, it has the following drawbacks:

- **Limited Adaptability:** Trajectory planning typically relies on predefined paths or trajectories, limiting adaptability to unforeseen changes or dynamic environments. If the environment changes significantly, the planned trajectory may no longer be optimal or even feasible.
- **Difficulty in Complex Environments:** In highly complex and cluttered environments, planning a feasible trajectory that avoids obstacles while reaching the goal can be challenging. The complexity increases with the number of obstacles and the intricacy of the environment.
- **Difficulty with Nonlinear Systems:** Trajectory planning struggles with highly nonlinear systems where the dynamics are hard to model accurately. Linearizing the system for planning purposes may lead to suboptimal or infeasible trajectories.
- **Static Planning:** Traditional trajectory planning is often static, assuming a stationary environment. It does not readily adapt to changing circumstances or dynamic obstacles, which limits its applicability in real-world scenarios.
- **High Computational Demands:** Some trajectory planning algorithms can be computationally intensive, especially for high-dimensional or complex robotic systems. This computational demand becomes a drawback, particularly in real-time or time-critical applications.

1.1.2 Reinforcement learning based control

In contrast to trajectory-based control, reinforcement learning (RL)-based control extracts an optimal policy through interactions with the environment, offering several advantages:

- **Adaptability and Flexibility:** RL enables systems to adapt and learn optimal behavior in environments that are dynamic and chaning. The control policy can continuously evolve based on new experiences and acquired knowledge, ensuring adaptability to varying circumstances.
- **Less Model Information Required:** In contrast to traditional control methods that often necessitate a precise mathematical model of the system, RL can directly learn from inter-

actions with the environment without relying on an explicit model. This characteristic is particularly valuable in scenarios where system dynamics are complex or unknown.

- **Effective Handling of Nonlinearities and Complex Systems:** RL proves highly effective in dealing with highly nonlinear systems and complex control tasks that might pose challenges for traditional control methods. The use of neural network-based function approximation allows for the capture of intricate relationships between states and actions.
- **Efficient Handling of High-Dimensional Input Spaces:** RL demonstrates an ability to efficiently manage high-dimensional and continuous input spaces, a crucial feature in many real-world applications such as robotics, finance, and game playing.

Reinforcement learning, by learning directly from the environment, offers a dynamic and adaptable approach to control, making it particularly suitable for complex and nonlinear systems.

1.2 Problem setup

Within the realm of nonlinear systems, a particularly challenging class is underactuated systems. These systems are characterized by having fewer control inputs than degrees of freedom. This makes them notably harder to control compared to fully actuated systems. Interestingly, a majority of robots and even living beings in nature fall into the category of underactuated systems. Consequently, studying the control of underactuated mechanical systems holds significant universal relevance.

The double pendulum, a simple setup comprising two links connected by two rotational joints, is a prime example. The joints involved are the shoulder joint, directly connected to the world frame, and the elbow joint, situated between the two links. The end effector is positioned at the tip of the second link. Active control is achieved by attaching motors to the shoulder and elbow joints. In the domain of underactuated control, if the shoulder joint is actuated, the setup is known as a pendubot. On the other hand, if the elbow joint is actuated, it's referred to as an acrobot.

Despite its simple configuration, the system exhibits highly nonlinear and chaotic behavior. The double pendulum setup presents two classic tasks: swing-up and stabilization around the highest point. Research on swing-up and stabilization of the double pendulum can be traced back to the 1990s, and it continues to be a crucial testbed for validating the effectiveness of newly designed control algorithms.

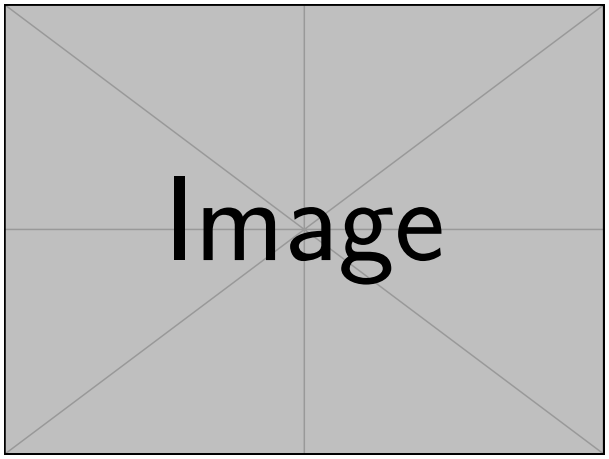


Figure 1.5: This is a sample image.

Our project's motivation is to develop a reinforcement learning-based control method suitable for underactuated control of the double pendulum system, specifically addressing swing-up and stabilization tasks. To evaluate the efficacy of this control method, we conduct both simulations and real system experiments.

1.3 Contribution

In this paper, our main contribution is as follows: developing an effective control strategy to achieve two key objectives with the double pendulum. The first task involves swinging the double pendulum from its lowest point to its highest point. The second task is to maintain stability at the highest point.

To tackle the swing-up task, we utilized a well-known model-free reinforcement learning algorithm called soft actor-critic. This algorithm allowed us to train a policy capable of reaching the region of attraction (RoA) of a continuous-time linear quadratic regulator (LQR) controller. Once the system enters the RoA, we seamlessly transition to the LQR controller to maintain stability around the highest point.

1.4 Content

The paper is structured as follows:

- **Chapter 2: State-of-the-Art**
  - This chapter provides an overview of current advancements in the field, summarizing essential theories, including those related to nonlinear and underactuated control.
- **Chapter 3: Methodology**
  - This chapter delves into the methodology, encompassing fundamental aspects of reinforcement learning, with a specific focus on the SAC algorithm. It explains the reward function used for training, the training procedure, and covers the concept of the LQR controller and how the combined controller was composed.
- **Chapter 4: Simulation Results**
  - In this chapter, we present the results obtained from simulations, showcasing the performance and behavior of the designed control strategy.
- **Chapter 5: Hardware Results**
  - This chapter reports the outcomes of experiments conducted on the hardware, providing insights into how we addressed the sim2real transfer problem.
- **Chapter 6: Discussion and Future Work**
  - The final chapter engages in a discussion about the obtained results and explores potential future directions for research and development.

2 State of the art

This chapter is about the state of the art.

2.1 Theory

This section is about the theory, for example dynamics and underactuated control.

2.1.1 Underactuated system

According to newtons second law($F = ma$), The dynamics of mechanical systems can be discribed as follows:

$$\ddot{q} = f(q, \dot{q}, u, t) \tag{2.1}$$

Where the state is given by a vector of positions(also known as the configuration vector), and a vector of velocities, $\dot{q}$.

For control, the second order differential equation can be rewritten as below:

$$\ddot{q} = f_1(q, \dot{q}, t) + f_2(q, \dot{q}, t)u \tag{2.2}$$

For a controlled dynamical system described by equation(2.2), if we have:

$$\text{rank}[f_2(q, \dot{q}, t)] < \dim[q] \tag{2.3}$$

then the system is underactuated at $(q, \dot{q}, t)$. There is another case for underactuation is even when f_2 is full rank, but additional constraints like $|\mathbf{u}| \leq 1$ can also make a system underactuated.

2.1.2 Dynamics of underactuated double pendulum system

As shown in figure 2.1, we model the dynamics of the double pendulum with 15 parameters which include 8 link parameters namely masses (m_1, m_2) , lengths (l_1, l_2) , center of masses (r_1, r_2) , inertias (I_1, I_2) for the two links, and 6 actuator parameters namely motor inertia I_r , gear ratio g_r , coulomb friction (c_{f1}, c_{f2}) , viscous friction (b_1, b_2) for the two joints and gravity.

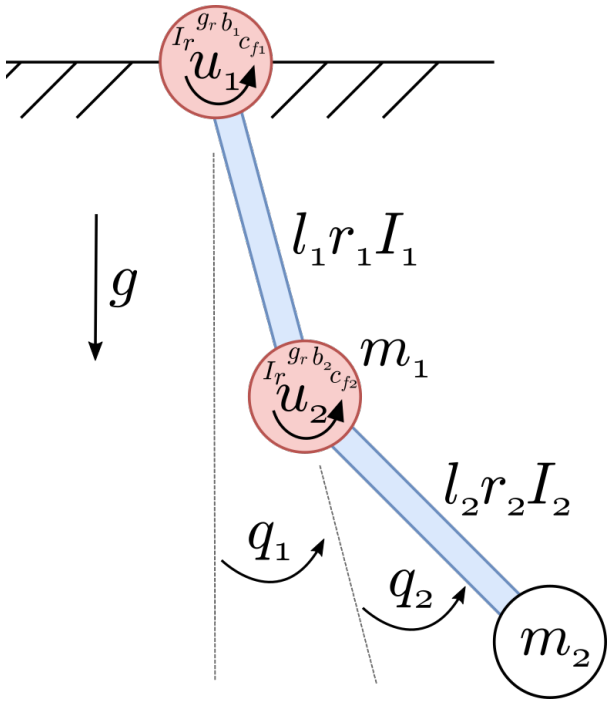


Figure 2.1: Double pendulum dynamics

The generalized coordinates $\mathbf{q} = (q_1, q_2)^T$ are the joint angles measured from the free hanging position. The state vector of the systems contains the position coordinates and their time derivatives: $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}})^T$. The torque applied by the actuators are $\mathbf{u} = (u_1, u_2)$. The equation of motion for the dynamics of a dynamical system can be derived following the blow steps:

Step 1. Define the Lagrangian (L):

The Lagrangian (L) is defined as the difference between the kinetic energy (T) and the potential energy (U) of the system:

$$L = T - U \tag{2.4}$$

Step 2. Express the Kinetic Energy (T):

The kinetic energy (T) of the double pendulum is the sum of the kinetic energies of both pendulums. The kinetic energy for a pendulum is given by:

$$T = \frac{1}{2}m_1(\dot{x}_1^2 + \dot{y}_1^2) + \frac{1}{2}m_2(\dot{x}_2^2 + \dot{y}_2^2) \tag{2.5}$$

where m_1 and m_2 are the masses of the pendulums, (x_1, y_1) and (x_2, y_2) are their positions, and $\dot{x}_1, \dot{y}_1, \dot{x}_2, \dot{y}_2$ are their respective velocities.

Step 3. Express the Potential Energy (U):

The potential energy (U) of the double pendulum is the sum of the potential energies of both pendulums. The potential energy for a pendulum in a gravitational field is given by:

$$U = m_1 g y_1 + m_2 g y_2 \tag{2.6}$$

where g is the acceleration due to gravity.

Step 4. Formulate the Lagrange's Equation:

Use Lagrange's equation to derive the equations of motion for the generalized coordinates x_1, y_1, x_2, y_2.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0 \tag{2.7}$$

Step 5. Solve the Equations of Motion:

Solve the obtained set of second-order differential equations to determine the equations of motion for the system. The system dynamics with friction is:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} = Du + G(q) - F(\dot{q}) \tag{2.8}$$

Because the state vector is $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}})^T$, the equation of motion can also be expressed as:

$$\begin{aligned} \dot{x} &= f(x,u) \\ &= \begin{bmatrix} \dot{q} \\ M^{-1}(Du - C(q,\dot{q})\dot{q} + G(q) - F(\dot{q})) \end{bmatrix} \end{aligned} \tag{2.9}$$

Consider the forward kinematics of double pendulum system, the coordinate of the joint between the first link and the second link is $P_1 = (x_1, y_1)$,the coordinate of the end effector is $P_2 = (x_2, y_2)$.

$$\begin{cases} x_1 = l_1 \sin(q_1) \\ y_1 = -l_1 \cos(q_1) \end{cases} \tag{2.10}$$

$$\begin{cases} x_2 = l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \\ y_2 = -l_1 \cos(q_1) - l_2 \cos(q_1 + q_2) \end{cases} \tag{2.11}$$

Put equation(2.12) and (2.13) into (2.7)(2.8)(2.9)(2.10),we can get the mass matrix (with $s_1 = \sin(q_1), c_1 = \cos(q_1), \ldots$)

$$\mathbf{M} = \begin{bmatrix} I_1 + I_2 + l_1^2 m_2 + 2 l_1 m_2 r_2 c_2 + g_r^2 I_r + I_r & I_2 + l_1 m_2 r_2 c_2 - g_r I_r \\ I_2 + l_1 m_2 r_2 c_2 - g_r I_r & I_2 + g_r^2 I_r \end{bmatrix} \tag{2.12}$$

the Coriolis matrix:

$$\mathbf{C} = \begin{bmatrix} -2\dot{q}_2 l_1 m_2 r_2 \sin(q_2) & -\dot{q}_2 l_1 m_2 r_2 \sin(q_2) \\ \dot{q}_1 l_1 m_2 r_2 \sin(q_2) & 0 \end{bmatrix}, \tag{2.13}$$

The gravity vector:

$$\mathbf{G} = \begin{bmatrix} -gm_1 r_1 \sin(q_1) - gm_2 \left(l_1 \sin(q_1) + r_2 \sin(q_{1+2})\right) \\ -gm_2 r_2 \sin(q_{1+2}) \end{bmatrix}, \tag{2.14}$$

The friction vector:

$$\mathbf{F} = \begin{bmatrix} b_1 \dot{q}_1 + c_{f1} \arctan(100\,\dot{q}_1) \\ b_2 \dot{q}_2 + c_{f2} \arctan(100\,\dot{q}_2) \end{bmatrix} \tag{2.15}$$

(the $\arctan(100\,\dot{q}_i)$ function is used to approximate the discrete step function for the coulomb friction)

and the actuator selection matrix $\mathbf{D}$:

$$\mathbf{D}_{full} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{D}_{pendu} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{D}_{acro} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.16}$$

for the fully actuated system, the pendubot or the acrobot.

2.2 Related work

In this section, we want to introduce the methods of underactuated control and the works related.

Extensive research has been conducted on complex robotic motion control, with learning-based methods gaining widespread adoption in recent times.

A notable example is the classical control module within the Gymnasium environment, a well-known reinforcement learning framework developed by OpenAI. This module serves as an ideal platform for exploring and evaluating new motion control algorithms.

Another noteworthy achievement in the field is the DeepMimic project [2]. Through imitation learning within a simulated environment, this project has enabled robots to acquire intricate human movements, including backflips and martial arts maneuvers.

Additionally, a research team from ETH Zurich has made significant progress in addressing the sim-to-real interface challenge [3]. Their methodology involves training a control model

for a mini-cheetah-like robot within a simulation environment. By utilizing a neural network and leveraging data collected from the real robot, they approximated the dynamics model of the physical robot. This approach has facilitated accurate implementation of the control policy derived from the virtual environment onto the real robot. These exemplary works demonstrate the promising applications of learning-based approaches in various aspects of robot control.

3 Methodology

Our primary goal is to achieve the swing-up movement in the pendubot or acrobot setup and to maintain stability around the highest points. Initial training trials with reinforcement learning have revealed challenges. These include potential entrapment in local minima and difficulty in maintaining stability at the highest point for extended periods. To address these challenges, we adopt two main strategies. For the stabilization issue, we introduce a combined controller. During the swing-up process, an RL-trained agent, using the Soft Actor-Critic—a classic model-free reinforcement learning algorithm—assumes control based on its learned policies. Yet, as the system nears the maximum point, a seamless transition takes place. This shift allows a continuous-time LQR controller to take over, ensuring the final stabilization required to maintain stability at the highest point.

3.1 Soft actor critic

Within the landscape of reinforcement learning, the Soft Actor Critic (SAC) stands out as an algorithm specifically designed for environments with continuous action spaces. Such environments, exemplified by our double pendulum system where actuators can be adjusted to any value within the torque limit range, influenced our decision to adopt SAC.

Like many other deep reinforcement learning algorithms, SAC optimizes a policy by maximizing the expected cumulative reward the agent obtains over time. This optimization is primarily achieved through an actor-critic structure.

The actor determines the best actions by interpreting the current environmental conditions and adhering to the existing policy. Typically, the actor is visualized as a shallow neural network that approximates the mapping between the input state and the output probability distribution over actions. Furthermore, SAC incorporates a stochastic policy within its actor, which fosters exploration and aids the agent in refining its policies.

On the other hand, the critic evaluates the value of state-action pairs. It estimates the expected cumulative reward the agent can achieve by following a particular policy. More often than not, the critic is depicted as a neural network that processes state-action pairs as inputs to yield the estimated value.

A distinguishing feature of SAC, besides the actor-critic framework, is entropy regularization. SAC utilizes a stochastic policy. This means that instead of always settling on a single best action for each state, the agent considers a probability distribution over potential actions. The incorporation of entropy in SAC aims to encourage exploration: high entropy signifies a more uniform distribution, implying the agent's uncertainty and tendency to explore diverse actions, while low entropy points to a concentrated distribution, suggesting the agent's confidence in a specific action. By definition, entropy quantifies randomness. Within SAC, it captures the unpredictability of the policy's action distribution. If x is a random variable with a probability density function P, the entropy H of x is defined as:

$$H(P) = \mathop{\mathbb{E}}_{x\sim P}[-\log P(x)] \tag{3.1}$$

By maximizing entropy, SAC encourages exploration and accelerates learning. It also prevents the policy from prematurely converging to a suboptimal solution. The trade-off between maximizing reward and maximizing entropy is controlled through a parameter, α. This parameter serves to balance the importance of exploration and exploitation within the optimization problem. The optimal policy π^* can be defined as follows:

$$\pi^* = \underset{\pi}{arg\max} \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H(\pi(\cdot \mid s_t))\Big)\right] \tag{3.2}$$

During training, SAC learns a policy π_θ and two Q-functions Q_{ϕ_1}, Q_{ϕ_2} concurrently. The loss functions for the two Q-networks are $(i \in 1,2)$:

$$L(\phi_i, D) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim D}\left[\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2\right], \tag{3.3}$$

where the temporal difference target y is given by:

$$\begin{aligned} y(r,s',d) &= r + \gamma(1-d) \times \left(\min_{j=1,2} Q_{\phi_{targ,j}}(s',\tilde{a}') - \alpha \log \pi_\theta(\tilde{a}' \mid s')\right), \\ \tilde{a}' &\sim \pi_\theta(\cdot \mid s') \end{aligned} \tag{3.4}$$

In each state, the policy π_θ should act to maximize the expected future return Q while also considering the expected future entropy H. In other words, it should maximize $V^\pi(s)$:

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}[Q^\pi(s,a)] + \alpha H(\pi(\cdot \mid s)) \tag{3.5}$$

$$= \mathop{\mathbb{E}}_{a\sim\pi}[Q^\pi(s,a)] - \alpha \log \pi(a \mid s) \tag{3.6}$$

By employing an effective gradient-based optimization technique, the parameters of both the actor and critic neural networks undergo updates, subsequently leading to the adaptation of the policies themselves.

In conclusion, SAC's combination of stochastic policies, exploration through entropy regularization, value estimation, and gradient-based optimization make it a well-suited algorithm for addressing the challenges posed by continuous state and action spaces.

3.2 Linear quadratic regulator

The Linear Quadratic Regulator (LQR) is an effective control method primarily designed for linear systems. Yet, when dealing with nonlinear dynamics, it remains applicable.The nonlinear system is linearized around a selected operating point, and based on this linearized version, the LQR controller can be sculpted.

Taking a step back, the general form of a nonlinear system can be expressed as:

$$\dot{x}(t) = f(x(t), u(t)) \tag{3.7}$$

In certain applications like pendubot or acrobot stabilization, it is important to select the appropriate operating point. We select the operating point around the upright position, specifically $x_{op} = [\pi, 0, 0, 0]^T$. Around this point, the system can be linearized, leading to:

$$\dot{\bar{x}}(t) = A\bar{x}(t) + Bu(t) \tag{3.8}$$

Here, the deviation from the desired state is given by $\bar{x} = x - x_{\text{op}}$, and its first derivative, $\dot{\bar{x}} = \dot{x}$. The linearized matrices A and B are derived as:

$$A = \left.\frac{\partial f}{\partial x}\right|_{\text{op}}, \quad B = \left.\frac{\partial f}{\partial u}\right|_{\text{op}} \tag{3.9}$$

To derive an optimal control strategy, we use a quadratic cost function J:

$$J = \int_0^\infty \left(x^T Q x + u^T R u\right) dt \tag{3.10}$$

Usually, the matrices Q and R are chosen to be symmetric and positive definite, which means $Q = Q^T$ and all its eigenvalues are positive, and similarly $R = R^T$ with all positive eigenvalues. With such choices, the Hamilton-Jacobi-Bellman equation, whose solution gives the optimal cost-to-go function from which the optimal policy is derived, can be reduced to the continuous-time algebraic Riccati equation.

$$A^T S + SA - SBR^{-1}B^T S + Q = 0 \tag{3.11}$$

Finally, the LQR control law obtained with above method for this linearized system is:

$$u(t) = -K\bar{x}(t) \tag{3.12}$$

with $K = R^{-1}B^T S$.

For an infinite-horizon LQR controller like this, torques will always be applied to steer the system state toward the origin of the linear system.

3.3 Combining SAC and LQR with region of attraction

In our approach, we employ a combined control method for both the swing-up and stabilization tasks. During the swing-up phase, we utilize the SAC controller. Once the state of the double pendulum approaches the vicinity of the desired goal state, we transition from the SAC controller to the LQR controller for final-stage stabilization. A vital aspect of any combined control strategy is determining the conditions under which the system is primed for a transition between control methods. In our SAC+LQR control strategy, the region of attraction method serves as the criterion for making this determination.

The region of attraction (ROA) for a nonlinear system, represented by R_a, signifies the set $\mathscr{B}$ of initial states surrounding a fixed point x_0. If a state lies within this region, the system will gravitate towards x_0 as $t \to \infty$. In the context of an LQR controller, the ROA demarcates the area within the state space where the controlled system exhibits asymptotic stability. For complex systems, directly computing R_a can be challenging; it is often estimated instead. Therefore, analyzing the stability of the controlled system within this region becomes crucial. The state space representation of a controlled linear system can be expressed as:

$$\dot{x}(t) = (A - BK)x(t) \tag{3.13}$$

where $A_c = A - BK$.

The most straightforward way to analyze the stability of a system is using the Lyapunov method. We choose a Lyapunov function in a quadratic form:

$$V(\bar{x}) = \bar{x}^T S \bar{x} \tag{3.14}$$

where S is a positive definite matrix. This function serves as an "energy-like" metric, and our goal is to demonstrate its decrease over time for a stable system.

Next, we compute the time derivative of the Lyapunov function. For the infinite horizon LQR, $\frac{\partial S}{\partial t} = 0$ and $\frac{\partial x_0}{\partial t} = 0$, hence $\dot{V}$ is:

$$\dot{V}(\bar{x}) = 2\bar{x}^T S \dot{\bar{x}} \tag{3.15}$$

For the system to be asymptotically stable at the equilibrium x_{op}, we require the system to satisfy the Lyapunov conditions:

$$\begin{cases} V(\bar{x}) > 0 \\ \dot{V}(\bar{x}) < 0 \quad \text{for all} \quad x \neq x_{op} \end{cases} \tag{3.16}$$

To construct an optimizable region of attraction, we define an upper bound ρ for the Lyapunov function V. We seek the greatest ρ such that the above Lyapunov conditions are satisfied:

$$\begin{cases} \mathscr{B} = \{x \mid 0 < V(x) < \rho\} \\ \dot{V}(x) = 2\bar{x}^T S\dot{\bar{x}} < 0 \end{cases} \tag{3.17}$$

Using the algorithm mentioned in [some paper], an inner approximation of the RoA is achieved by randomly chosing initial states sampled from a successively shrinking estimate $\mathscr{B}$. Consequently, the resulting shape of the ROA in a 4D state space resembles an ellipsoid.

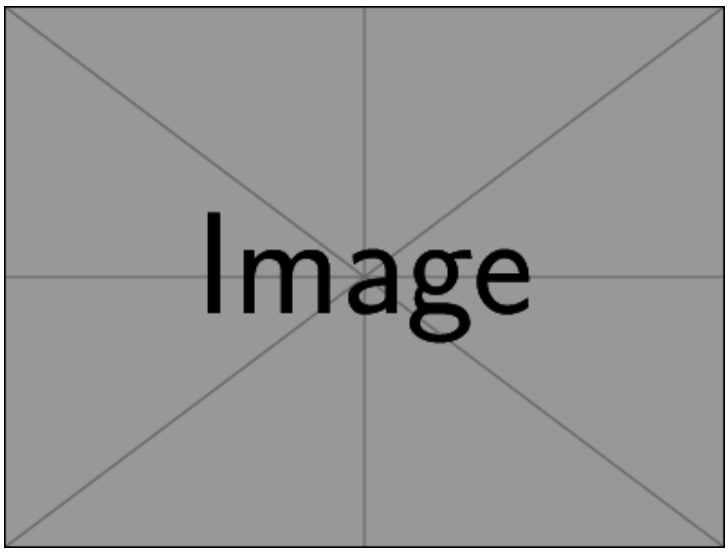


Figure 3.1: Region of Attraction

3.4 Reward shaping

In theory, the reward function is designed to guide the agent's behavior towards achieving stability around the system's goal point. However, initial training attempts have revealed challenges. For instance, the agent might get stuck in a local minimum for an extended period, fail to perform a swing-up, or be unable to maintain stability around the upright position for a prolonged duration. The stabilization problem is addressed by the LQR controller and its region of attraction, which guarantees asymptotic stability. To tackle the swing-up issue, we designed a customized three-stage reward function to steer the agent away from problematic

local minima and into the region of attraction of the LQR controller. The full equation for this reward function is:

$$\begin{aligned} r(x,u) = &-(x-x_g)^T Q_{train}(x-x_g) - u^T R_{train} u \\ &+ \begin{cases} r_{line} & \text{if } h(p_1,p_2) \geq h_{line} \ , \\ 0 & \text{else} \end{cases} \\ &+ \begin{cases} r_{LQR} & \text{if } (x-x_g)^T S_{LQR}(x-x_g) \geq \rho \ , \\ 0 & \text{else} \end{cases} \\ &- \begin{cases} r_{vel} & \text{if } |v_1| \geq v_{thresh} \ , \\ 0 & \text{else} \end{cases} \\ &- \begin{cases} r_{vel} & \text{if } |v_2| \geq v_{thresh} \ , \\ 0 & \text{else} \end{cases} \end{aligned} \tag{3.18}$$

In the initial stage, a quadratic reward function is employed to encourage smooth swinging of the entire system within a relatively small number of training sessions. The matrix $Q_{train} = diag(Q_1, Q_2, Q_3, Q_4)$ is a diagonal matrix, while R_{train} is a scalar. This is due to the nature of underactuated control in the double pendulum system, where only a single control input is available.

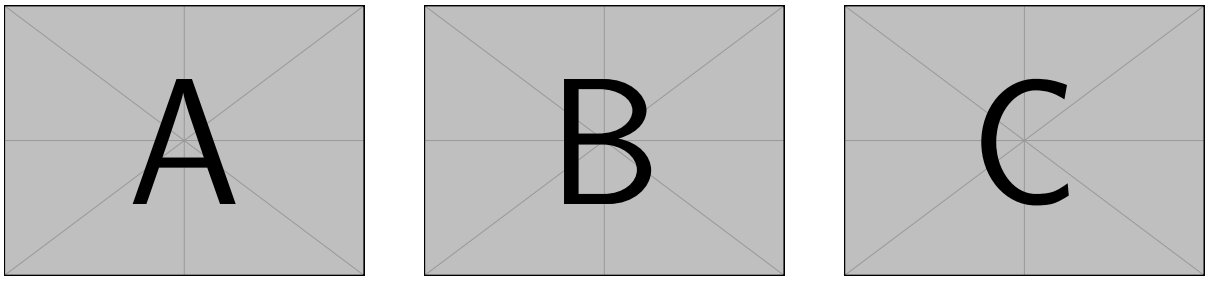
As the end effector reaches a threshold line $h_{line} = 0.8(l_1 + l_2)$, we introduce a second level of reward r_{line}. The end effector height is given by

$$h(p_1, p_2) = -l_1 \cos(p_1) - l_2 \cos(p_1 + p_2). \tag{3.19}$$

with the link lengths l_1 and l_2. This reward provides the agent with a fixed value but is carefully designed to prevent the system from spinning rapidly in either clockwise or counterclockwise directions. To discourage the agent from exploiting rewards by spinning at excessive speeds, a significant penalty $-r_{vel}$ is implemented for any speed exceeding $v_{thresh} = 8\,\text{rad/s}$ in absolute value. This penalty effectively compels the agent to approach the maximum point while adhering to the predefined speed interval. The speed penalty was only needed for the acrobot.

The third level of reward r_{LQR} aims to provide a substantial reward to the agent when it remains within the Region of Attraction (RoA) of the LQR controller. By this we want to achieve that the policy learns to enter the LQR controller RoA so that there can be a smooth transition between both controllers. For details on the LQR controller and its region of attraction, we refer to these lecture notes. The parameters, we used in the cost matrices of the LQR controller are listed

in Table 4.1. We computed the RoA similar to but with a sums of squares method. Once the RoA is computed, it can be checked whether a state x belongs to the estimated RoA of the LQR controller by calculating the cost-to-go of the LQR controller with the matrix S_{LQR} and comparing it with the scalar ρ.



(a) First image caption *(b) Second image caption* *(c) Third image caption*

Figure 3.2: Overall caption for all images

4 Experiment: agent training and simulation

Chapters 4 and 5 focus on the experimental phase of our project. This chapter details the training procedure of our SAC agent for the swing-up task and then transitions to a simulation phase to validate results for both the swing-up and stabilization tasks. We have structured this chapter into three distinct subsections: training setup, training process, and simulation results.

The first subsection describes the foundation of a reinforcement learning interaction rooted in a stable baseline3-based RL algorithm. It also touches upon a customized environment inherited from the OpenAI Gym environment. The second subsection centers on hyperparameter tuning and highlights the challenges encountered during training. The third subsection displays the results acquired from both the pendubot and acrobot setups.

4.1 Training setup

Stable Baseline 3(SB3)[some paper] is an open-source implementation of deep reinforcement learning algorithms based on the Python language. This library includes seven commonly used model-free deep reinforcement learning algorithms including SAC. Prior implementations of deep RL algorithms often encountered a problem wherein small implementation details could greatly affect performance, typically exceeding the differences between algorithms[some paper]. The developers of SB3 have done a commendable job stabilizing the performance of these deep RL algorithms by benchmarking each one on common environments and comparing them to prior implementations. Owing to its user-friendly and reliable nature, we chose Stable Baseline 3 as our RL library, which greatly simplified our research process.

OpenAI Gymnasium[some paper] is a toolkit for developing and comparing reinforcement learning algorithms, and it is widely used in the research community. Among its many advantages are its open-source nature and standardized environments, which facilitate the rapid testing and benchmarking of new algorithms. Additionally, it offers easy visualization and monitoring. Our decision to use the Gym library is based on its extensibility—from standard environments to highly customized ones—as well as its capability to integrate with tools like PyTorch and TensorFlow for GPU-accelerated computations. Furthermore, Gym provides the ability to construct stacked training environments for parallel training.

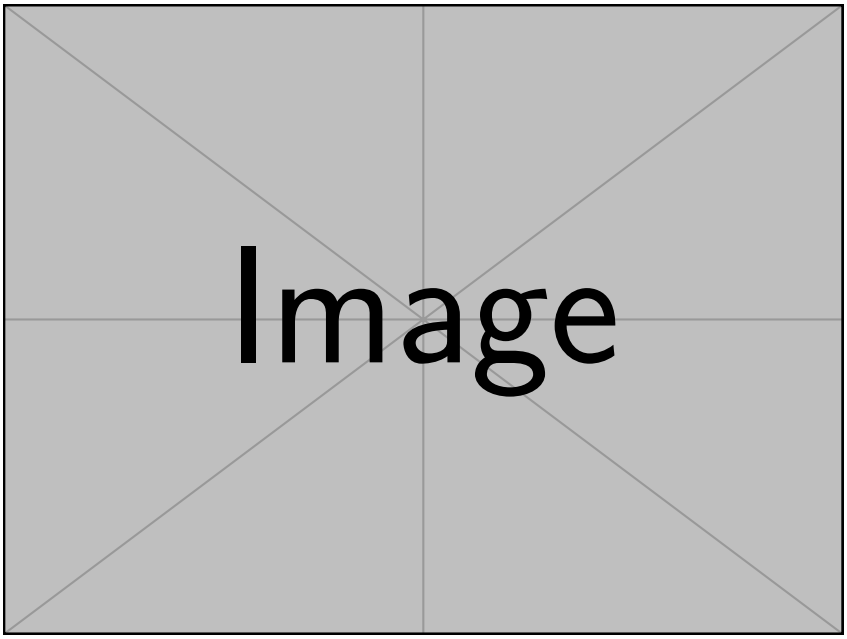


Figure 4.1: Interaction between reinforcement learning agent and customized environment

To construct the customized environment, we begin with the symbolic dynamic function that captures the nonlinear dynamics of the underactuated double pendulum. This function uses the current state observation from the simulation and the action determined by the control policy, producing a prediction of the next state via a Runge-Kutta integrator.

This predicted state is then input into the reward function, which yields a scalar output. This output is subsequently relayed to the SAC algorithm for policy evaluation and update. After these computations, the simulation provides an estimation of the current state, and the policy identifies the most probable action. Both of these values are then directed back to the dynamics function, initiating a new cycle.

It's widely recognized **some paper** that reinforcement learning algorithms tend to converge more effectively when using normalized state and action spaces. Therefore, we've designed a scaling mechanism to translate the state and action spaces from their normalized versions to real-world measurements. Users can choose to activate or deactivate this scaling. For example, while a normalized state resides within the interval $[-1,1]$, we might wish to map it to real-world measurements $[\text{pos1}, \text{pos2}, \text{vel1}, \text{vel2}]$, such as pos1 in $[0, 2\pi]$, pos2 in $[-\pi, \pi]$, and velocities vel1 and vel2 in $[-20, 20]$ rad/s, while keeping the torque within $[-\tau_{\text{limit}}, \tau_{\text{limit}}]$. When this scaling mechanism is activated, it ensures that states and actions in the SAC algorithm always

stay within the $[-1, 1]$ boundary, which often leads to faster convergence. The mapping relation of the scaling mechanism is demonstrated in the picture below:

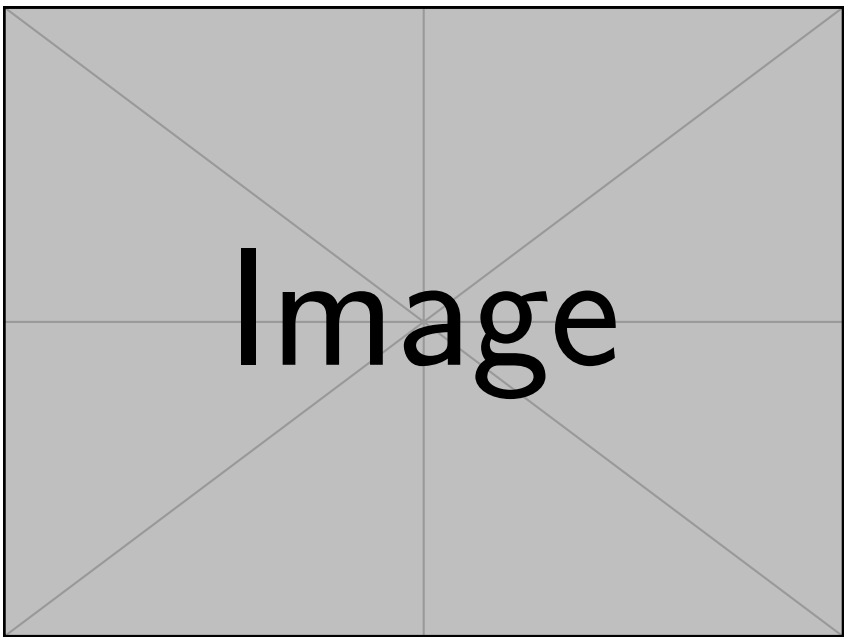


Figure 4.2: mapping relation of the scaling mechanism

The logic of the pipeline of the interaction of customized environment is described in the picture below.

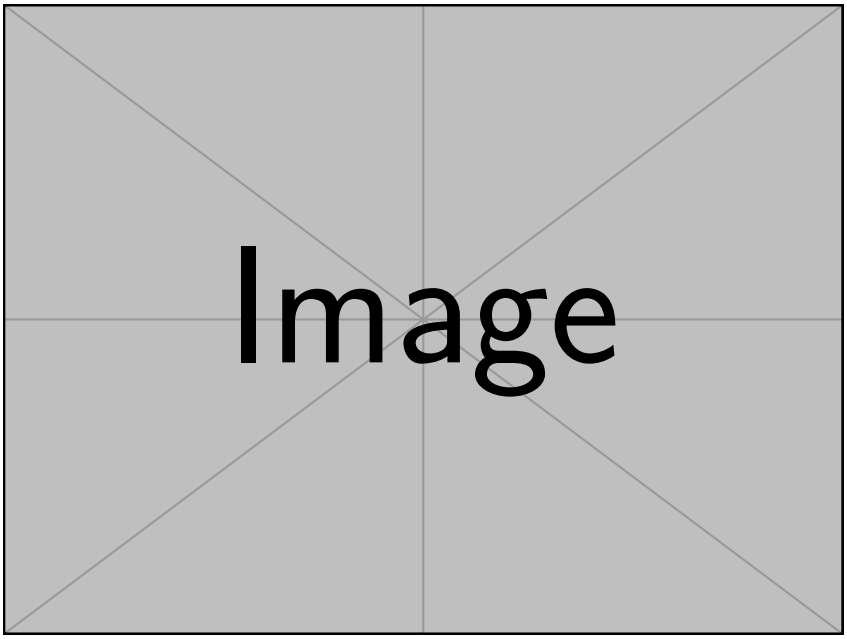


Figure 4.3: logic of the interaction

4.2 Training phase

The training phase started immediately after we finished setting up the training pipeline. During the training of the SAC controller for both the acrobot and pendubot, our primary focus was on tuning several key hyperparameters. These encompassed the learning rate, control frequency, episode length, and learning timestep. We invested much effort in adjusting hyperparameters for the reward function and the LQR controller, given their pivotal roles in the learning process, as detailed in Table 4.1.

We set the learning rate at 0.01 to promote effective learning and adaptation. Additionally, we configured the control frequency of the simulation to 100Hz, ensuring frequent updates and heightened responsiveness in the control process. We selected an episode length of 1000 for both the Acrobot and Pendubot, translating to 10-second-long episodes, to provide sufficient exploration and learning opportunities. To harness the full training potential, we executed a total of 2e7 learning time steps for the pendubot and 5e7 for the acrobot, allowing the agent to accumulate vast experience and enhance its performance.

Robot	Quadratic Reward	Constant Reward	LQR
Pendubot	Q_1 = 8.0 Q_2 = 5.0 Q_3 = 0.1 Q_4 = 0.1 R = 1e-4	$r_{line} = 500$ $r_{vel} = 0.0$ $r_{LQR} = 1e4$	Q_1 = 1.92 Q_2 = 1.92 Q_3 = 0.3 Q_4 = 0.3 R = 0.82
Acrobot	Q_1 = 10.0 Q_2 = 10.0 Q_3 = 0.2 Q_4 = 0.2 R = 1e-4	$r_{line} = 500$ $r_{vel} = 1e4$ $r_{LQR} = 1e4$	Q_1 = 0.97 Q_2 = 0.93 Q_3 = 0.39 Q_4 = 0.26 R = 0.11

Table 4.1: Hyper parameters used for the SAC training and the LQR controller.

The learning curve for a total of 2e7 timesteps for the pendubot is illustrated below.As depicted in the figure, from 0 to 1e7 timesteps, the learning curve experiences a gradual ascent. Between 1e7 and 1.2e7 timesteps, the learning curve surges rapidly, peaking in reward around one million. After 1.2e6 timesteps, the curve begins to stabilize with occasional fluctuations, and no substantial increase in reward is observed.

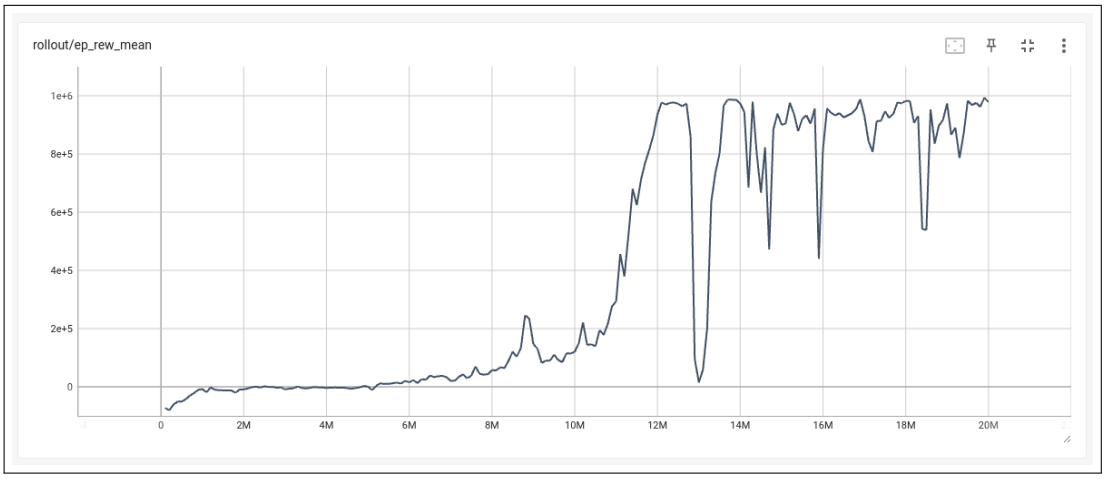


Figure 4.4: pendubot learning curve

As shown in the learning curve over a total of 3e7 timesteps for the acrobot, the curve experienced a relatively steady growth until 1.8e7 timesteps. After a sudden drop in reward between 1.8e7 and 2e7 timesteps, the learning curve began to increase drastically, with two major fallbacks. By

3e7 timesteps, the learning curve hadn't reached its peak. We extended the learning period to 5e7 using a warm start from the model obtained at 3e7 timesteps, and the learning curve began to stabilize around 4e7 time steps.

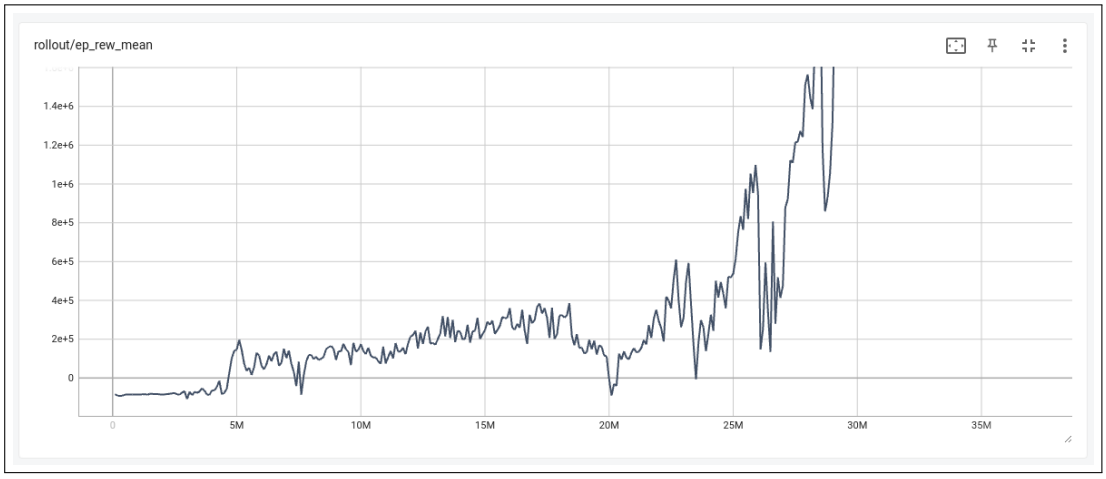


Figure 4.5: acrobot learning curve

In conclusion, the training for the pendubot converges more quickly than that for the acrobot. Additionally, based on the training curves, the pendubot's training process is more stable than the acrobot's.

4.3 Simulation results

In this section, the simulation results for the acrobot and pendubot are presented separately. The model trained using the SAC algorithm is utilized during the swing-up stage and switches to the LQR controller when nearing the upright position. The primary success criteria for both swing-up and stabilization involve swinging up the double pendulum and maintaining its stability around the upright position for a prolonged period. Therefore, a total simulation time of 10 seconds is used. If the double pendulum fails to swing up within these 10 seconds, the result is deemed unsuccessful. Similarly, if the double pendulum loses stability within this time frame, the outcome is still considered a failure.

4.3.1 Pendubot simulation in ideal environment

A testing environment, identical to the customized learning environment, has been established to validate the training results and accurately replicate the agent's behavior learned during the reinforcement learning process. The pendulum is initialized at its lowest point with zero velocity and begins its swing-up using the control policy derived solely from the SAC.

As depicted in the figure, the swing-up time is under 1 second. After this, the state of the pendubot enters the Region of Attraction (ROA) of the LQR controller. The transition between the two controllers is both seamless and effective, with the system stabilizing towards the desired state under the LQR controller's influence. This validates the effectiveness of the ROA method for the LQR takeover.

A significant highlight from this successful simulation is the impressively short swing-up time, despite having strict torque limit in place. However, a concerning feature observed from this simulation is the noisiness of the input control signal. The torque alternates signs rapidly, and the gradient of the torque tends to have a high absolute value.

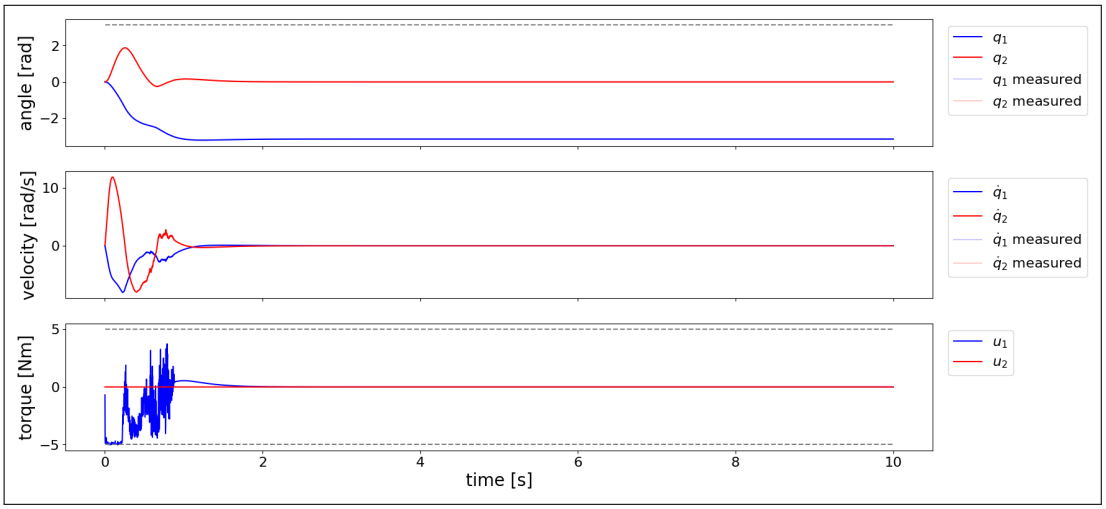


Figure 4.6: pendubot simulation result

4.3.2 Acrobot simulation in ideal environment

Just like the pendubot simulation, the acrobot setup is tested in an ideal environment identical to its training environment. The acrobot begins its swing from the downward position with the goal of stabilizing around its highest point.

The accompanying image depicts a successful swing-up and stabilization within a 10-second window. The swing-up process takes about 2 seconds before the LQR effectively assumes control, maintaining an asymptotic stability around the target state with minimal fluctuations.

In comparison to the pendubot's simulation curves, the swing-up time for the acrobot is roughly twice as long. This aligns with the notion that controlling the acrobot is a more challenging task than the pendubot. A shared drawback observed in both simulations is the lack of torque smoothness. For the acrobot, the input torque exhibits several significant jumps from one torque limit to the other, which might cause difficulties when translating to real-world hardware control.

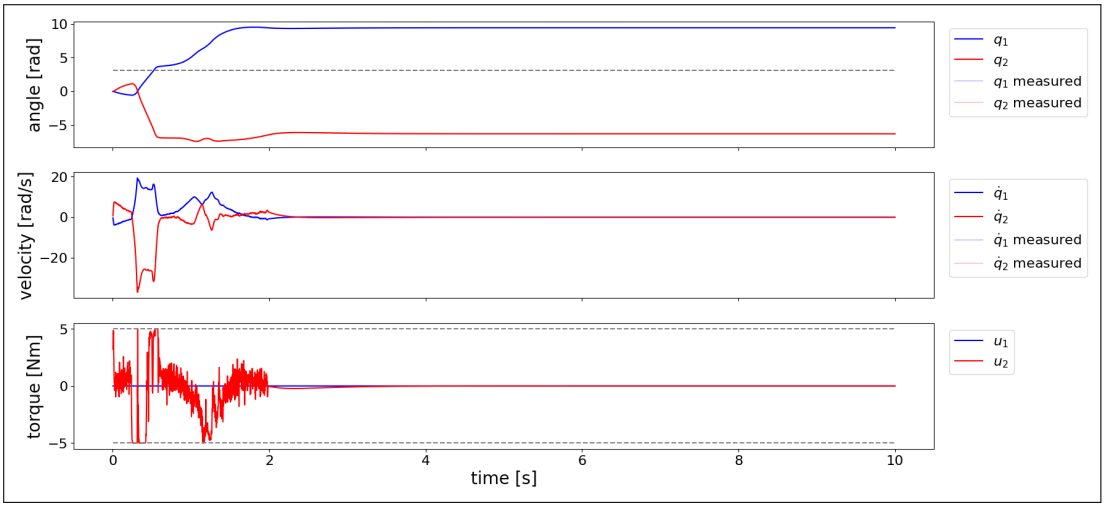


Figure 4.7: acrobot simulation result

5 Experiment: hardware system

In this chapter, we discuss experiments conducted on the hardware system. The content is organized into four sections. The first section provides an overview of the hardware setup for the double pendulum system. The second section delves into the system identification of the hardware. The third section details our approach to addressing the sim-to-real gap challenge. The final section presents the successful outcomes of our hardware experiments.

5.1 Hardware setup

Mechanically, the double pendulum system is a straightforward 2-R linkage. The first revolute joint attaches to the base, while the second one connects the two links. Quasi-direct drive motors are mounted on each joint to provide torque, and a counterweight is positioned at the end of the second link.

Our mechanical design for the double pendulum underwent two iterations. In the initial design, the base consisted of a bent aluminum plate, and the links featured a sandwich structure with aluminum on the outside and engineering plastic on the inside. These links were of homogeneous size, meaning that the cross-sectional areas at the link intersections were consistent. We abandoned this design due to rotational imbalances. Specifically, the rotation plane of the links wasn't consistently perpendicular to the motor axes, leading to vibrations and accelerated wear during regular use. In extreme cases, when the links rotated at high angular velocities, this imbalance was exacerbated. This not only resulted in the links bending but also led to catastrophic system failures. Such failures presented significant safety risks to personnel.

In the second iteration, we addressed the issues encountered in the previous design, leading to two major modifications. First, we replaced the aluminum-plastic combination with a carbon fiber-foam blend. While the incorporation of carbon fiber marginally increased the cost, it significantly boosted the yield strength. Second, we introduced triangular-shaped links with central cutouts. This design, while lightweight, also substantially enhanced the yield strength due to the changed intersections. Overall, this iteration resulted in a mechanical structure that is considerably more reliable than the first.

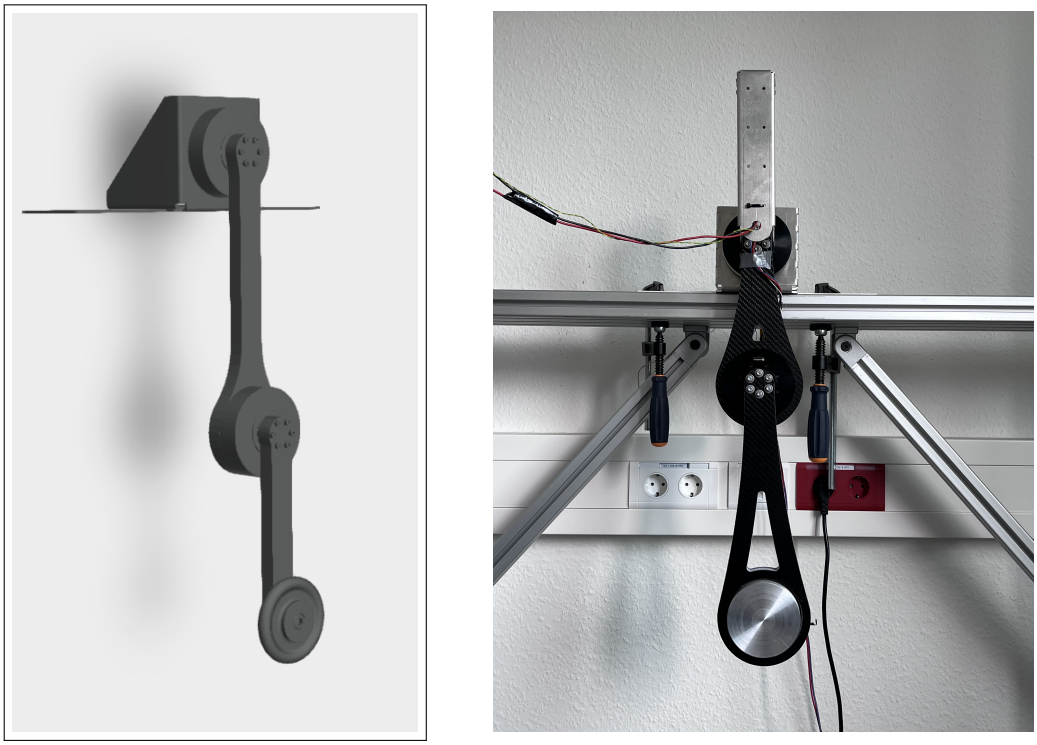
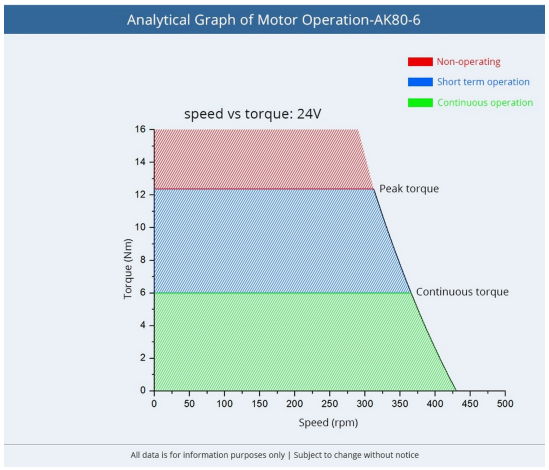


Figure 5.1: Double pendulum setup in CAD (version 1) and in real world (version 2)

For the quasi-direct drive motors, we selected the AK80-6 V100 motors from the company CubeMars. This motor's design facilitates easy mounting from both the front and rear ends. As depicted in Figure5.2(b), the motor's maximum torque during continuous operation is 6 Nm, which aligns well with our torque limit of 5 Nm. Additionally, the motor is designed for compatibility with both serial bus and CAN bus, simplifying the development process.

(a) AK80-6 V100 motor



(b) Speed-torque diagramm

Figure 5.2: Quasi direct drive motor

For communication, We have chosen CAN in this implementation. The Controller Area Network (CAN) bus is a robust, flexible, and efficient communication protocol that has been employed in various applications. There are many advantages to using the CAN bus for control. The CAN bus provides error checking and fault confinement capabilities. It operates in real-time, enabling high control frequencies with relatively simple wiring. In our configuration, there is only one master node (the PC) and two slave nodes (two motors). The control loop simply consists of a CAN-to-USB converter, one CAN high cable, and one CAN low cable, with a termination resistor of 120Ω at both ends. The detailed connection is depicted in the figure below:

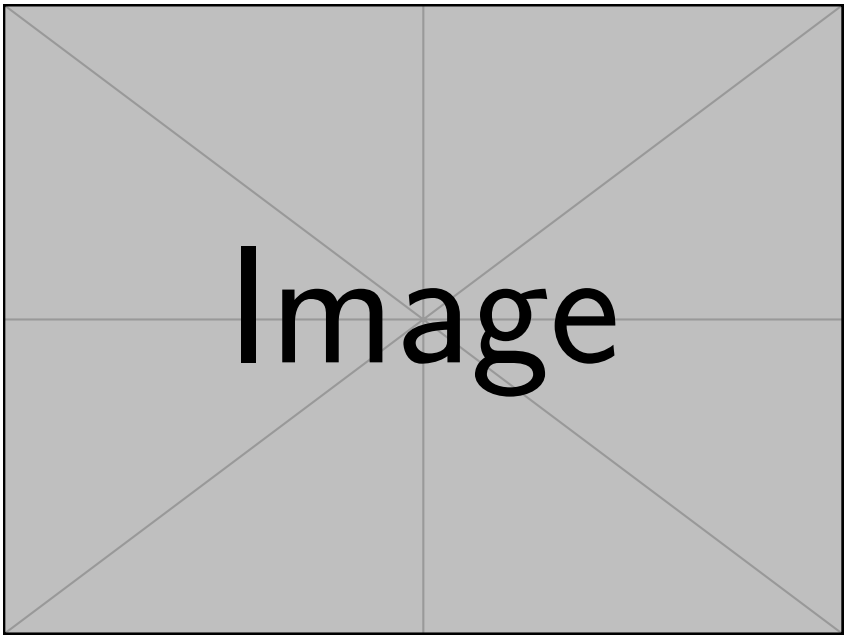


Figure 5.3: CAN connection

To ensure a higher control frequency of around 500 Hz, we chose the CAN-USB/2 product from ESD GmbH Hannover. This specialized CAN-to-USB interface utilizes USB 2.0, which supports a data rate of 480 Mbit/s. Its CAN capability is 1 Mbit/s, in accordance with ISO 11898-2. Additionally, it supports the SocketCAN interface included in the Linux Kernel 2.6, making it easier to use in a Linux development environment.

Figure 5.4: blue box

Due to accidents that occurred during testing with the mechanical systems from the first iteration, several safety protocols have been implemented to ensure the safety of both human lives and equipment. Four major measures have been taken.

An emergency stop is connected directly to the 24V power source. If the behavior of the double pendulum deviates from the expected range during testing, the power supply can be manually cut off immediately. The energy in the mechanical system will dissipate rapidly, and the system will return to its initial state automatically.

In scenarios where the system is moving at a very high speed when the emergency stop is engaged, the motors attached to the revolute joints act as generators, dissipating the mechanical energy from the system's motion. The generated current is fed back into the circuit, and in extreme cases, it could overload the power supply. To counteract this, a capacitor is connected to the power supply to absorb any electrical surge resulting from a sudden stop.

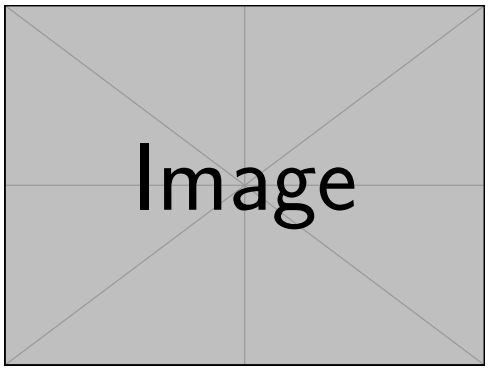


Figure 5.5: picture of the capacitor

Further, speed and position limits have been set at the software level. As the links of the pendulum can experience vibrations and rotational imbalances at high speeds, potentially leading to structural disassembly, a speed limit of 20 rad/s has been established. Any speed exceeding this value will trigger a full system stop, equivalent to pressing the emergency stop button. The position limit is set to 2π for both pos1 and pos2. Excessive rotations could cause the CAN and power cables to become entangled, leading to interference and potential cable damage.

Lastly, to guard against unpredictable system failures, a custom protective cage has been constructed using aluminum profiles and thick acrylic boards. This cage fully encloses the double pendulum hardware, significantly reducing the risk of accidents.

These measures have been instrumental in minimizing potential hazards during testing and operation.

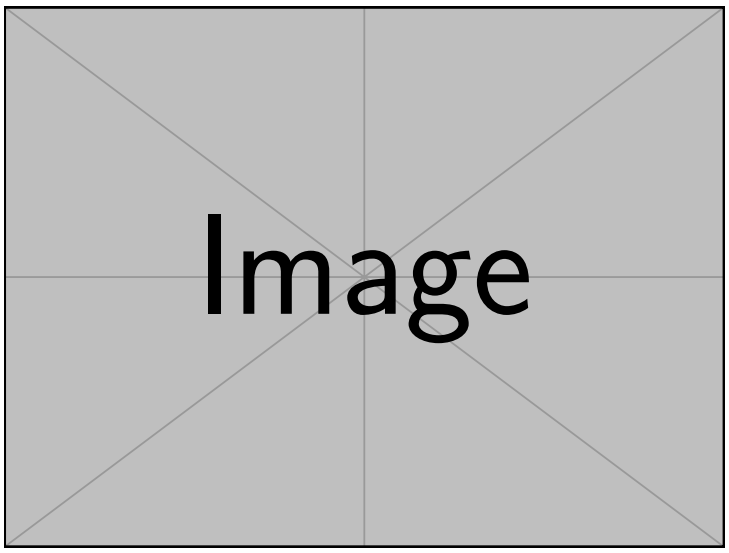


Figure 5.6: a overview of experiment setup

5.2 System identification

System identification is the process of deriving mathematical models of dynamic systems from observed input-output data. This method is fundamental in control theory, used to analyze, predict, and control the behavior of real-world systems. After setting up the hardware system and before transitioning the working model from a successful simulation to the real system, it's necessary to undergo a system identification process. This helps ascertain the real-world parameters governing the dynamics of the double pendulum.

Out of all model parameters, 15 are selected. While the naturally provided parameters g and g_r are held constant, the easily measurable parameters l_1 and l_2 are considered independent. The remaining system parameters, which are

$$m_1 r_1, m_2 r_2, m_1, m_2, I_1, I_2, I_r, b_1, b_2, c_{f1}, c_{f2}$$

need to be identified. The ultimate objective is to discern the parameters of the dynamic matrices present in the equations of motion.

$$M\ddot{q} + C(q,\dot{q})\dot{q} - G(q) + F(\dot{q}) - Du = 0 \tag{5.1}$$

By running excitation trajectories on the actual hardware, data tuples in the form $(q, \dot{q}, \ddot{q}, u)$ can be collected. To determine the most accurate system parameters, one can leverage the linearity of the dynamic matrices M, C, G, and F in relation to the independent model parameters. Consequently, a least squares optimization can be performed on the recorded data, relative to the dynamics equation.

The identified model parameters are shown in table below:

[here include a table]

5.3 Sim2Real problem

Transferring working models from simulation to real systems to produce similar performance has always been a challenge in controller design. This challenge is even more pronounced in model-free reinforcement learning for several reasons.

Firstly, model-free reinforcement learning relies solely on interaction with the environment to gain experience and select actions. While a simulation environment is merely a simplification of the real-world scenario, the agent in simulation might not capture all the factors, such as friction, sensor noise, or real-world dynamics, accurately. Therefore, a control policy optimized for a simplified model might not perform as expected in the more intricate real world.

Secondly, many simulations operate in discrete time and space, whereas the real world functions continuously. In our implementation, the control frequency presents a significant challenge. We use a control frequency of 100 Hz in simulation; however, it does not suffice in the real system. To enhance performance, we increased the control frequency to 400 Hz when experimenting on the real system, and this adjustment yielded positive results.

Thirdly, uncertainties in the environment can lead to substantial complications when attempting to apply learned strategies or actions. Generally speaking, the uncertainty of the environment is addressed by the robustness of the model. However, for highly chaotic systems like the

pendubot or acrobot, even slight measurement errors can result in significant deviations from the planned behavior. Our model for controlling the pendubot and acrobot requires a higher level of robustness or a more effective sim2real transfer method.

5.3.1 Validation with noisy simulation

Our approach to addressing the sim2real challenge involves training multiple agents using the SAC algorithm under similar setups. Subsequently, we validate them in a noisy simulation. Only those agents that prove robust against perturbations in this noisy environment proceed to real system testing. Any agent failing these noisy simulation tests is deemed insufficiently robust and is discarded.

Throughout our experiments with real systems, we identified four critical factors contributing to successful swing-up and stabilization. Of these, friction emerged as the most significant. This is because our agent training was conducted in an ideal environment devoid of friction, thereby making friction the primary differentiator between the simulation and real-world conditions. To counter this, we adopted a friction compensation approach, beginning with modeling based on Coulomb's friction.

Coulomb's friction model is among the simplest existing friction models. It possesses two primary advantages: it's independent of both the contact area and the relative velocity. The frictional force (F_f) is directly proportional to the normal force (F_n) between the surfaces, represented by:

$$F_{fi} = c_{fi} \arctan(100\dot{q}_i), \quad i = 1,2 \tag{5.2}$$

where c_f is the coefficient of kinetic friction. This frictional force opposes the relative motion between the surfaces. To neutralize this force, friction compensation applies torque in the same direction as the angular motion, energizing the system to mitigate friction's effects.

Though friction coefficients c_{f1} and c_{f2} were determined during the system identification phase, they seemed imprecise during real system tests. Consequently, we resorted to free fall tests to estimate the coefficients, fine-tuning them manually until the system behaved conservatively under friction's influence.

The second most critical factor is measurement noise. While the AK80-6 motors use built-in encoders to measure the position difference from the initial position with high accuracy, velocity measurement, which is derived from the first derivative of the position measurement, tends to

have a relatively high error. In the simulation environment, we assume zero measurement error. However, in real-world applications, this error becomes significant.

To address this issue, we model the measurement error as a normal distribution where the mean corresponds to the true velocity value and the standard deviation is manually adjustable. Let's consider the measurement noise vector $\varepsilon = \left[\delta\text{pos1}, \delta\text{pos2}, \delta\text{vel1}, \delta\text{vel2}\right]^T$. We model this noise as following a multivariate normal distribution, given by:

$$\varepsilon \sim \mathcal{N}(\mu, \Sigma) \tag{5.3}$$

where μ is the mean vector (which represents the true values in the absence of noise), and Σ is the 4×4 covariance matrix that captures the spread or uncertainty of the noise in each dimension. Since the measurements of the four states are considered independent of each other, Σ is diagonal.

Latency is the third significant factor. Since any communication system requires time to transmit and receive data, and programs also take time to execute, latency is inevitable. This latency poses a substantial risk to control systems, especially those based on reinforcement learning. This is because reinforcement learning is grounded in the principles of Markov Decision Processes (MDPs) which adhere to the Markov property. This property dictates that the future state of a process is contingent only on the current state and action, not on the sequence of states that led to it. Latency undermines the Markov property by causing state mismatches and creating dependencies on historical data. If not addressed, this can have severe consequences.

The fourth factor to consider is torque responsiveness. We observed that when confronted with rapidly alternating control signals with significant differences between time steps, the motor struggles to produce torque that matches the control signal. In other words, the motor cannot respond timely to changes in torque. This lag means that the controller cannot operate at its full potential due to hardware constraints. Some potential solutions to this include reducing the control frequency and increasing torque smoothness.

In summary, we take into account friction, measurement error, latency, and torque responsiveness in the noisy simulation to select the most suitable agent.

As shown in the figure below, a trained pendubot agent successfully executed a swing-up and stabilization in the noisy simulation environment, indicating its robustness is sufficient for real-world application.

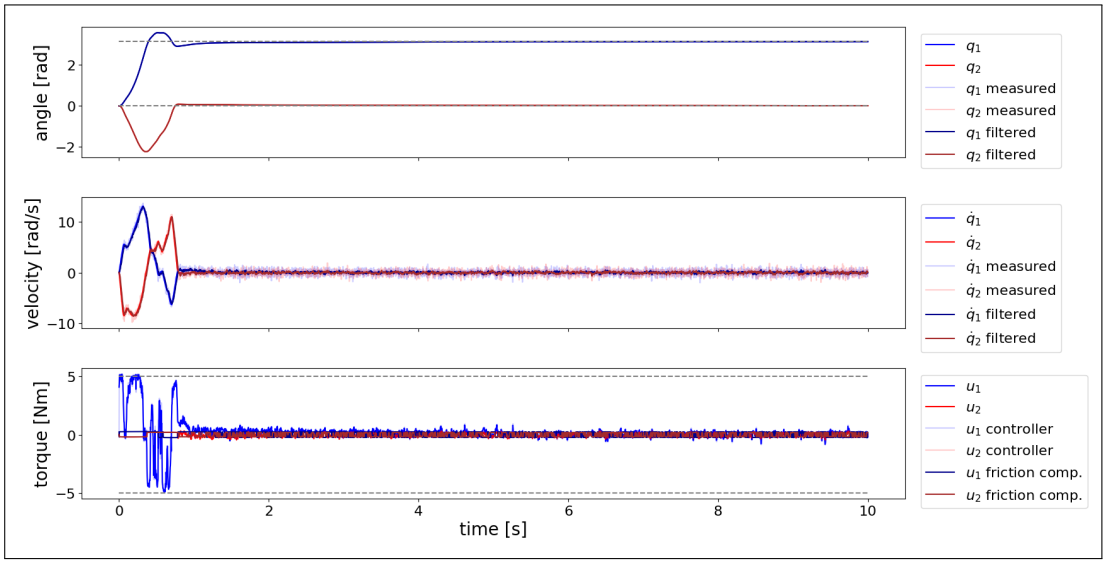


Figure 5.7: pendubot noisy simulation result

5.3.2 Training with domain randomization

Hope we have a good result.

5.4 Real hardware results

In this section, we present the results from the real hardware tests. Only agents trained for the pendubot setup successfully passed the noisy simulation check, so results are limited to this setup. The complete real-world test procedure is as follows: power up, manually set the initial state, release the emergency button, run the initialization script (which enables the motor, sets the current position to zero, and tests the CAN connection), confirm the start of the test, record video, draw plots, and exit the test. This procedure is illustrated in the flow chart below:

A remote testing system has been established to allow global access and sharing of our double pendulum testing hardware. This infrastructure was developed for the IJCAI 2023 competition, RealAIGym. Several groups have tested their algorithms on this remote system, including teams from the University of Padova and the Technical University of Darmstadt.

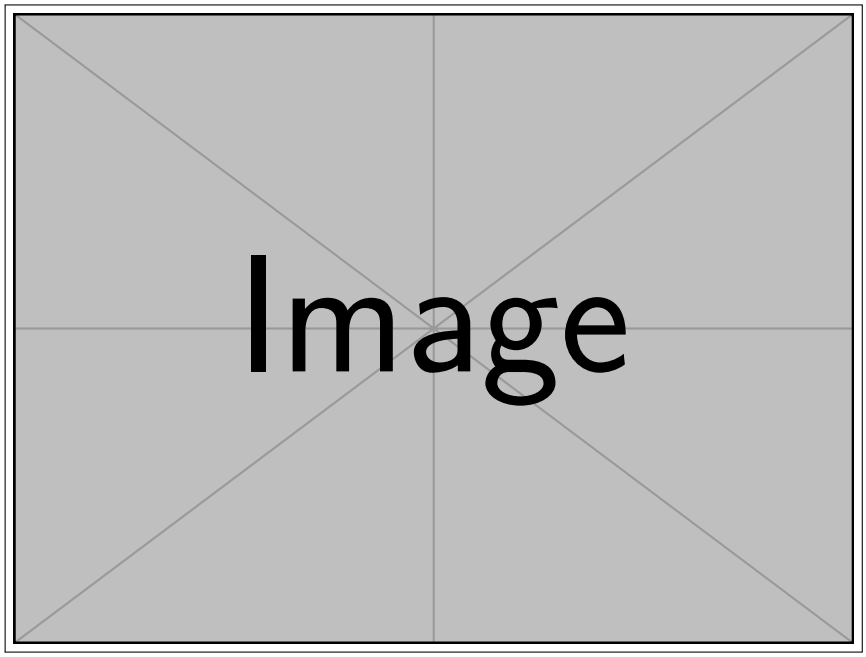


Figure 5.8: Real hardware system testing procedure

5.4.1 pendubot results

Below is an example of one of our successful tests. [some explanation] We executed a sequence of 10 tests to determine the success rate and other key metrics for evaluating performance and robustness. These results will be discussed in the next chapter.

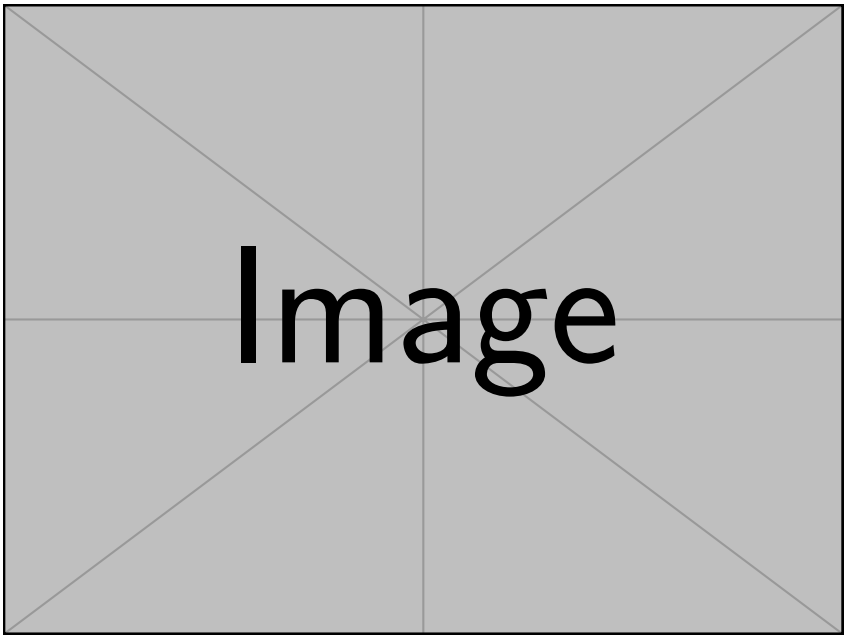


Figure 5.9: Pendubot results on real systems

5.4.2 acrobot results

acrobot:

6 Discussion

In this chapter, we delve into the results of experiments conducted both in simulation and on the real system. The chapter is structured as follows: The first subsection provides a comprehensive introduction to the leaderboard metrics employed. Sections 6.2, 6.3, and 6.4 discuss the three facets of leaderboard analysis, namely simulation, robustness, and real hardware, respectively. The final section wraps up our current findings and hints at directions for future research.

6.1 Introduction to leaderboard metrics

To compare the performances of various controllers developed for the double pendulum testbench, we have established three distinct leaderboards: the simulation leaderboard, the robustness leaderboard, and the real system leaderboard. Each of these leaderboards is further subdivided into two categories based on the pendubot and acrobot setups.

6.1.1 Performance Leaderboard in Simulation and Real system

In our evaluation of controller performance, we employ a set of metrics that go beyond just measuring task success. They delve deeper into the nuances of controller operation. For experiments conducted both in simulation environments and on real hardware, the performance evaluation metrics remain consistent. These measures span from assessing the fundamental success of a swing-up maneuver to understanding the intricate details of energy and torque usage. Below is a detailed breakdown of each metric:

- **Swingup Success** c_{success}: Determines if the end-effector successfully remains above the predefined threshold by the simulation's conclusion.
- **Swingup Time** c_{time}: Measures the duration taken for the pendubot or acrobot to achieve and maintain its position above the threshold line. The metric only considers the swingup successful if the end-effector remains above the threshold until the simulation's end.
- **Energy** c_{energy}: Quantifies the total mechanical energy expended during the task.
- **Max Torque** $c_{\tau,\text{max}}$: Captures the highest torque applied at any point during the task.

- **Integrated Torque** $c_{\tau,\text{integ}}$: Represents the cumulative torque applied throughout the task's duration.
- **Torque Cost** $c_{\tau,\text{cost}}$: A quadratic metric that weighs the torques used, defined as $c_{\tau,\text{cost}} = \sum u^T R u$, where $R = 1$.
- **Torque Smoothness** $c_{\tau,\text{smooth}}$: Reflects the variability or fluctuations in the torque signals by measuring their standard deviation.
- **Velocity Cost** $c_{\text{vel, cost}}$: A metric assessing the joint velocities achieved, computed as $c_{\text{vel}} = \dot{q}^T Q \dot{q}$, with Q being the identity matrix.

We utilize the subsequent criteria to determine the cumulative RealAI Score based on the specified formula:

$$\begin{aligned} S = c_{\text{success}} \Bigg(& w_{\text{time}} \frac{c_{\text{time}}}{n_{\text{time}}} + \\ & w_{\text{energy}} \frac{c_{\text{energy}}}{n_{\text{energy}}} + w_{\tau,\text{max}} \frac{c_{\tau,\text{max}}}{n_{\tau,\text{max}}} + w_{\tau,\text{integ}} \frac{c_{\tau,\text{integ}}}{n_{\tau,\text{integ}}} + \\ & w_{\tau,\text{cost}} \frac{c_{\tau,\text{cost}}}{n_{\tau,\text{cost}}} + w_{\tau,\text{smooth}} \frac{c_{\tau,\text{smooth}}}{n_{\tau,\text{smooth}}} + w_{\text{vel, cost}} \frac{c_{\text{vel, cost}}}{n_{\text{vel, cost}}} \Bigg) \end{aligned} \tag{6.1}$$

The weights and normalizations are:

Criteria	Normalization n	Weight w
Swingup time	10.0	0.2
Energy	100.0	0.1
Max. Torque	6.0	0.1
Integrated Torque	60.0	0.1
Torque Cost	360	0.1
Torque Smoothness	12.0	0.2
Velocity Cost	1000.0	0.2

Table 6.1: Weights and normalizations for performance leaderboards

In the simulation experiments, the pendubot is modeled using a Runge-Kutta 4 integrator with a timestep of $dt = 0.002s$ over a span of $T = 10s$. We initiate the pendubot in a hanging down configuration, represented as $x_0 = (0,0,0,0)$, and aim to reach the unstable fixed point of the upright configuration, denoted as $x_g = (\pi,0,0,0)$. The double pendulum is deemed to

have achieved its upright position once the end-effector surpasses the threshold line situated at $h = 0.45m$, with the origin being the mounting point.

When it comes to real hardware experiments, there's a torque limit of 0.5 Nm on the passive joint, which serves to offset the motor's friction. The actuators can operate with a control frequency as high as 500Hz, and each experiment lasts for 10 seconds. The pendubot starts from a hanging down position, with the objective being the unstable fixed point in the upright configuration. Successful attainment of the upright position is confirmed when the end-effector crosses the threshold line set at $h = 0.45m$, measured from the mounting point's origin.

6.1.2 Simulation Robustness Leaderboard

- **Model Inaccuracies** c_{model}: Model parameters determined through system identification are never perfectly accurate. To evaluate potential inaccuracies, we modify each model parameter individually within the simulator, while keeping the controller's parameters unchanged.
- **Measurement Noise** $c_{vel,noise}$: Controller outputs heavily rely on the captured system state. Particularly with the Quick Double Derivatives (QDDs), online velocity measurements can be noisy. It's crucial for successful transferability that a controller can handle this inherent noise. We test controllers with both the presence and absence of a low-pass noise filter.
- **Torque Noise** $c_{\tau,noise}$: It's not only the measurements that can exhibit noise. Sometimes the torque output from the controller might deviate from the intended value.
- **Torque Response** $c_{\tau,response}$: The torque requested by the controller is dynamic and can vary throughout execution. Due to mechanical limitations, the motor might not always adjust immediately to abrupt torque changes, leading to overshooting or undershooting the desired torque value. We model this with the equation $\tau = \tau_{t-1} + k_{\text{resp}}(\tau_{\text{des}} - \tau_{t-1})$, where τ_{des} is the desired torque. In this model, a k_{resp} value of 1 indicates flawless torque response, while any deviation from 1 indicates imperfect motor responses.
- **Time Delay** c_{delay}: Operating in a real-world environment inevitably introduces time delays due to communication lag and system reaction times. It's essential to account for these when evaluating controller performance.

The above criteria are employed to compute the comprehensive RealAI Score using the given formula:

$$S = w_{model}c_{model} + w_{vel,noise}c_{vel,noise} + w_{\tau,noise}c_{\tau,noise} + w_{\tau,response}c_{\tau,response} + w_{delay}c_{delay} \tag{6.2}$$

The weights are:

$$w_{model} = w_{vel,noise} = w_{\tau,noise} = w_{\tau,response} = w_{delay} = 0.2 \tag{6.3}$$

6.2 Interpretation of simulation leaderboard

In the table below, you can see the performance leaderboard results for both the pendubot and acrobot in simulation experiments. Three major types of controllers are listed for comparison. The SAC+LQR controller is our design, and it is based on the model-free reinforcement learning method. MC-PILCO stands for Monte Carlo Probabilistic Inference for Learning Control. It is a model-based reinforcement learning method that uses probabilistic models to predict the system's dynamics and employs Monte Carlo methods to optimize control policies based on these predictions. tvLQR is an extension of the standard Linear Quadratic Regulator (LQR) control design, tailored for systems with time-dependent state-space matrices or where the optimal control needs to vary over time. It represents the optimal control method.

Criteria	SAC+LQR		MC-PILCO		tvLQR	
	Pendubot	Acrobot	Pendubot	Acrobot	Pendubot	Acrobot
Swingup Success	success	success	success	success	success	success
Swingup time [s]	0.65	2.06	1.43	1.1	4.2	3.98
Energy [J]	9.4	29.24	12.67	9.81	9.06	10.92
Max. Torque [Nm]	5.0	5.0	2.4	2.82	2.82	5.0
Integrated Torque [Nm]	2.21	4.57	3.48	1.27	2.57	2.27
Torque Cost [N²m²]	8.58	12.32	7.77	2.27	2.0	2.47
Torque Smoothness [Nm]	0.172	0.954	0.07	0.057	0.031	0.077
Velocity Cost [m²/s²]	44.98	193.78	94.68	242.44	137.31	100.34
RealAI Score	0.801	0.722	0.891	0.869	0.827	0.8

Table 6.2: Performance scores of various controllers for pendubot and acrobot experiments.

6.3 Interpretation of robust leaderboard

Criteria	SAC+LQR		MC-PILCO		tvLQR	
	Pendubot	Acrobot	Pendubot	Acrobot	Pendubot	Acrobot
Model inaccuracy [%]	71.9	76.7	45.2	40.5	75.2	59.0
Velocity noise [%]	100.0	71.4	90.5	66.7	100.0	95.2
Torque noise [%]	100.0	100.0	100.0	81.0	100.0	100.0
Torque response [%]	100.0	100.0	100.0	90.5	100.0	100.0
Time delay [%]	76.2	61.9	90.5	19.0	100.0	76.2
Overall Score	0.896	0.820	0.852	0.595	0.950	0.861

Table 6.3: Robustness scores of various controllers for pendubot and acrobot experiments.

6.4 Interpretation of real system leaderboard

This section is about explaining the hardware results.

Criteria	SAC+LQR		MC-PILCO		tvLQR	
	Pendubot	Acrobot	Pendubot	Acrobot	Pendubot	Acrobot
Swingup Success	4/10	insuccess	10/10	10/10	8/10	10/10
Swingup time [s]	0.67	-	1.37	1.55	4.12	4.03
Energy [J]	37.12	-	11.66	17.95	34.02	13.75
Max. Torque [Nm]	5.0	-	4.99	5.0	5.0	2.98
Integrated Torque [Nm]	24.87	-	3.72	5.93	19.06	5.61
Torque Cost [N^2m^2]	78.7	-	8.93	11.73	51.88	3.26
Torque Smoothness [Nm]	0.774	-	0.54	0.671	0.643	0.108
Velocity Cost [m^2/s^2]	114.04	-	84.61	118.38	242.34	109.77
Best RealAI Score	0.767	-	0.843	0.82	0.695	0.822
Average RealAI Score	0.298	-	0.839	0.817	0.547	0.821

Table 6.4: Real hardware performance scores of multiple controllers for pendubot and acrobot experiments.

6.5 Conclusion and future work

This section is to talk about things to be done.

Appendix

A An appendix

You can structure appendices, just like your thesis, with the `\chapter`, `\section`, and `\subsection` commands. Referencing also works as usual.

If your thesis does not contain an appendix, comment out the creation of the appendix at the appropriate place in the `Thesis.tex` file.