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 and 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 Robotics Innovation Center, DFKI GmbH. Prof. Dr. Frank Kirchner(University of Bremen)
Supervisor	Robotics Innovation Center, DFKI GmbH. Felix Wiebe, Prof. Dr. Shivesh Kumar(Chalmers University of Technology) Laboratory for Product Development and Lightweight Design Akhil Sathuluri, Maximilian Amm (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

Abstract

The aim of our project is to have swing-up and stabilization tasks performed for underactuated double pendulum systems. For the swing-up task, a reinforcement learning-based control method is employed, while an LQR controller is utilized for stabilization around the highest point. The entire project is divided into two phases: simulation and real-world testing. In the simulation phase, SAC-based agents are successfully trained in an ideal environment for both the pendubot and acrobot, and they are subsequently combined with an LQR controller and tested in an ideal simulation, yielding positive results. During the real-world hardware phase, a noisy simulation is established to mimic real-world features, and controllers that pass validation in this noisy simulation are then tested on real hardware. To address the sim-to-real issue, a noisy training process based on domain randomization is implemented to enhance robustness. The SAC+LQR controller, which demonstrates a success rate of 40% only on the pendubot, is evaluated in terms of performance and robustness based on the results from both simulation and real-world tests, with the quantitive metrics reflected on the respective leaderboards.

Background

This thesis is the master thesis of Chi Zhang, representing a collaborative effort between the Laboratory for Product Development and Lightweight Design (LPL) at the Technical University of Munich (TUM) and the Underactuated Lab at the Robotics Innovation Center (RIC) of the German Research Center for Artificial Intelligence GmbH (DFKI). This thesis builds upon prior work conducted by colleagues at DFKI RIC, expanding the controller range to include reinforcement learning-based control.

This work was supported by the federal state of Bremen for setting up the Underactuated Robotics Lab under Grant 201-342-04-2/2021-4-1.

Acknowledgement

I would like to express my heartfelt gratitude to my thesis advisors and supervisors, Prof. Dr. Markus Zimmermann, Prof. Dr. Frank Kirchner, Felix Wiebe, Akhil Sathuluri, Prof. Dr. Shivesh Kumar, and Maximilian Amm for their unwavering support, invaluable guidance, and expertise throughout my research journey.

I extend my sincere appreciation to the federal state of Bremen for their generous financial support, which made this research possible.

I am thankful to my dedicated co-workers, Shubham Vyas, Bingbin Yu, and my labmates Maximilian Albracht and V.P. Rodrigues, for their invaluable assistance and fruitful discussions. Special recognition goes to all the previous researchers on this project for their collaborative spirit and contributions.

I wish to convey my deep gratitude to my family, Wenbin Zhang and Yan Wang, as well as my friends, for their unwavering encouragement and understanding during this challenging endeavor.

I offer my heartfelt thanks to the Technical University of Munich for providing me with the opportunity to conduct this research.

This work would not have been possible without the support, guidance, and encouragement of all these individuals and institutions. Thank you.

Chi Zhang

15.11.2023

Project Note

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

My supersivor Felix Wiebe, Shivesh Kumar, Akhil Sathuluri 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

Felix Wiebe, Shivesh Kumar, Akhil Sathuluri

Contents

1 Introduction **1**
1.1 Motivation 1
1.1.1 Trajectory planning and tracking 2
1.1.2 Reinforcement Learning Based Control 4
1.2 Problem setup 6
1.3 Contribution 9
1.4 Content 10

2 State of the Art **13**
2.1 Theory 13
2.2 Related work 17

3 Methodology **23**
3.1 Soft actor critic 23
3.2 Linear quadratic regulator 26
3.3 Combining SAC and LQR with region of attraction 27
3.4 Reward shaping 30
3.5 Introduction to leaderboard metrics 33
3.5.1 Performance Leaderboard in Simulation and Real system 33
3.5.2 Simulation Robustness Leaderboard 35

4 Agent Training and Experiments in Ideal Simulation Environments **37**
4.1 Training setup in ideal environment 37
4.2 Training process in ideal environment 40
4.3 Results in ideal simulation 42
4.3.1 Pendubot simulation in ideal environment 43
4.3.2 Acrobot simulation in ideal environment 44
4.3.3 Self stablizing behaviour on both pendubot and acrobot 45

5 Experiments on Real Hardware **49**
5.1 Hardware setup 49
5.2 System identification 57
5.3 Sim-to-Real transfer 59
5.3.1 Validation with noisy simulation environment 60
5.3.2 Noisy training based on domain randomization 62

5.4 Results on real hardware 63
5.4.1 Agent selection procedure for real world tests 63
5.4.2 Pendubot results in real world 64
5.4.3 Acrobot results in real world 67

6 Discussion 73
6.1 Interpretation of simulation leaderboard 73
6.2 Interpretation of robust leaderboard 75
6.3 Interpretation of real system leaderboard 76

7 Conclusion and Future work 79
7.1 Conclusion 79
7.2 Future Work 80

Bibliography 83

1 Introduction

Nonlinear systems[44] [26], as their name suggests, do not exhibit linear relationships between inputs and outputs. This means their responses can't be simply predicted using linear equations. In the real world, nearly all systems display some degree of nonlinearity, which can manifest in various phenomena. For instance, in systems with multiple inputs and outputs (MIMO), the relationships between input and output variables can become interdependent, leading to coupling issues. A widely-discussed challenge that arises from nonlinearity is chaotic behavior[15][39], where minor changes in initial conditions can lead to vastly different outcomes.

Achieving precise control over nonlinear systems has long been a primary focus in the field of control theory. While linear systems, often represented by linear differential equations, can typically be solved quickly and analytically, nonlinear equations representing nonlinear dynamics usually lack closed-form solutions[49]. This necessitates the use of approximations and numerical methods. Executing these methods efficiently and accurately presents a central challenge in nonlinear control. Throughout history, control engineers have devised a wide range of strategies to manage a variety of complex systems. The rise of robotics in recent decades has introduced new methods specifically tailored to address the challenges presented by nonlinearities.

1.1 Motivation

Robots are purposefully engineered as programmable mechanical structures, enabling them to perform a variety of tasks, either autonomously or partial under human supervision. Typical tasks include mobility, manipulation, and active interaction with their surroundings.

In the field of modern robotics, the mechanical systems are highly complex and nonlinear, posing significant challenges for precise and effective control. However, with the advancement of modern control methodologies and the increasing capabilities of artificial intelligence, numerous innovative control approaches are emerging each year. Many products in modern robotics have achieved remarkable commercial success, some well-known instances include quadruped robotics[9], autonomous vehicles[45], quadcopters[36], and humanoid robots[43].

(a) Quadruped

(b) Quadcopter

(c) Humanoid

(d) Autonomous vehicle

Figure 1.1: Four successful examples for controlling dynamic and nonlinear systems in the field of modern robotics

1.1.1 Trajectory planning and tracking

In the traditional control of complex systems such as robotics, which exhibit significant nonlinearity, it is crucial to employ reliable nonlinear control strategies to guarentee motion capabilities.

Typically, the procedure for executing precise motion within these systems subjects to a two-phase method[7], encompassing trajectory planning and trajectory tracking. This structured approach ensures precision and stability in robotic system control.

Trajectory planning[18][19] involves calculating a smooth and feasible path for the robot to follow, aiming for a specific target position or operational point. A widely-used method in this stage, trajectory optimization[6], seeks to minimize a cost function that accounts for metrics like travel time, energy consumption, and motion smoothness. Techniques such as gradient descent or genetic algorithms are often utilized for optimization. These techniques are subject to constraints, with aspects like maximum velocities and accelerations being crucial in the process.

Upon successful trajectory planning, the next step is trajectory tracking, which requires the use of control algorithms to guide the robot along the predetermined path. A feedback control approach is typically employed, continually monitoring the robot's position and adjusting control inputs as necessary. However, in real-world systems, external disturbances, uncertainties, and system limitations may lead to significant deviations from the planned trajectory. Therefore, ensuring accurate state estimation and implementing robust control strategies are imperative in feedback control to address these challenges.

Below, two examples of trajectory-based control are discussed. In conventional industrial robot control[20], it is typical for the control process to be divided into distinct phases of offline planning and execution. This segmentation is primarily due to the non-critical need for real-time responsiveness in such applications. Consider the planning phase, for example, where an optimal route is determined by an algorithm such as A*, based on a predefined task. Tasks often include navigating around obstacles and minimizing travel distance, leading to the generation of a set of discrete waypoints[13]. These waypoints are then subjected to cubic interpolation techniques[8], resulting in the crafting of a smooth and continuous trajectory that can be realistically followed by the robot. This approach ensures not only seamless motion but also compliance with the robot's kinematic constraints. Subsequently, the process transitions to the execution phase, during which a robust and precise control strategy, such as PID, is implemented. This strategy is crucial, as it ensures the robot's adherence to the pre-established trajectory, maintaining accuracy and reliability throughout the operation[37].

Yet, in dynamic domains like automotive and flight control, the demand for real-time responsiveness takes center stage. In such cases, Model Predictive Control (MPC)[31] serves as a key example of integrating online trajectory planning and execution, meeting the critical need for real-time responsiveness. In the planning phase, MPC forecasts a series of future control actions. This is achieved by minimizing discrepancies between the current observed state and the intended trajectory. Once the planning is complete, MPC implements the first

set of these predicted control inputs during the execution phase. Immediately following this implementation, the system's new state is observed and fed back into the MPC algorithm. This leads to a re-evaluation and update of the future control actions in what becomes a continuous, iterative process. MPC's strength lies in its ability to simultaneously create, optimize, and follow trajectories, dynamically adjusting in real-time to address any disturbances and uncertainties. Such capability is essential for ensuring precise control and adaptability in complex systems that operate in real-time environments.

In conclusion, the traditional trajectory planning and tracking approach, while effective for numerous systems, has its own set of limitations.

- **Limited Adaptability:** Trajectory planning typically relies on predefined paths or trajectories, limiting adaptability to unforeseeable changes or dynamic environments. If the environment changes significantly during execution, the planned trajectory may no longer be optimal or even feasible.

- **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.

- **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

Transitioning from a focus on predefined trajectories, reinforcement learning(RL)-based control presents an alternative framework.

Reinforcement learning (RL)[50] is a subset of machine learning that focuses on agents learning optimal behavior through trial-and-error interactions with their environments. Essentially, the trained agent makes sequential decisions, observes the outcomes of its actions, and receives feedback in the form of rewards or penalties. This feedback serves as a guiding mechanism, enabling the agent to evaluate the outcomes of its actions. It then adjusts its policy accordingly to maximize cumulative rewards over time.

The mathematical framework that describes RL is the Markov Decision Process (MDP)[41], which provides a structured model for the decision-making problem. In an MDP, the agent's current state, the available actions, the potential next states, and the rewards associated with state-action transitions are all clearly defined. MDPs require the fulfillment of the Markov property, which states that the future state of the system depends only on the current state and the action taken.

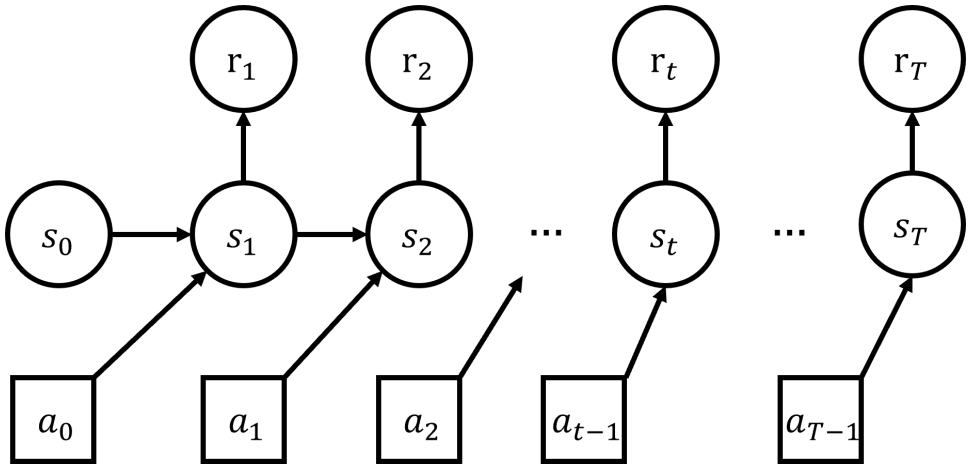


Figure 1.2: Markov decision process

The ultimate objective in reinforcement learning (RL) is to discover an optimal policy. This policy is a strategic mapping from states to actions, designed to select the most beneficial action in each state to achieve maximum long-term rewards. The learning process is dependent on the tuple (s, a, r, s'), which describes the current state s, the action taken a, the reward received r, and the subsequent state s'. Over time, the agent engages in exploring the state and action space to gather information about the effects of various actions, which is essential for acquiring new knowledge. This exploration is complemented by exploitation, where the agent utilizes its accumulated knowledge to make informed decisions.

Maintaining a balance between exploration and exploitation is a key challenge in RL. It is crucial for the agent to effectively navigate its environment and learn from the rewards it receives. Through this iterative process of exploration and exploitation, the agent continuously refines and updates its policy. Gradually, this leads to the convergence towards optimal behavior, where the policy consistently yields the highest possible rewards within the constraints of the environment.

In the field of robotics, the integration of reinforcement learning in control tasks[29] has become increasingly popular due to the numerous distinct advantages this approach offers:

- **Adaptability and Flexibility:** RL allows systems to adapt and improve in dynamic environments; its control policy evolves through accumulating new experiences and knowledge from interactions with the environment, making it highly adaptable to varying circumstances.

- **Reduced Dependency on Accurate Models:** In contrast to traditional control methods that depend on exact mathematical models, RL works without the necessity of a predetermined model, particularly when learning from real-world data. This feature is immensely beneficial in situations where the system dynamics are either complex or not well understood, as RL refines its performance through direct interactions with the environment.

- **Effective Handling of Nonlinearities and High-Dimensional Systems:** RL excels at managing nonlinear systems and complex control tasks using neural network-based function approximation. This enables it to navigate high-dimensional input and output spaces and deal with coupling problems in multiple input and output systems.

In conclusion, reinforcement learning is a straightforward control method with high adaptability, setting itself apart from traditional trajectory planning and tracking techniques. While conventional methods frequently struggle with challenges such as limited adaptability, nonlinearity in systems, and intensive computational demands, reinforcement learning addresses these issues through direct interactions with the environment. It excels in unpredictable conditions and in managing complex, high-dimensional systems. This is largely attributed to its reduced dependency on accurate dynamic models, ensuring robustness, effectiveness, and versatility across a wide range of applications.

1.2 Problem setup

In the realm of nonlinear systems, underactuated systems[35] present a particularly challenging class. These systems are characterized by having fewer control inputs than degrees of freedom, or their control inputs are constrained in some way. This characteristic makes them significantly more challenging to control when compared to fully actuated systems. What's intriguing is that a majority of robots and even natural organisms fall into the category of underactuated systems. Consequently, the study of underactuated mechanical systems' control holds universal relevance.

The concept of underactuated dynamics can be briefly introduced as follows. According to Newton's second law ($F = ma$), the dynamics that govern any mechanical system can be mathematically expressed as shown in Equation 1.1:

$$\ddot{q} = f(q, \dot{q}, u, t) \tag{1.1}$$

In this equation, $\ddot{q}$ represents acceleration, which is the second derivative of the variable q, typically representing the position of the system. The function $f(q, \dot{q}, u, t)$ describes how $\ddot{q}$ depends on various parameters, including the state variables q and $\dot{q}$, the control inputs u, and the time t.

The state of the system, denoted as x and represented as $[q, \dot{q}]^T$, consists of two vectors: q, which signifies positions, and $\dot{q}$, which signifies velocities.

When dealing with control-affine systems, we can express the second-order differential equation in the following manner:

$$\ddot{q} = f_1(q, \dot{q}, t) + f_2(q, \dot{q}, t)u \tag{1.2}$$

In this equation, $f_1(q, \dot{q}, t)$ corresponds to one part of the function that affects $\ddot{q}$, while $f_2(q, \dot{q}, t)$ represents another part that interacts with the control input u.

In the context of a controlled dynamical system, as described by Equation 1.2, we evaluate the condition of underactuation at specific states $[q, \dot{q}]^T$ at time t. Underactuation can be identified through two distinct scenarios:

- **Case 1:**

  The system is classified as underactuated at a particular state if the rank of the matrix $[f_2(q, \dot{q}, t)]$ is less than the dimension of q. This condition is expressed as:

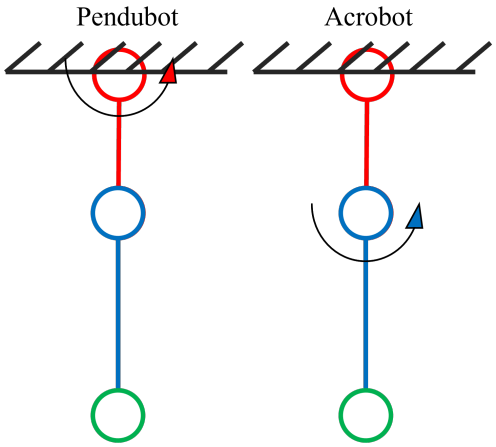
$$\text{rank}[f_2(q, \dot{q}, t)] < \dim[q] \tag{1.3}$$

- **Case 2:**

  Alternatively, underactuation may also arise even when f_2 is full rank, provided there are additional constraints on the control inputs. For instance, if constraints such as $|\mathbf{u}| \leq 1$ limit the control inputs, the system's controllability can still be restricted, resulting in underactuation.

These principles are explained in the work by Russ Tedrake[51].

A well-discussed example of underactuated control is found in the double pendulum—a simple setup consisting of two links connected by two rotational joints. These joints include the shoulder joint, which is directly connected to the world frame, and the elbow joint, situated between the two links. The end effector is located 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 referred to as a pendubot (see Figure 1.3a). Conversely, if the elbow joint is actuated, it is known as an acrobot (see Figure 1.3b).



(a) Pendubot setup *(b) Acrobot setup*

Figure 1.3: Two variations of underactuated double pendulum

Despite its simple configuration, the system exhibits highly nonlinear and chaotic behavior[46]. 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[58], and it continues to be a crucial testbed for validating the effectiveness of newly designed control algorithms[57][59][3].

Our project's objective 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. The real system setup is shown in Figure 1.4

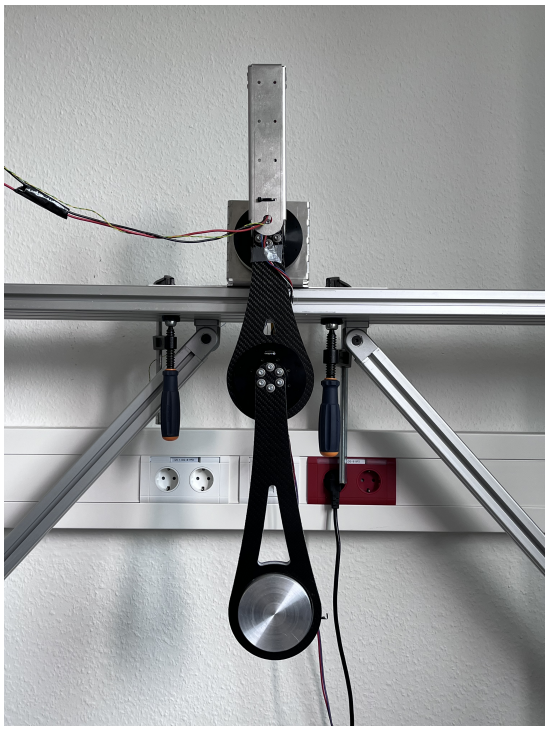


Figure 1.4: Double pendulum in real system

1.3 Contribution

In this thesis, the primary contribution is the development of an effective control strategy for a double pendulum to achieve two key objectives. The first objective involves swinging the double pendulum from its lowest point to its highest point. The second objective is to ensure stability at the highest point over an extended period.

For the swing-up task, a well-known model-free reinforcement learning algorithm, Soft Actor-Critic (SAC), was employed[23]. This algorithm enabled the training of a policy capable of reaching the Region of Attraction (RoA) of a continuous-time linear quadratic regulator (LQR)

controller[33]. Upon reaching the RoA, the system transitions seamlessly to the LQR controller, maintaining stability around the highest point.

The findings from the simulation and real hardware tests were presented at the IJCAI 2023 conference in Macau, in the RealAIGym competition[21]. Furthermore, this work has been made available as open source, accessible at:

`https://github.com/dfki-ric-underactuated-lab/double_pendulum`.

1.4 Content

This thesis comprises seven chapters, namely: Introduction, State of the Art, Methodology, Agent Training and Experiments in a Simulation Environment, Experiments on Real Hardware, Discussion, and Conclusion and Future Work. At the end of Chapter 1, the content of the following chapters is outlined below.

- **Chapter 2: State of the Art**:
  - In this chapter, an explanation is provided on the fundamental theories related to double pendulum dynamics, followed by an overview of recent advancements in the field of robotic control, with a particular emphasis on learning-based control methods.
- **Chapter 3: Methodology**:
  - The methodology is explored in this chapter, covering fundamental aspects of reinforcement learning, with a specific focus on the SAC algorithm. The reward function used for training, the training procedure, and the concept of the LQR controller and the composition of the combined controller are explained. Furthermore, evaluation metrics used to assess the performance and robustness of the newly designed control strategies in both simulated environments and real-world experiments are introduced.
- **Chapter 4: Agent Training and Experiments in Ideal Simulation Environments**:
  - The training process of agents based on the SAC algorithm in ideal simulation environments is presented in this chapter. Additionally, results obtained from these

simulations are demonstrated to showcase the performance and behavior of the designed control strategy.

- **Chapter 5: Experiments on Real Hardware**:
  - The hardware setup, including mechanics, electronics, and safety measures, is explained first in this chapter. Subsequently, system identification is introduced. The approach for bridging the sim-to-real gap is discussed next, encompassing the setting up of noisy simulations for evaluation and a noisy training process based on domain randomization. The process for selecting the suitable agent for real-world tests is also introduced. Finally, the outcomes of experiments conducted on the real hardware are provided, highlighting that the combined controller is functional only on the pendubot.
- **Chapter 6: Discussion**:
  - A discussion on the obtained results is undertaken in this chapter, including a comparison of the performance and robustness leaderboards in simulations and the performance leaderboards in the real world.
- **Chapter 7: Conclusion and Future Work**:
  - The final chapter offers a conclusion of the work and the results obtained. It also discusses future work that was not completed in this thesis.

2 State of the Art

In this chapter, the state of the art will be explored. The first section will discuss the basic dynamics of a double pendulum system, while the second section will review some of the most renowned works related to learning-based control in the field of robotics.

2.1 Theory

In this section, the fundamental principles governing the dynamics of a double pendulum will be explored.

Broadly speaking, a double pendulum system is a mechanical structure consisting of two pendulum arms or masses suspended independently, allowing them to swing freely. These pendulum arms are typically connected in series, with the motion of the second pendulum influenced by the motion of the first.

Double pendulum setups take various forms, including the widely researched double inverted pendulum on a cart (DIPC)[10] and the Furuta pendulum[12][17]. The double pendulum used in this thesis is relatively simple, comprising two links connected in series by revolute joints. Unlike the Furuta pendulum, both links of the double pendulum move in the same plane within three-dimensional space. These links are also connected to the world frame via revolute joints. In contrast to the DIPC setup, the actuation capability is solely limited by the torque that the joints can generate, unrestricted by the length of a prismatic rail. The dynamics of the double pendulum system will be discussed in the following section.

As shown in Figure 2.1, the dynamics of the double pendulum is modeled 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 g.

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

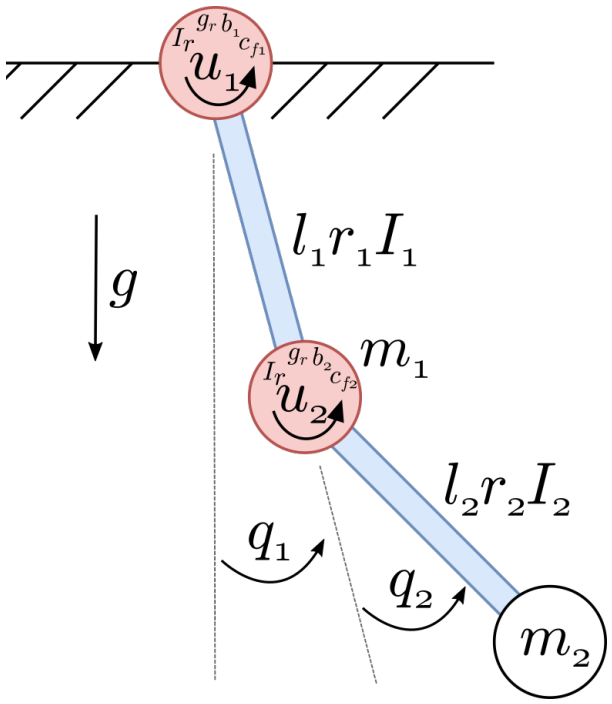


Figure 2.1: Double pendulum dynamics[14]

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 below 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:

$$L = T - U \tag{2.1}$$

Step 2. Express the Kinetic Energy (T):

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

$$T = \frac{1}{2} m_1 (\dot{x}_1^2 + \dot{y}_1^2) + \frac{1}{2} m_2 (\dot{x}_2^2 + \dot{y}_2^2) \tag{2.2}$$

where m_1 and m_2 are the masses of the links, (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 links. The potential energy for a link in a gravitational field is given by:

$$U = m_1 g y_1 + m_2 g y_2 \tag{2.3}$$

where g is the acceleration due to gravity.

Step 4. Formulate the Lagrange's Equation:

The equations of motion (EoM) expressed in joint coordination q are derived using Lagrange's equation:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0, \qquad i = 1, 2 \tag{2.4}$$

Step 5. Solve the Equations of Motion:

Considering the forward kinematics of the double pendulum system, the Cartesian coordinates of the joint between the first and second links are represented as $P_1 = (x_1, y_1)$, while the coordinates of the end effector are represented as $P_2 = (x_2, y_2)$. These Cartesian coordinates can be expressed in terms of the joint coordinates q as follows:

$$\begin{cases} x_1 = l_1 \sin(q_1) \\ y_1 = -l_1 \cos(q_1) \end{cases} \tag{2.5}$$

$$\begin{cases} x_2 = l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \\ y_2 = -l_1 \cos(q_1) - l_2 \cos(q_1 + q_2) \end{cases} \tag{2.6}$$

The set of second-order differential equations shown in Equation 2.4 is solved to determine the system's equations of motion without friction, as expressed in Equation 2.7.

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} = Du + G(q) \tag{2.7}$$

The system dynamics, including friction, are represented in Equation 2.8:

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

Expressing the equation of motion in terms of the state vector $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T$, we have:

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

Substituting Equation 2.5 and 2.6 into 2.2, 2.3, 2.4, and 2.8, we can derive the mass matrix (where $s_1 = \sin(q_1), c_1 = \cos(q_1), \ldots$).

$$\mathbf{M} = \begin{bmatrix} I_1 + I_2 + l_1^2 m_2 + 2 l_1 m_2 r_2 c_2 + g_r^2 I_r + I_r & I_2 + l_1 m_2 r_2 c_2 - g_r I_r \\ I_2 + l_1 m_2 r_2 c_2 - g_r I_r & I_2 + g_r^2 I_r \end{bmatrix} \tag{2.10}$$

The Coriolis matrix:

$$\mathbf{C} = \begin{bmatrix} -2\dot{q}_2 l_1 m_2 r_2 \sin(q_2) & -\dot{q}_2 l_1 m_2 r_2 \sin(q_2) \\ \dot{q}_1 l_1 m_2 r_2 \sin(q_2) & 0 \end{bmatrix}, \tag{2.11}$$

The gravity vector:

$$\mathbf{G} = \begin{bmatrix} -g m_1 r_1 \sin(q_1) - g m_2 \left(l_1 \sin(q_1) + r_2 \sin(q_{1+2})\right) \\ -g m_2 r_2 \sin(q_{1+2}) \end{bmatrix}, \tag{2.12}$$

The friction vector:

$$\mathbf{F} = \begin{bmatrix} b_1\dot{q}_1 + c_{f1}\arctan(100\,\dot{q}_1) \\ b_2\dot{q}_2 + c_{f2}\arctan(100\,\dot{q}_2) \end{bmatrix} \tag{2.13}$$

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

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

$$\mathbf{D}_{full} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{D}_{pendu} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{D}_{acro} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.14}$$

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

2.2 Related work

This thesis focuses on learning-based robotic motion control, a field that has garnered increasing attention in recent years, with significant contributions from various research institutes.

Today, much of the research in this field is conducted within simulation environments due to their cost-effectiveness and the ability to facilitate rapid iteration. One noteworthy project from 2018 is the DeepMimic project[40], undertaken by researchers at the University of California, Berkeley. This work resides at the intersection of deep reinforcement learning, imitation learning, and robotics.

The DeepMimic project utilizes physics-based simulations to successfully replicate the diverse range of behaviors exhibited in the real world by 3D characters. These characters include real-world examples such as humanoid and Atlas robotics, as well as fictional characters like T-Rexes and dragons. Instead of relying on manually designed controllers, the project employs deep reinforcement learning methods to generalize to new skills and situations, often without human intervention.

In the training process, the agent is provided with reference data recorded by motion capture actors or keyframed animations. Through imitation learning, the agent is guided to achieve specific predefined goals. The central contribution of this project lies in its framework, which combines goal-directed reinforcement learning with reference data generated by humans. This combination enables the imitation of a wide variety of motion skills.

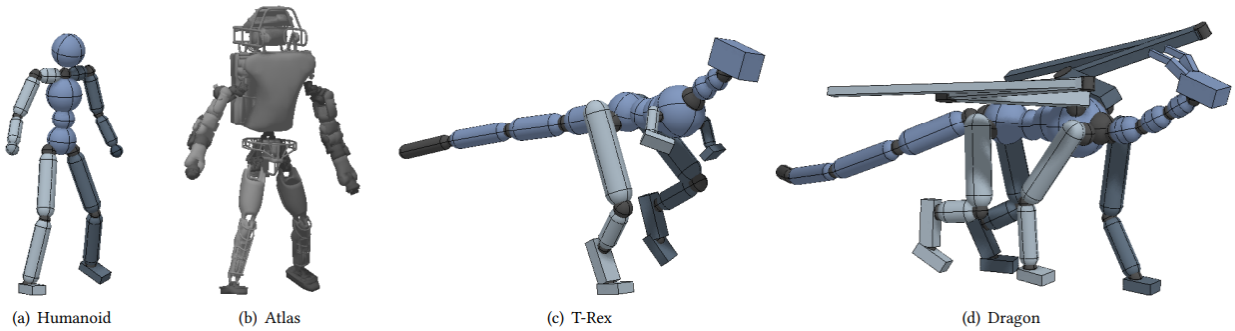


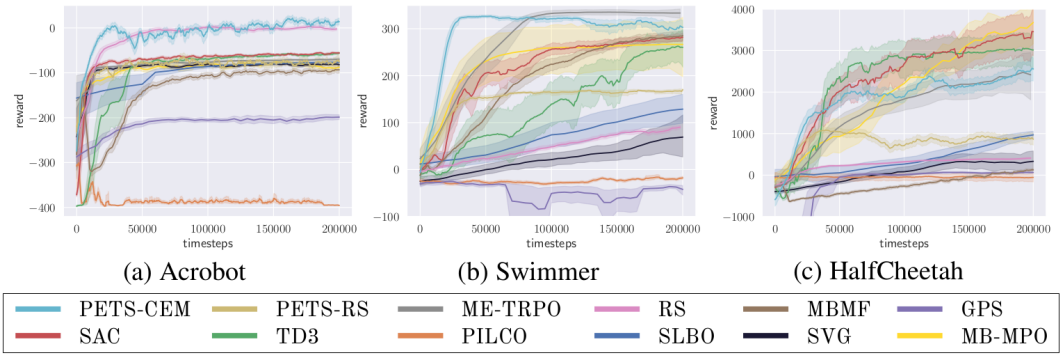
Figure 2.2: 3D characters in Deepmimic project[40]

In the context of reinforcement learning, a distinction exists between model-free and model-based approaches. Model-free reinforcement learning(MFRL) does not require information about the transition model, whereas model-based reinforcement learning(MBRL) leverages the transition model to make decisions based on a prior known model or one learned from interactions.

The model-free approach is notably characterized by its sample inefficiency. Successful training often takes many hours, if not days or weeks.

An intriguing project that relies solely on model-free deep reinforcement learning is the Learning-to-Walk-in-20 Minutes project[47] conducted by researchers from UC Berkeley. They employ an algorithmic framework closely related to DroQ [24], an extension of the SAC algorithm [23] incorporating dropout [48] and layer normalization [5]. Remarkably, their training is conducted directly on the real system. They demonstrate that current deep RL methods can effectively teach quadrupedal locomotion in under 20 minutes, a stark contrast to previous research conducted by Kumar et al. [32], which employed the same hardware but required 1.2×10^9 samples to train a robust controller for locomotion. This corresponds to roughly 4.5 months' worth of cumulative experience.

Model-based reinforcement learning is recognized for its integration of an environment model with trial-and-error learning. One notable advantage is the potential for higher sample efficiency compared to the model-free approach. This work, conducted by the University of Toronto[55], provides a comprehensive comparison by benchmarking model-based reinforcement learning.



(a) Acrobot (b) Swimmer (c) HalfCheetah

Figure 2.3: Benchmarking model-based reinforcement learning[55]

The team has collected 11 Model-Based Reinforcement Learning (MBRL) algorithms and 4 Model-Free Reinforcement Learning (MFRL) algorithms across 18 benchmarking environments specifically designed for MBRL in simulation. These benchmarking environments are based on the standard OpenAI Gym[11], ranging from simple 2D tasks like cart-pole to complex setups like humanoid. The benchmarking process is further extended by introducing noise into the environment, including disturbances in observations and actions.

The team has discovered that while 1 million time-step training is common for MFRL algorithms, many MBRL algorithms converge much earlier, often within 200k time-steps. Within the field of MBRL, when it comes to the evaluation of sample efficiency, asymptotic performance, and robustness, there is no clear and consistent best MBRL algorithm. This leaves ample opportunities for future research to leverage the strengths of different approaches.

Another notable challenge in reinforcement learning-based robotic control is the sim-to-real gap problem. Since agents are typically trained in carefully designed simulated worlds, these simulations can sometimes be idealized or oversimplified to some extent. Consequently, the optimal policy derived from the simulation often fails to account for the uncertainties of the real world, leading to failures when executing intended tasks. There are several approaches to bridging the sim-to-real gap, and the following work presents an interesting approach.

A research team from ETH Zurich has made significant progress in addressing the sim-to-real interface challenge [25]. Their methodology involves training a control model for a quadruped 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 the 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.

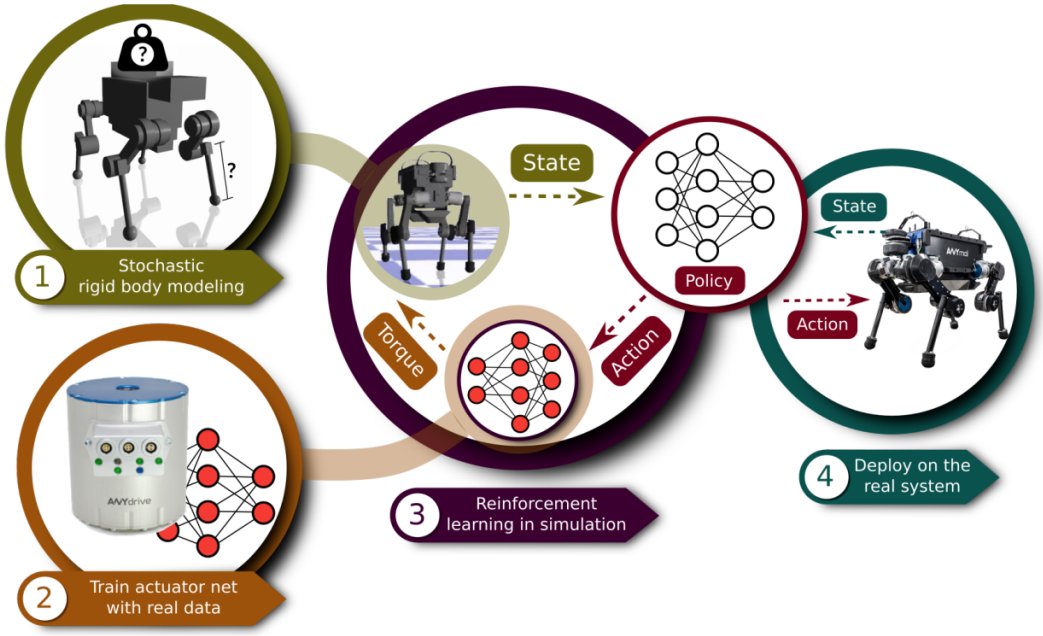


Figure 2.4: Sim-to-real framework for RL-based quadruped controller[25]

In conclusion, in the field of reinforcement learning-based control in robotics, researchers currently face three major challenges:

1. **Sample Inefficiency in Model-Free Reinforcement Learning:**

   Model-free reinforcement learning is known for its sample inefficiency, where the time required for random trial and error can be excessively long.

2. **Immaturity of Model-based RL**

Model-based reinforcement learning is a growing force but still in its infancy. Finding the right balance between model information and reinforcement learning remains an ongoing challenge.

3. **Sim-to-Real Gap Limitations:**

   Deploying controllers learned through reinforcement learning on real systems is limited due to the significant sim-to-real gap. This limitation confines a substantial portion of research to simulations.

The research community has much work to do before reinforcement learning becomes a stable and widely accepted approach in the industry.

3 Methodology

The primary goal of this thesis is to achieve the swing-up movement in the pendubot or acrobot setup and to maintain stability around the highest points. Challenges have been revealed in initial training trials with reinforcement learning, including potential entrapment in local minima and difficulty in maintaining stability at the highest point for extended periods.

To address these challenges, two main strategies are adopted. For the stabilization issue, a combined controller is introduced. During the swing-up process, control is assumed by an RL-trained agent, utilizing the Soft Actor-Critic—a classic model-free reinforcement learning algorithm—based on its learned policies. However, as the system nears the maximum point, a seamless transition occurs. This shift allows for the takeover of control by a continuous-time LQR controller, ensuring the final stabilization required to maintain stability at the highest point. For the entrapment in local minima problem, a carefully designed reward function is utilized to guide the agent away from hazadous local minima and achieve successful swing up.

3.1 Soft actor critic

Within the landscape of reinforcement learning, the Soft Actor Critic (SAC)[23] stands out as an algorithm specifically designed for environments with continuous state and action spaces. Such environments, exemplified by our double pendulum system where actuators can be adjusted to any value within the torque limit range, and state measurement can take any real number, 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[30].

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[1]. 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:

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

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. Each interactioin between the agent and the environment can be recorded as a tuple (s_t, a_t, R, s_{t+1}). The optimal policy π^* can be defined as follows:

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

Where s_t and a_t represent the state and action at time t, and s_{t+1} represents the state at time $t+1$. R denotes the immediate reward received by the agent after taking action a_t in state s_t, while γ is the discount factor that determines the agent's emphasis on long-term cumulative rewards over immediate ones.

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 $V^\pi(s)$:

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

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

The interaction experiences are stored as tuples (s,a,r,s',d) in a replay buffer D. Here, d represents the signal for episode termination. When it is time to update the Q-value and policy, a batch of transitions $B = \{(s,a,r,s',d)\}$ is randomly sampled from D. The Q-functions are updated by one step of gradient descent, the gradients of the loss functions for the Q-networks are calculated by:

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2 \quad \text{for} \quad i = 1,2 \tag{3.7}$$

The policy is updated using one step of gradient ascent, the gradient is expressed as:

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \widetilde{a}_\theta(s)) - \alpha \log \pi_\theta(\widetilde{a}_\theta(s)|s)\right) \tag{3.8}$$

where $\widetilde{a}_\theta(s)$ is a sample of action derived from $\pi_\theta(\cdot|s)$ which is differentiable with respect to θ.

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)[33] 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 defined by state vector x and input vector u can be expressed as:

$$\dot{x}(t) = f(x(t), u(t)) \tag{3.9}$$

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

$$\dot{\bar{x}}(t) = A\bar{x}(t) + Bu(t) \tag{3.10}$$

Here, the deviation from the desired state is given by $\bar{x} = x - x_{\text{op}}$. Its first derivative, $\dot{\bar{x}} = \dot{x} - \dot{x}_{\text{op}} = \dot{x}$, for x_{op} being a constant. The linearized state matrices A and input matrix B around the operation point are derived as:

$$A = \left.\frac{\partial f}{\partial x}\right|_{x=x_{\text{op}}}, \quad B = \left.\frac{\partial f}{\partial u}\right|_{x=x_{\text{op}}} \tag{3.11}$$

To derive an optimal control strategy, a quadratic cost function J is needed:

$$J = \int_0^T \left(x^T Q x + u^T R u\right) dt \tag{3.12}$$

The matrix Q must be chosen to be positive-semidefinite and symmetric, meaning $Q = Q^T$ with all its eigenvalues non-negative, and R must be chosen to be symmetric and positive definite, meaning $R = R^T$ with all its eigenvalues positive.

The Hamilton-Jacobi-Bellman equation (HJB) characterizes the optimal value function $J^*(x)$, representing the minimum cost-to-go from state x to termination at T. The HJB equation is as follows:

$$\frac{dJ^*(x)}{dt} = \min_u \left(x^T Q x + u^T R u + \frac{dJ^*(x)}{dx}(Ax + Bu) \right) \tag{3.13}$$

By calculating $\frac{dJ^*(x)}{dt}$ from Equation 3.12, the HJB equation can be reduced to the continuous-time algebraic Riccati equation:

$$A^T S + SA - SBR^{-1}B^T S + Q = 0 \tag{3.14}$$

Finally, the algebraic Riccati equation must be solved for S numerically, typically using iterative methods like the Newton-Raphson algorithm. Then the LQR controller can be expressed as shown in Equation 3.15.

$$u(t) = -K\bar{x}(t) \tag{3.15}$$

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

For a 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, a combined control method is utilized for both the swing-up and stabilization tasks. This combined control framework is a natural choice and has previous work that supports this method. In this work by S. Gillen et al.[22], a similar structure combining a local controller

and a learned controller with a gate function was employed. The work was conducted on chaotic control of an acrobot setup in simulation, which resembles our project.

In our implementation, during the swing-up phase, the SAC controller is employed. Once the state of the double pendulum approaches the vicinity of the desired goal state, the transition from the SAC controller to the LQR controller is made for final-stage stabilization. An essential aspect of any combined control strategy is the determination of the conditions under which the system is ready for a transition between control methods. In our SAC+LQR control strategy, the gate function for making this determination is provided by the region of attraction method[38].

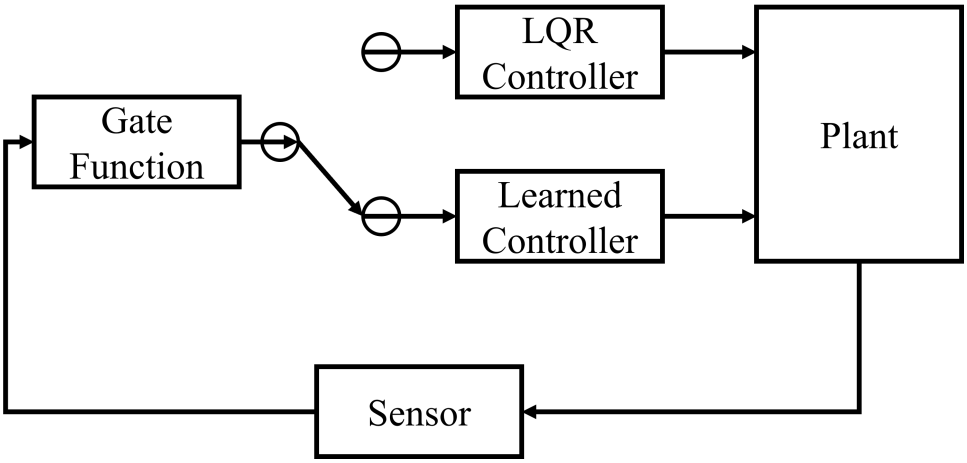


Figure 3.1: Combined controller

The region of attraction (RoA) for a nonlinear system, denoted as R_a, describes the set of initial states surrounding a fixed point x_0. If a state lies within this region, the system will converge towards x_0 as $t \to \infty$. In the context of an LQR controller, the RoA signifies the area within the state space where the controlled system exhibits asymptotic stability. For complex systems, directly computing R_a can be challenging; instead, it is often estimated. The simplest way to estimate such a set requires the assistance of a Lyapunov function $V(x)$ bounded by a scalar ρ[28].

$$B = \{x | V(x) < \rho\} \tag{3.16}$$

Here, B represents the estimated subset of the real region of attraction R_a.

The goal is to find the largest ρ for which the Lyapunov conditions are satisfied. These conditions are as follows:

$$\begin{cases} V(x) > 0 \\ \dot{V}(x) < 0 \quad \text{for} \quad x \in B \end{cases} \tag{3.17}$$

The Lyapunov function for a controlled linear system, $\dot{x}(t) = (A - BK)x(t)$, is chosen in a quadratic form:

$$V(x) = x^T S_{LQR} x \tag{3.18}$$

Here S_{LQR} is a positive definite matrix. This function serves as an 'energy-like' metric. Next, we calculate the time derivative of the Lyapunov function.For the infinite horizon LQR, $\frac{\partial S_{LQR}}{\partial t} = 0$ and $\frac{\partial x_0}{\partial t} = 0$, hence, $\dot{V}$ is:

$$\dot{V}(x) = 2x^T S_{LQR} \dot{x} \tag{3.19}$$

Combining equations and conditions 3.17, 3.18, 3.19, we have the following expression:

$$\begin{cases} B = \{x \mid 0 < V(x) < \rho\} \\ \dot{V}(x) = 2x^T S_{LQR} \dot{x} < 0 \end{cases} \tag{3.20}$$

The RoA is computed similar to [38] but with a sums of squares method[52]. The resulting shape of the RoA in a 4D state space resembles an ellipsoid.

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.

3.4 Reward shaping

The reward function is intended to guide the agent in performing desired behavior. In our design, the reward function aims to guide the double pendulum system towards achieving stability around the highest point.

One of the primary reasons why controlling an underactuated double pendulum system using RL is so challenging is the state space that can lead to successful stabilization is very small. Initial training attempts using OpenAI Gym Acrobot-v1[54] rewards have revealed this challenge. For instance, the agent may accidentally get close to the goal state but not receive enough reward. This can result in being stuck in an unsuitable position for an extended period or a high-speed rotation of the second link with the first link almost upright.

Another significant challenge in our RL-based training is that, due to our plan for real hardware deployment, the dynamic model of the double pendulum is more detailed than that of most simulation environments like OpenAI Gym Acrobot-v1[54]. Because we are using quasi-direct drive motors to simplify joint dynamics, one of the significant drawbacks of this motor type is its low gear ratio, resulting in a relatively low torque limit (5 Nm).

Therefore, there is a need for an innovative reward function design suitable for our problem setup. To tackle the swing-up issue, a customized three-stage reward function is designed 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:

$$\begin{aligned} r(x,u) = &- (x-x_g)^T Q_{train}(x-x_g) - u^T R_{train} u \\ &+ \begin{cases} r_{line} & \text{if } h(p_1,p_2) \geq h_{line} \ , \\ 0 & \text{else} \end{cases} \\ &+ \begin{cases} r_{LQR} & \text{if } (x-x_g)^T S_{LQR}(x-x_g) \geq \rho \ , \\ 0 & \text{else} \end{cases} \\ &- \begin{cases} r_{vel} & \text{if } |v_1| \geq v_{thresh} \ , \\ 0 & \text{else} \end{cases} \\ &- \begin{cases} r_{vel} & \text{if } |v_2| \geq v_{thresh} \ , \\ 0 & \text{else} \end{cases} \end{aligned} \tag{3.21}$$

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.

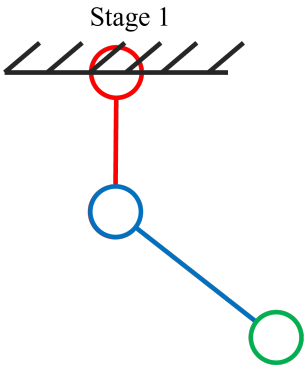


Figure 3.2: Swing up stage 1

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

$$h(p_1, p_2) = -l_1 \cos(p_1) - l_2 \cos(p_1 + p_2). \tag{3.22}$$

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 $v_{thresh} = 8\,\mathrm{rad/s}$ in absolute value. This penalty effectively compels the agent to approach the maximum point while adhering to the predefined speed interval(less than 20 rad/s). The speed penalty was only needed for the acrobot when the experiments are confined in simulation.

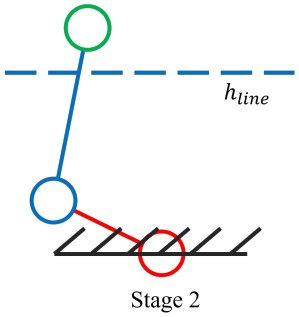


Figure 3.3: Swing up stage 2

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. The parameters, we used in the cost matrices of the LQR controller are listed in Table 4.2.

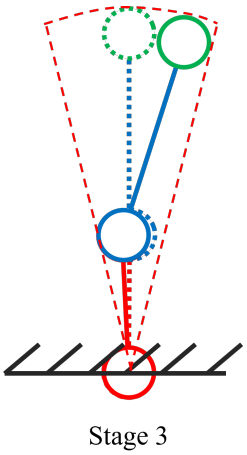


Figure 3.4: Swing up stage 3

3.5 Introduction to leaderboard metrics

To facilitate a comparison of controller performance for the double pendulum testbench, three separate leaderboards have been created: the simulation leaderboard, the robustness leaderboard, and the real system leaderboard[56]. Each of these leaderboards is additionally divided into two categories, which are determined by the pendubot and acrobot configurations.

3.5.1 Performance Leaderboard in Simulation and Real system

In the evaluation of controller performance, a set of metrics is utilized that extends beyond mere task success, delving into the finer aspects of controller operation. The same performance evaluation metrics are consistently applied to experiments conducted in both simulation environments and on real hardware. These metrics encompass various aspects, ranging from fundamental swing-up maneuver success to detailed examinations of energy consumption and torque utilization. Below, we provide a comprehensive 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.

The cumulative RealAI Score is determined based on the specified formula, using the following criteria:

$$\begin{aligned} S = c_{\text{success}} \Bigg(& w_{\text{time}} \frac{c_{\text{time}}}{n_{\text{time}}} + \\ & w_{\text{energy}} \frac{c_{\text{energy}}}{n_{\text{energy}}} + w_{\tau,\text{max}} \frac{c_{\tau,\text{max}}}{n_{\tau,\text{max}}} + w_{\tau,\text{integ}} \frac{c_{\tau,\text{integ}}}{n_{\tau,\text{integ}}} + \\ & w_{\tau,\text{cost}} \frac{c_{\tau,\text{cost}}}{n_{\tau,\text{cost}}} + w_{\tau,\text{smooth}} \frac{c_{\tau,\text{smooth}}}{n_{\tau,\text{smooth}}} + w_{\text{vel, cost}} \frac{c_{\text{vel, cost}}}{n_{\text{vel, cost}}} \Bigg) \end{aligned} \tag{3.23}$$

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 3.1: 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 $T = 10s$. The pendubot is initiated in a hanging down configuration, represented as $x_0 = [0,0,0,0]^T$, with the goal of reaching the unstable fixed point in the upright configuration, denoted as $x_g = [\pi,0,0,0]^T$. The double pendulum is considered 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.

In real hardware experiments, there exists a torque limit of 0.5 Nm on the passive joint, which compensates for the motor's friction. The actuators are capable of operating at a control frequency as high as 500Hz, and each experiment has a duration of 10 seconds. The pendubot commences from a hanging down position, aiming to reach 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.

3.5.2 Simulation Robustness Leaderboard

In addition to performance metrics, robustness metrics are also considered. As the ultimate goal is to transfer successful models from a simulation environment to real hardware, it's essential to assess the robustness of controllers developed within the simulation. This helps determine the types of perturbations that affect each controller.

- **Model Inaccuracies** c_{model}: Model parameters determined through system identification are inherently subject to inaccuracies. To assess these inaccuracies, variations are introduced one at a time into the independent model parameters within the simulator while maintaining the use of the original model parameters in the controller.

- **Velocity Measurement Noise** $c_{vel,noise}$: The outputs of the controllers depend on the measured system state, and in the case of the QDDs, online velocity measurements introduce noise. Therefore, it is important for transferability that a controller can handle at least this level of noise in the measured data. Testing is conducted with and without a low-pass noise filter.

- **Torque Noise** $c_{\tau,noise}$: Beyond measurement noise, the torque output by the controller may not precisely match the desired value.

- **Torque Response** $c_{\tau,response}$: The controller's requested torque typically varies during execution, and the motor may not be able to instantaneously respond to significant torque changes. Instead, it may overshoot or undershoot the desired torque value. This behavior is modeled by 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}: In real-system operations, time delays inevitably arise due to communication and 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:

$$S = w_{model}c_{model} + w_{vel,noise}c_{vel,noise} + w_{\tau,noise}c_{\tau,noise} + w_{\tau,response}c_{\tau,response} + w_{delay}c_{delay} \tag{3.24}$$

The weights are:

$$w_{model} = w_{vel,noise} = w_{\tau,noise} = w_{\tau,response} = w_{delay} = 0.2 \tag{3.25}$$

4 Agent Training and Experiments in Ideal Simulation Environments

Chapters 4 and 5 focus on the experimental phase of the project. This chapter details the training procedure of SAC agents for the swing-up task and then transitions to a simulation phase in ideal environments to validate results for both the swing-up and stabilization tasks. The chapter is structured into three distinct subsections: training setup, training process, and simulation results.

The first section describes the training framework, combining a Stable-Baseline3-based RL algorithm with a customized environment inherited from the OpenAI Gym environment. The second section centers on hyperparameter-tuning and highlights the challenges encountered during training. The third section displays the results acquired from both the pendubot and acrobot setups in ideal simulations.

4.1 Training setup in ideal environment

Stable Baseline 3(SB3)[42] 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[27]. 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[54] 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 Gymnasium 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, Gymnasium provides the ability to construct stacked training environments for parallel training.

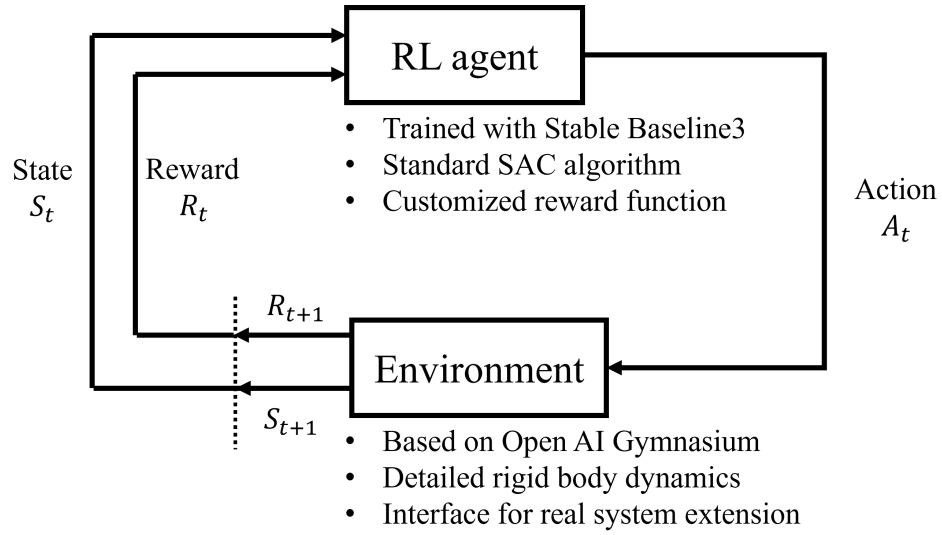


Figure 4.1: Interaction between reinforcement learning agent and customized environment

The construction of the customized environment begins with the dynamic function that captures the nonlinear dynamics of the underactuated double pendulum, as detailed in this repository[56]. The dynamic function, utilizing the current state observation and the action determined by the control policy, produces the next state via a Runge-Kutta integrator.

The integrated state is then processed by the reward function, which generates a scalar output. This output is, in turn, used by the SAC algorithm for policy evaluation and update. Following these computations, the simulation is tasked with calculating the current state, and the policy has the role of identifying the most probable action. These values are subsequently fed back into the dynamics function, thus initiating a new cycle.

It is widely recognized in the field of reinforcement learning that algorithms tend to converge more effectively when utilizing normalized state and action spaces [50]. Therefore, a scaling mechanism is designed to map the state and action spaces from their normalized versions to real-world measurements. The activation or deactivation of this scaling is optional.

For instance, in the case of a normalized state within the interval $[-1,1]$, we may choose to map it to real-world measurements such as p_1 in $[0,2\pi]$, p_2 in $[-\pi,\pi]$, and velocities v_1 and v_2 in $[-20,20]$ rad/s, while maintaining torque within the range $[-\tau_{\text{limit}},\tau_{\text{limit}}]$. When this scaling

mechanism is activated, it ensures that states and actions within the SAC algorithm always remain within the $[-1, 1]$ boundary, thereby often leading to faster convergence.

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

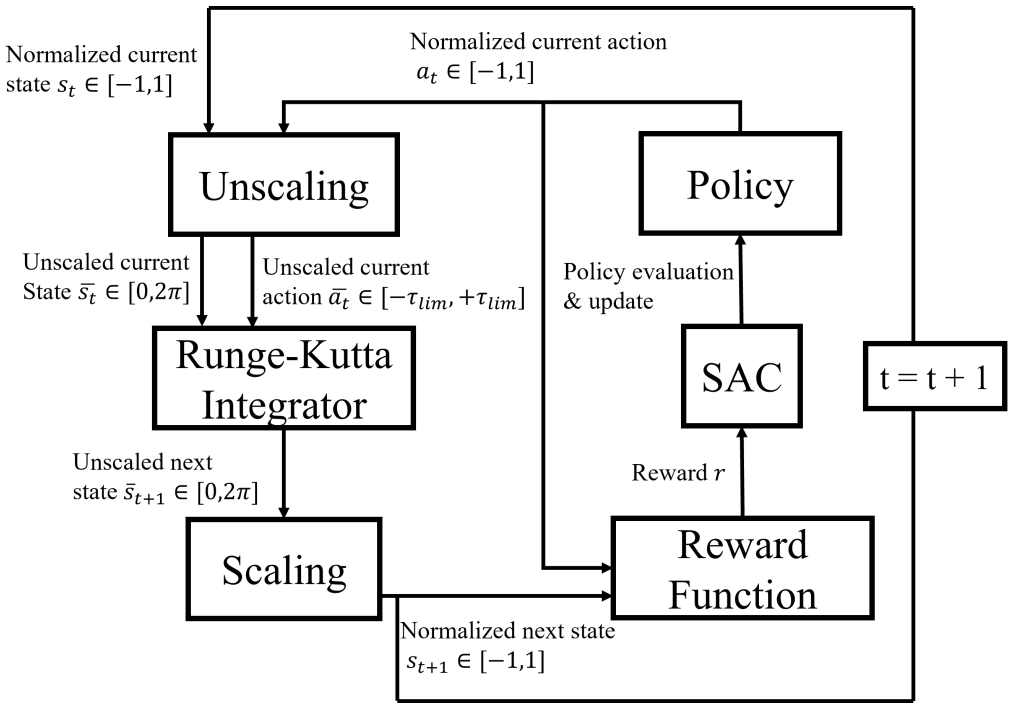


Figure 4.2: Detailed scaling pipeline in training process

Previous work in the double pendulum repository has featured several combinations of link lengths, designated as designs A, B, C, and D. Each design has undergone various system identification attempts, resulting in different outcomes labeled as models 1.0, 2.0, and so on. The combinations of these designs are displayed below:

	design A	design B	design C	design D
l_1(m)	0.3	0.3	0.2	0.3
l_2(m)	0.2	0.4	0.3	0.3

Table 4.1: Combinations of l_1 and l_2 for different designs

In addition, the trial phase during training introduces a noisy initialization by adding Gaussian noise to the presumed initial state $[0,0,0,0]^T$, thereby increasing exploration. The mean of

this Gaussian noise is set to $[0,0,0,0]^T$, and the covariance matrix is diagonal, specified as $diag(0.01, 0.01, 0.005, 0.005)$. For the evaluation phase, a zero initialization is employed to ensure the exploitation of the learned policy.

4.2 Training process in ideal environment

In the training of the SAC controller applied to both the acrobot and pendubot systems, the optimization of key hyperparameters was prioritized to ensure effective learning. The learning rate was carefully set to 0.01, a value chosen to balance between rapid adaptation and stable convergence. Control frequency was another critical hyperparameter, set at 100Hz, which provided the controller with frequent updates, contributing to a heightened responsiveness in the control process.

The length of each episode was standardized to 1000 steps, equivalent to 10-second-long episodes, for both systems. This duration was chosen to afford the agent ample opportunity for exploration and learning within diverse scenarios. To fully capitalize on the training, a significant number of learning timesteps were executed—2e7 for the pendubot and 5e7 for the acrobot. This extensive experience allowed the agent to progressively refine its performance.

Additionally, considerable effort was invested in fine-tuning the hyperparameters specific to the reward function and the LQR controller, both of which play pivotal roles in the agent's learning process. The particulars of these hyperparameter settings are meticulously documented in Table 4.2.

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 4.2: Hyper parameters used for the SAC training and the LQR controller

The architecture of the resulting SAC agent is an actor network, which comprises 67,586 parameters. This network is systematically structured into three segments: firstly, a feature extractor that preprocesses input states; secondly, a latent policy network consisting of two fully connected layers, each with 256 neurons, responsible for decision-making; and lastly, two distinct output layers tasked with generating the mean and the standard deviation of the policy's probability distribution, which are essential components for the SAC's stochastic policy.

Experiments on the pendubot in the ideal simulation phase use Design A, namely, $l_1 = 0.3$ m and $l_2 = 0.2$ m. One of the successful learning curves, spanning a total of 2e7 timesteps for the pendubot, is illustrated in Figure 4.3. As depicted in the figure, the learning curve experiences a gradual ascent from 0 to 1.25e7 timesteps. Between 1.25e7 and 1.6e7 timesteps, the curve surges rapidly, peaking at around 9e6 in reward. After 1.6e7 timesteps, the curve begins to stabilize with occasional fluctuations, and no substantial increase in reward is observed.

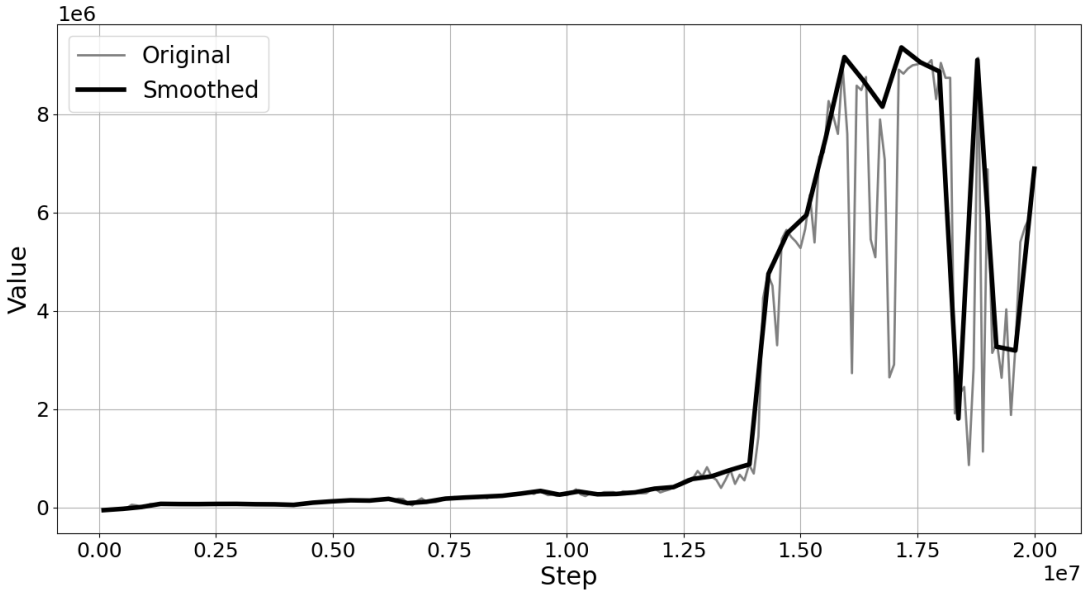


Figure 4.3: Pendubot learning curve

Experiments on the acrobot during the ideal simulation phase were based on Design C, with l_1 set to 0.2m and l_2 to 0.3m. As depicted in Figure 4.4, the acrobot underwent a total of 3e7 timesteps of learning. Initially, the learning curve exhibited relatively steady growth up until 1.9e7 timesteps. Afterwards, the learning curve began to increase drastically after that. By

3e7 timesteps, the learning curve had not yet reached stability. This learning of 3e7 timesteps delivered a working agent, but the behaviour was not stable. Consequently, we extended the learning period to 5e7 timesteps, using a warm start from the agent obtained at 3e7 timesteps, and observed that the learning curve began to stabilize around 4e7 timesteps. The resulting agent has a good result in ideal simulation.

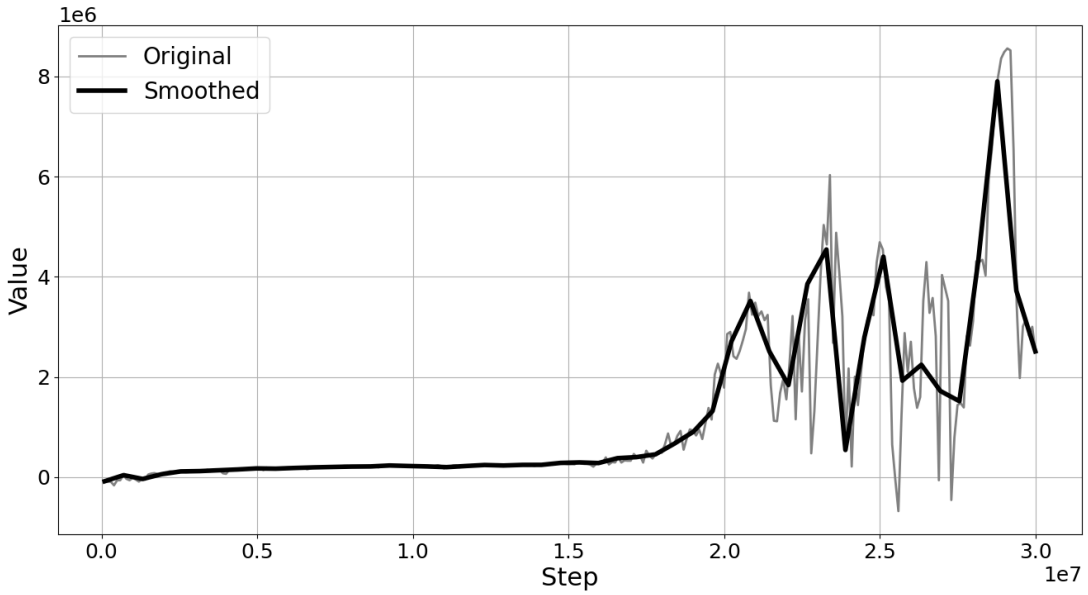


Figure 4.4: Acrobot learning curve

4.3 Results in ideal simulation

In this section, the ideal simulation results for the acrobot and pendubot are presented separately. Agents trained using the SAC algorithm are generally utilized during the swing-up stage, and the systems transition to the LQR controller when approaching the upright position. The primary criteria for success in both the swing-up and stabilization tasks involve swinging the double pendulum up to the upright position and maintaining stability for a prolonged period. Therefore, a simulation duration of 10 seconds is established. If the double pendulum fails to swing up or maintain stability within this timeframe, the trial is deemed unsuccessful.

4.3.1 Pendubot simulation in ideal environment

In the testing environment employed, also referred to as the ideal environment, disturbances such as friction and damping are excluded, with the focus placed solely on the effects of gravity, joint torque, and mechanical constraints. Zero initialization, as opposed to noisy initialization, is utilized, meaning the system commences from an initial state of $[0,0,0,0]^{\mathrm{T}}$, indicating a start at zero position with zero velocity, and the swing-up is initiated using a control policy exclusively derived from SAC.

As depicted in Figure 4.5, the swing-up time is under 1 second, after which the state of the pendubot enters the Region of Attraction (ROA) for the LQR controller. The transition between the two controllers is seamless and effective, with the LQR controller subsequently maintaining the system's stability at the desired state. This demonstrates the effectiveness of the ROA method for LQR control 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.

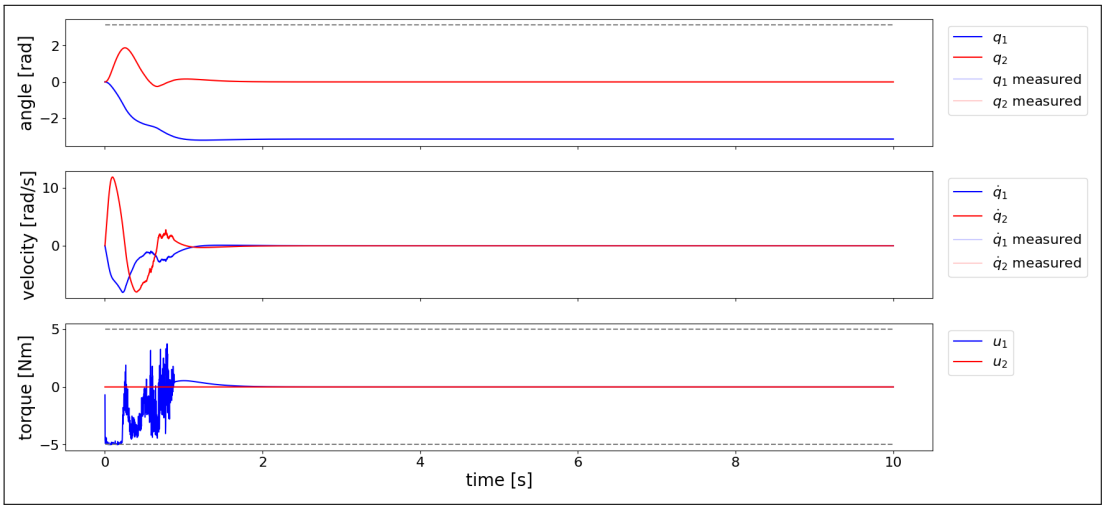


Figure 4.5: Pendubot result in ideal simulation

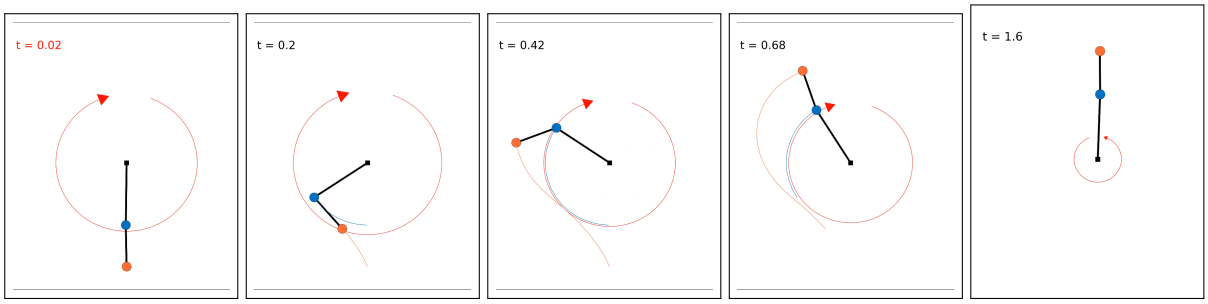


Figure 4.6: Snapshots of a pendubot ideal simulation test

4.3.2 Acrobot simulation in ideal environment

Just like the pendubot simulation, experiments on the acrobot setup are conducted in an ideal environment. The acrobot begins its swing from the downward position with zero velocity, aiming to stabilize around its highest point.

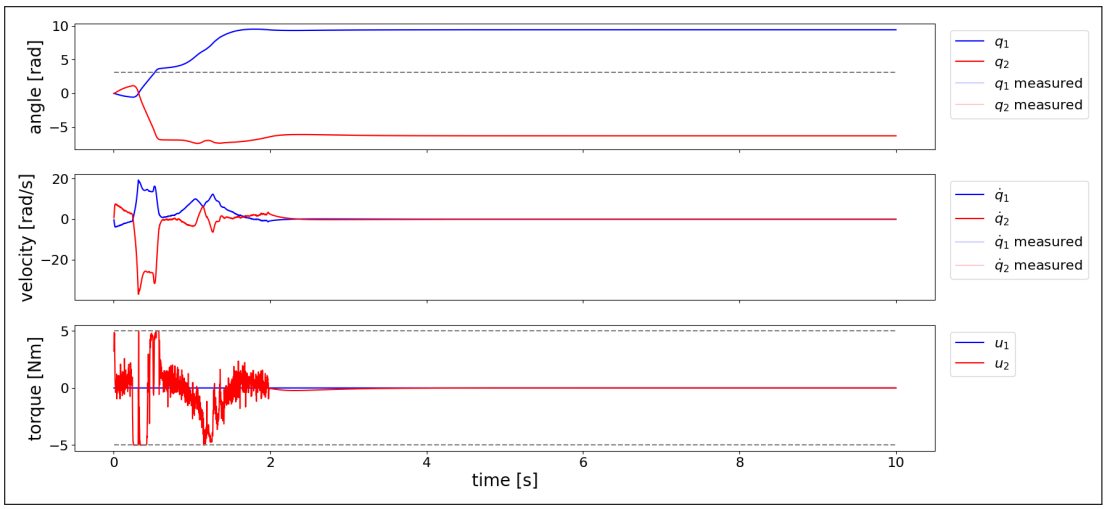


Figure 4.7: Acrobot result in ideal simulation

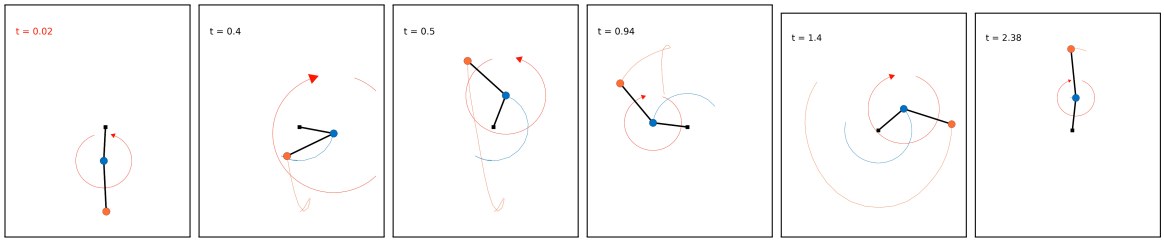


Figure 4.8: Snapshots of an acrobot ideal simulation test

The accompanying image (Figure 4.7) depicts a successful swing-up and stabilization within a 10-second window. The swing-up process takes approximately 2 seconds before the LQR controller effectively takes over, maintaining stability around the target state for a prolonged time period.

In comparison to the pendubot's simulation curves, the swing-up time for the acrobot is longer. 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.

4.3.3 Self stablizing behaviour on both pendubot and acrobot

An unexpected outcome emerged during the testing of the swing-up and stabilization of the underactuated double pendulum system using only the RL-learned policy. It was observed that some agents not only entered the Region of Attraction (ROA) of the LQR controller as intended but also remained upright for an extended period without LQR assistance. This self-stabilizing behavior was noticeable in both the pendubot and acrobot configurations.

In the pendubot setup, self-stabilization was more frequently observed. With a sufficient number of training timesteps (typically around 2e7), agents tasked with swing-up maneuvers in the pendubot configuration were highly likely to achieve self-stabilization at the upright position. As depicted in Figure 4.9, the stabilization by the RL-learned policy is less smooth compared to that achieved by the LQR-based stabilizer. The torque fluctuations have an amplitude of approximately 1.5 Nm, and there are minor fluctuations in position and velocity.

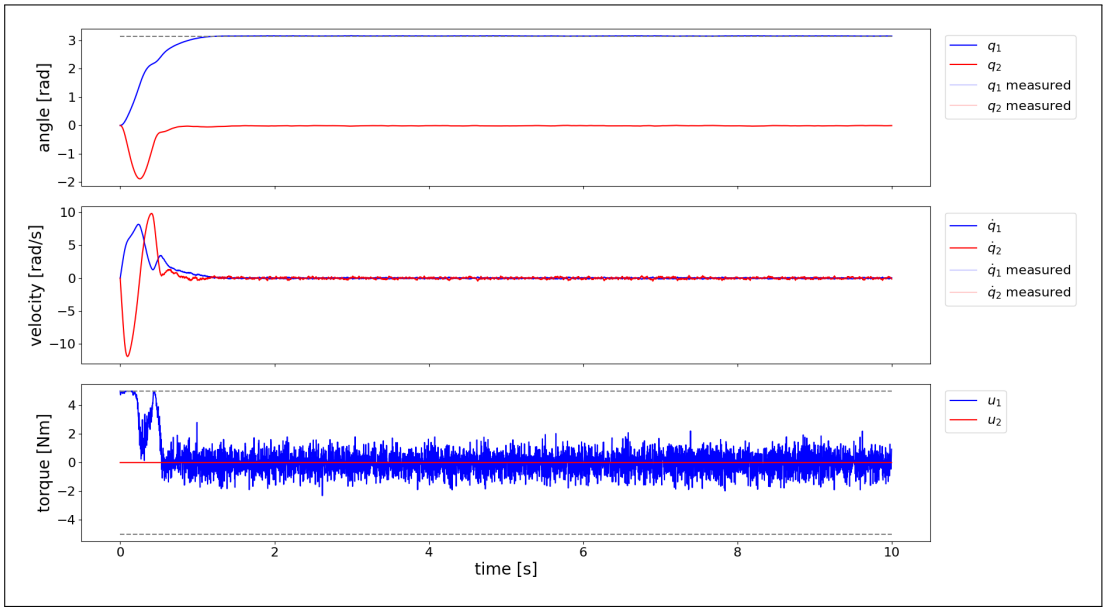


Figure 4.9: Self stabilization of pendubot in ideal simulation

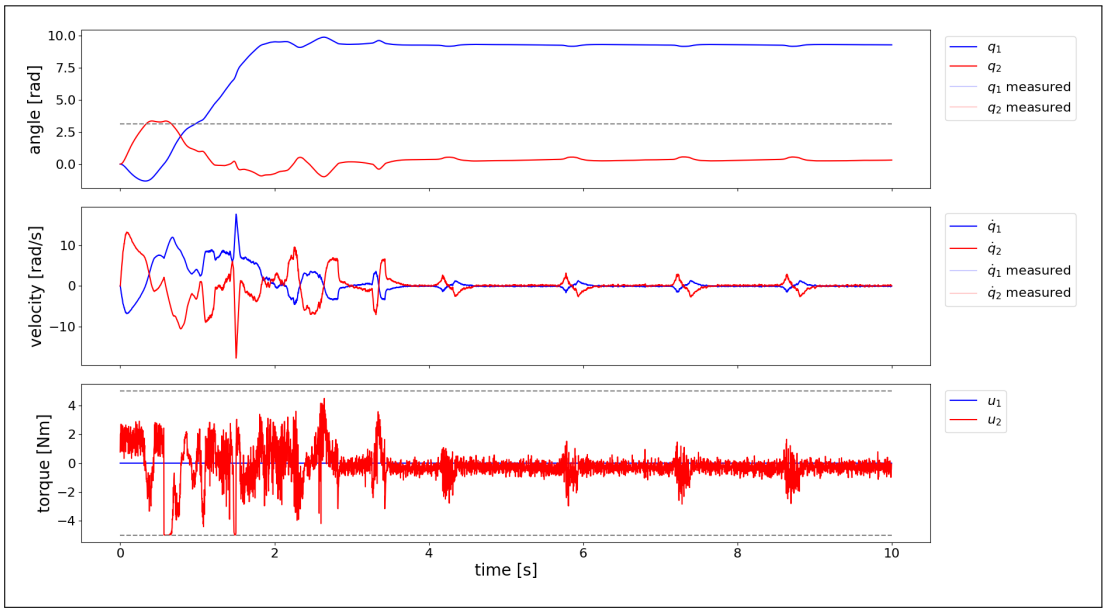


Figure 4.10: Self stabilization of acrobot in ideal simulation

Self-stabilization behavior has been observed in the acrobot setup as well. However, no consistent pattern has yet been identified as to the specific conditions under which the training yields a self-stabilizing agent. Fortunately, the same agent discussed in Section 4.3.2 exhibited a degree of self-stabilization, which is depicted in Figure 4.10. In the absence of a LQR controller, the RL-based controller was compelled to devise its own stabilization strategy. As illustrated in the graph, a relatively stable period commences at approximately 4 seconds, during which the system state exhibits prolonged fluctuations around the upright position. This occurs as the controller continuously corrects any deviation from the desired state. The stabilization provided by the RL-based controller is also less smooth than that of the LQR-based controller.

5 Experiments on Real Hardware

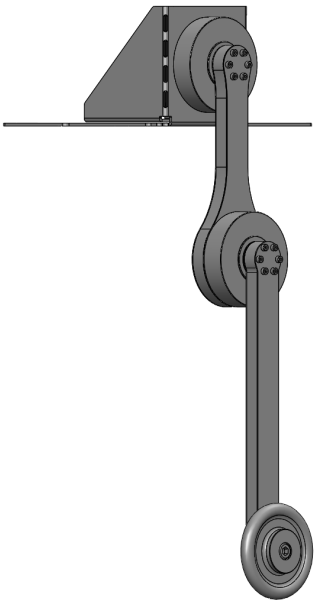
In this chapter, the experiments conducted on the hardware system are discussed. The content is organized into four sections. An overview of the hardware setup for the double pendulum system is provided in the first section. The system identification of the hardware is delved into in the second section. The approach to addressing the sim-to-real gap challenge is detailed in the third section. The outcomes of the hardware experiments are presented in the final section.

5.1 Hardware setup

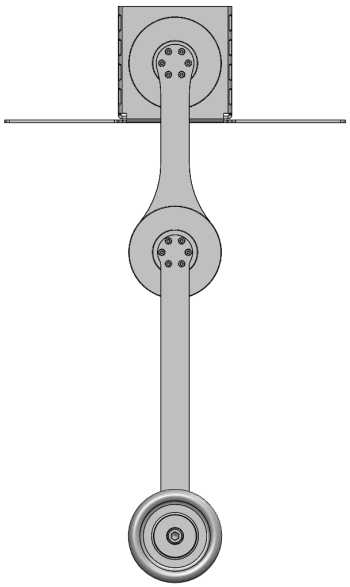
The hardware design and manufacturing were carried out by previous researchers and coworkers at the Underactuated Lab, part of the Robotics Innovation Center branch at the German Research Center for Artificial Intelligence GmbH. The hardware setup was initially created for the IJCAI 2023 competition RealAIGym[21]. This competition aims to explore the possibilities of swinging up and stabilizing an underactuated double pendulum in both simulation and the real world. For validation in the real world, strict position and speed restrictions are applied. The purpose of this section is to understand the logic behind all the involved subsystems and to set up the test environment using the existing components.

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.

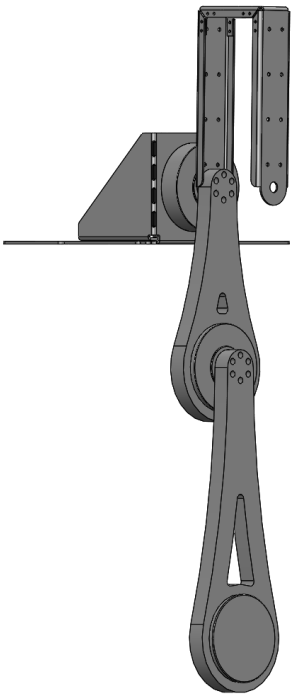
The mechanical design of the double pendulum underwent two iterations. The initial design was characterized by rotational imbalances due to a slightly misaligned motor axis and homogeneous link sizes, leading to vibrations and potential failures. Significant improvements were made in the second iteration: the original aluminum-plastic links were replaced with a carbon fiber-foam composite, and a lightweight, triangular design with central cutouts was introduced. These changes not only rectified the weaknesses of the previous design but also resulted in a mechanical structure that was markedly more reliable and safer, with enhanced yield strength and reduced safety risks.



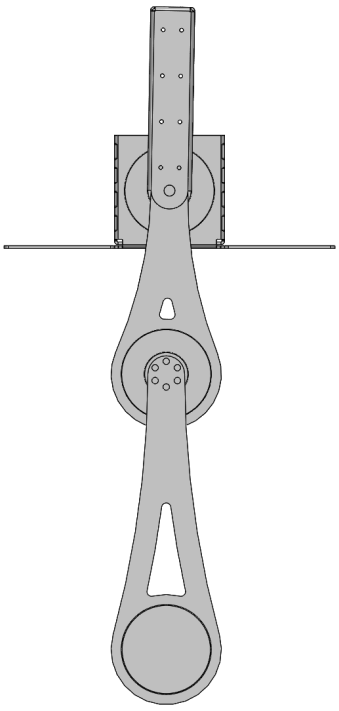
(a) Iteration 1 in isometric view



(b) Iteration 1 in front view



(c) Iteration 2 in isometric view



(d) Iteration 2 in front view

Figure 5.1: Double pendulum mechanical system iteration 1 and iteration 2

For actuation selection, quasi-direct drive (QDD) motors were chosen. QDDs are popular choices for actuation in robotics, commonly utilized in applications that demand both high torque and precise control, such as robotic arms or exoskeletons. They represent a compromise between direct drive systems, which connect the load directly to the motor without any gear reduction, and traditional geared systems, which use gears to increase torque at the cost of speed and may introduce backlash. The advantages of employing QDDs are apparent: they offer high precision and allow for precise control. Furthermore, the low gear ratio simplifies joint dynamics, which can typically be neglected when modeling the overall system dynamics. The drawbacks, however, are also evident. QDDs are costlier than average motors, and the low gear ratio limits the torque output.

Figure 5.2: AK80-6 V100 motor[2]

The AK80-6 V100 motors from CubeMars were selected for their ease of mounting, which is facilitated from both the front and rear ends[2]. As shown in Figure 5.3, these motors are characterized by a peak torque of 12 Nm and a rated torque of 6 Nm during continuous operation, aligning with the project's set torque limit of 5 Nm. They are designed to operate on a 24V voltage with a gear ratio of 6:1. Additionally, their design incorporates compatibility with both serial and CAN bus systems, which simplifies the development process.

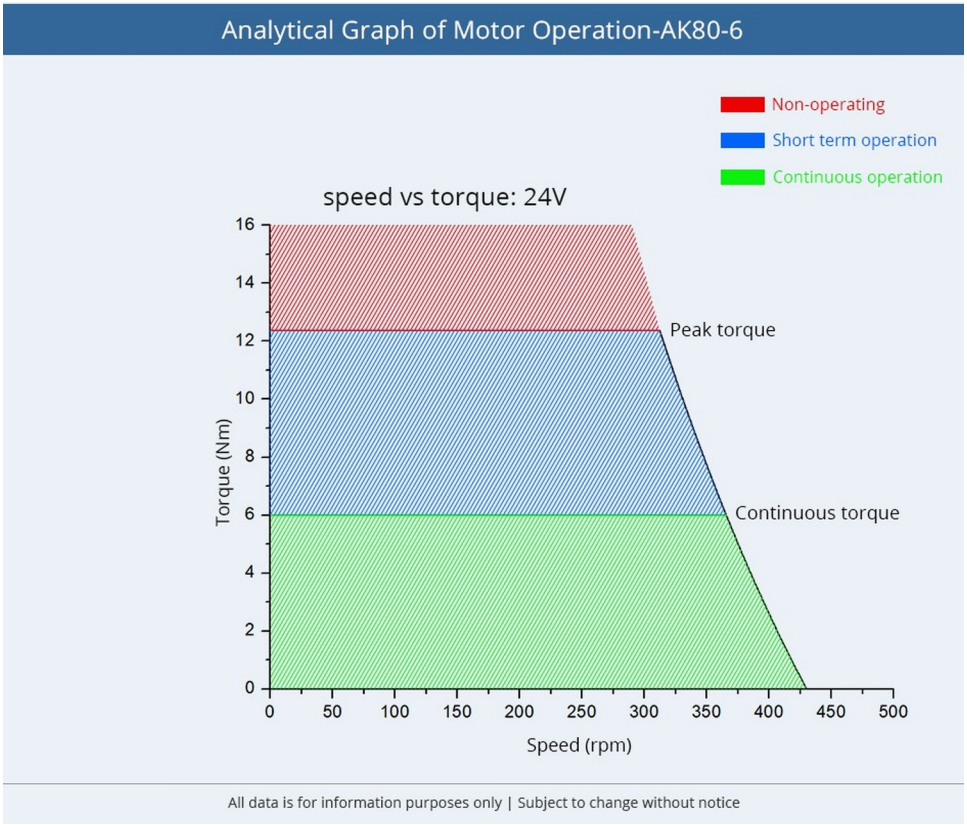


Figure 5.3: Torque speed curve of AK80-6 V100 motor[2]

For communication, the Controller Area Network (CAN) bus was chosen due to its compatibility with the actuators. Known for its robustness, flexibility, and efficiency, the CAN bus is a communication protocol that has been widely utilized in various applications. The adoption of the CAN bus for control brings numerous advantages. It provides error checking and fault confinement capabilities and supports real-time operation, facilitating high control frequencies with a relatively simple wiring arrangement.

In our application (Figure 5.4), the network comprises one master node (the PC) and two slave nodes (the motors). The control loop is constituted by a CAN-to-USB converter, a single CAN high cable, and a single CAN low cable, with termination resistors of 120Ω at both the initial and terminal points of the network.

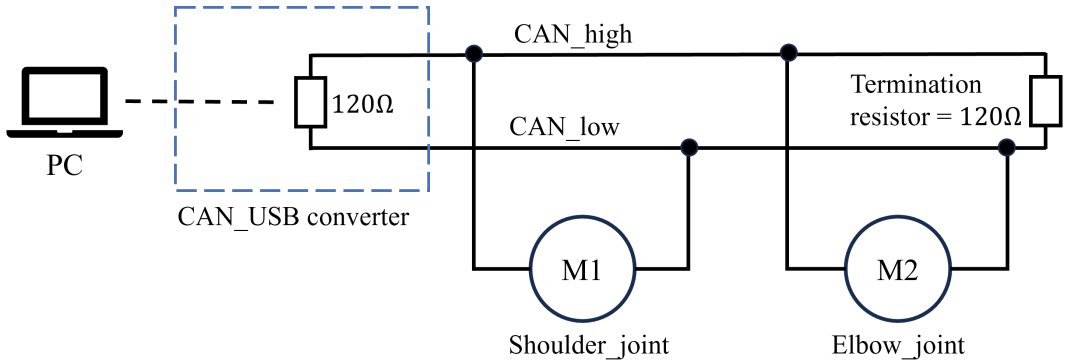


Figure 5.4: CAN connection diagram

To achieve a higher control frequency of approximately 500 Hz, the CAN-USB/2 product from ESD GmbH Hannover [16] was selected. This specialized interface exploits the USB 2.0 standard, which allows for a data rate of 480 Mbit/s, and features CAN capability at 1 Mbit/s as per ISO 11898-2. Furthermore, compatibility with the SocketCAN interface, which is incorporated into the Linux Kernel 2.6, is supported, thereby facilitating its use within a Linux development environment.

Figure 5.5: High speed CAN to USB interface[16]

Following accidents encountered during testing with the initial mechanical systems, several safety protocols have been instituted to safeguard human lives and equipment. Four principal measures are now in place.

Emergency stop:

An emergency stop has been interfaced directly with the 24V power supply. In the event that the double pendulum's behavior deviates from the expected parameters during tests, the power can be disconnected manually with immediate effect. Subsequently, the mechanical system's energy will dissipate swiftly, causing an automatic reversion to its initial state.

Capacitor:

In instances where the emergency stop is activated while the system operates at high velocities, the motors at the revolute joints serve as generators, converting the mechanical energy into electrical energy. This conversion process results in a current that is channeled back into the circuit, which, under extreme conditions, has the potential to overload the power supply. To mitigate this, a capacitor has been incorporated into the power supply circuitry to capture any excessive electrical energy that may arise from the abrupt cessation of the system's movement.

Figure 5.6: Capacitor

Speed and position limit:

Due to rules for the IJCAI 2023 competition RealAIGym[21], speed and position limits have been set at the software level. Due to the risk of vibrations and rotational imbalances at high speeds, which could lead to structural disassembly, a maximum speed limit of 20 rad/s has been implemented. If this speed threshold is exceeded, an automatic halt of the system will be triggered, functioning similarly to an emergency stop mechanism. Position limits have been set at 2π radians for both joints to prevent the risk of entanglement of CAN and power cables, which could cause interference and potential cable damage. A schematic of the entire wiring system is shown in Figure 5.8.

Physical enclosure:

A custom-designed enclosure(see Figure 5.7) has been fabricated to serve as a safeguard against unanticipated system failures. Constructed from aluminum profiles and reinforced with thick acrylic boards, the enclosure comprehensively surrounds the double pendulum setup, significantly mitigating the potential for accidents.

These safety measures have been crucial in reducing the risks associated with testing and operational procedures.

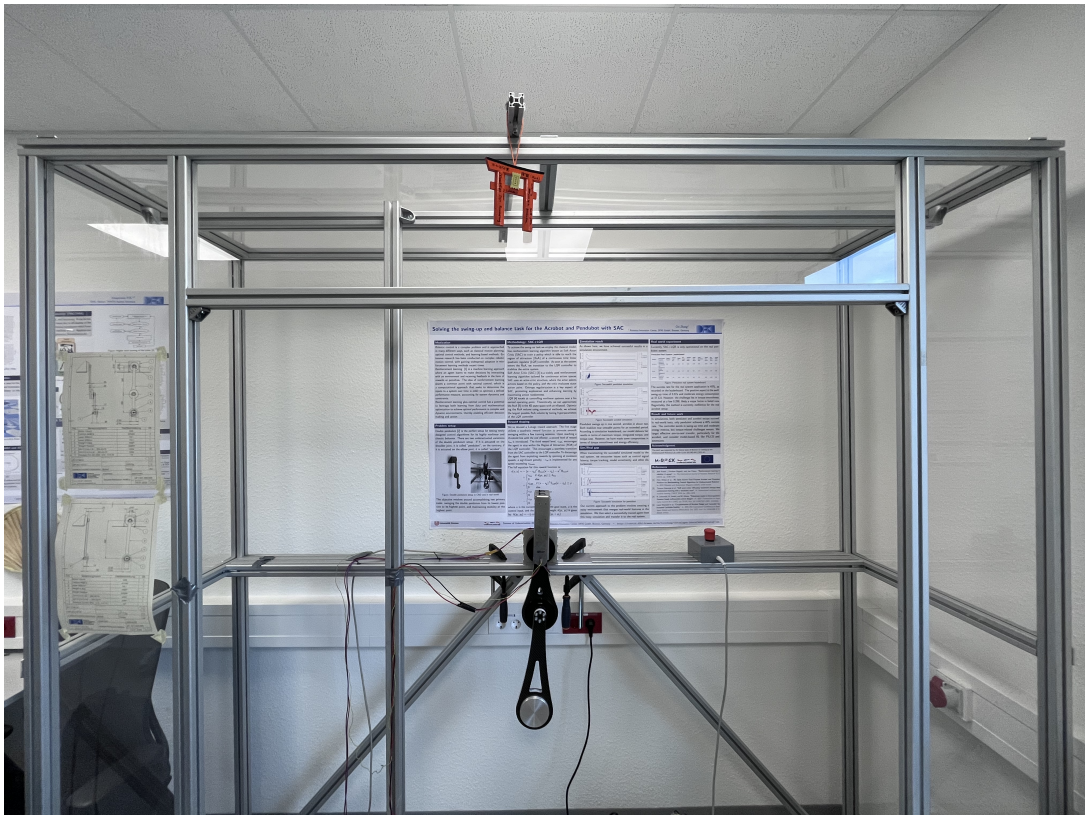


Figure 5.7: Physical enclosure for protection

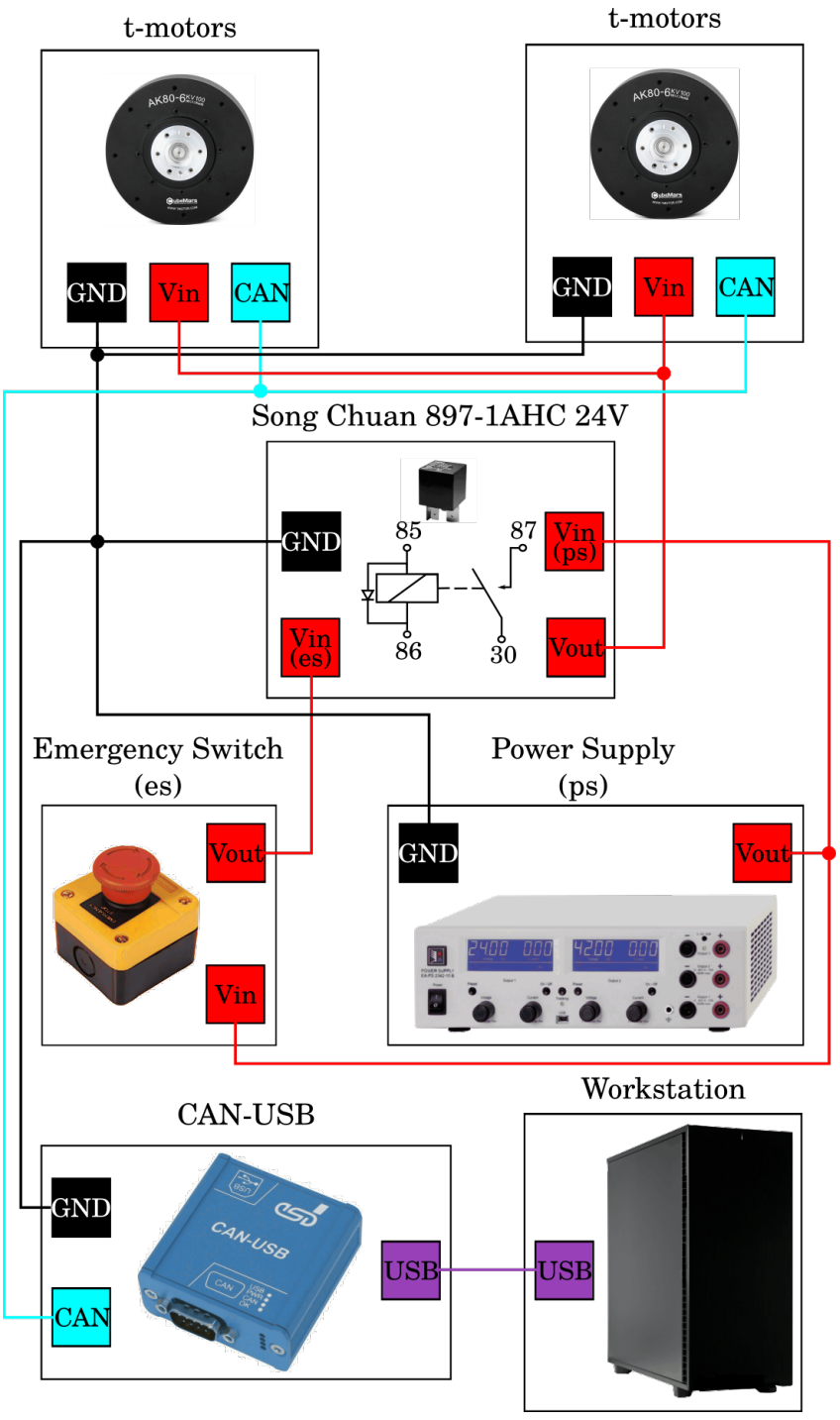


Figure 5.8: Wiring diagram[56]

5.2 System identification

Before implementing control strategies on a real system, it is crucial to ensure that the dynamics of the real system match those anticipated in theory. To achieve this, identifying the values used in the equation of motion is essential, and the system identification method is employed.

System identification is the process of deriving mathematical models of dynamic systems from observed input-output data. This method is widely used in control theory to analyze and predict the behavior of real-world systems. The common procedure of system identification includes model structure selection, data collection, and parameter estimation.

Model Structure Selection:

As described in Section 2.1.2, the equation of motion for the target system is derived using the Lagrangian method, as expressed in Equation 2.8. Fifteen parameters are required to fully describe the system dynamics, including masses (m_1, m_2), lengths (l_1, l_2), centers of masses (r_1, r_2), inertias (I_1, I_2) for the two links, and six actuator parameters, namely motor inertia I_r, gear ratio g_r, Coulomb friction (c_{f1}, c_{f2}), and viscous friction (b_1, b_2) for the two joints, and gravity g.

While the naturally provided parameters g and g_r are held constant, the easily measurable parameters l_1 and l_2 are measured by hand and recorded in Table 4.1. The remaining 11 system parameters need to be determined. We define the following terms as independent variables, which are to be estimated during the system identification experiment:

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

Data Collection:

System identification experiments are conducted by running excitation trajectories on the real hardware. The excitation trajectories include inputs of $[t, q, \dot{q}, \ddot{q}]^T$, where t is the timestamp, $q = [p_1, p_2]^T$ represents position, $\dot{q} = [v_1, v_2]^T$ represents velocity, and $\ddot{q} = [a_1, a_2]^T$ represents acceleration. Data tuples in the form $(q, \dot{q}, \ddot{q}, u)$ can be collected. One of the excitation trajectories are shown in Figure 5.9.

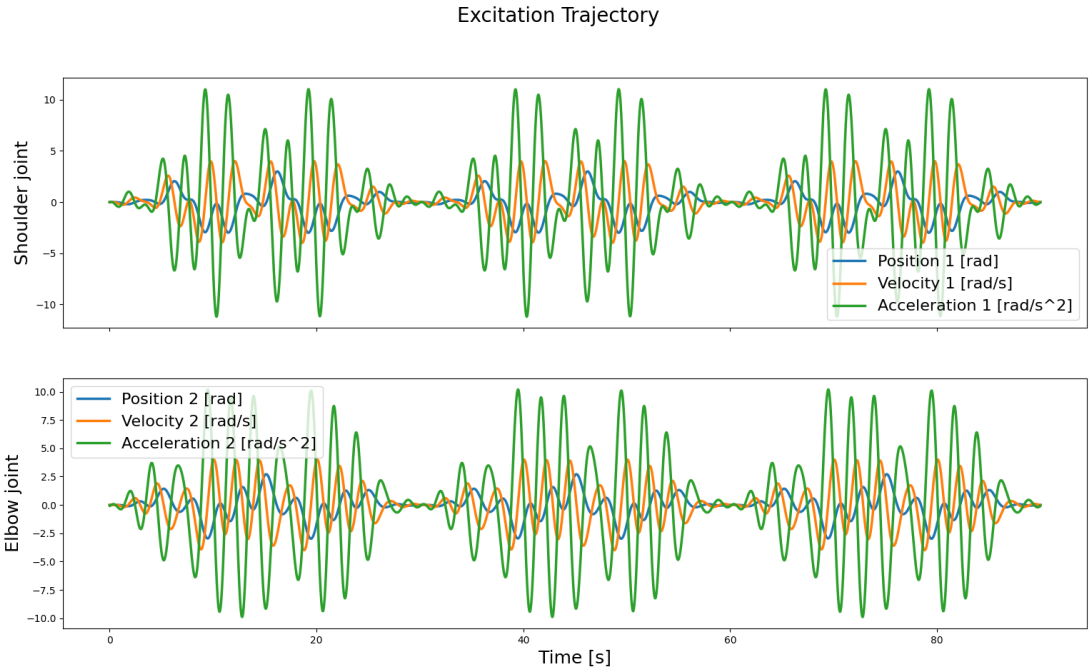


Figure 5.9: Excitation trajectory

Parameter Estimation:

To determine the most accurate system parameters, a least squares optimization can be performed on the recorded data. The objective of the least squares optimization is to find the values of model parameters that minimize the sum of squared differences between the observed output data and the model's predictions. The objective function can be expressed as:

$$J(\theta) = \sum (y_{\text{observed}} - y_{\text{predicted}}(\theta))^2 \tag{5.1}$$

Where:

- $J(\theta)$ is the cost function to be minimized.
- θ represents the vector of model parameters to be estimated.
- y_{observed} is the vector of observed output data.

- $y_{\text{predicted}}(\theta)$ is the vector of model predictions based on the current parameter values θ.

The least square fitting method was used to optimize the objective function, and the identified model parameters are shown in Table 5.1. These parameters are based on design C ($l_1 = 0.2m$, $l_2 = 0.3m$), which is used in real-world tests.

Parameter	Value
I_1	0.031887199591513114
I_2	0.05086984812807257
I_r	6.287203962819607e-05
b_1	0.001
b_2	0.001
c_{f1}	0.16
c_{f2}	0.12
g	9.81
g_r	6.0
l_1	0.2
l_2	0.3
m_1	0.5234602302310271
m_2	0.6255677234174437
r_1	0.2
r_2	0.25569305436052964

Table 5.1: Parameter values from system identification

5.3 Sim-to-Real transfer

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 two main 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.

5.3.1 Validation with noisy simulation environment

In addressing the challenge of sim-to-real transfer, multiple agents were trained using the Soft Actor-Critic (SAC) algorithm under similar setups in an ideal environment. These agents were then subjected to validation within a noisy environment, which is an ideal environment incorporated with real-world features. Only those agents demonstrating robustness to perturbations within this environment were advanced to real-world tests. Agents that did not withstand the noisy environment were deemed insufficiently robust and subsequently discarded.

In the course of real world experiments, four critical factors were identified that differ from ideal simulations: friction, measurement noise, latency, and torque responsiveness.

Friction:

Friction, identified as the predominant factor affecting performance, was not accounted for in the simulated training environment. To bridge this gap with real-world conditions, a friction compensation strategy was employed, starting with modeling the friction based on Coulomb's law as detailed in Equation 2.13.

Since frictional force counteracts the relative motion between contact surfaces, this compensation involved exerting additional torque in the same direction as the angular motion, providing the system with the necessary energy to overcome the effects of friction.

Despite the friction coefficients c_{f1} and c_{f2} being ascertained during the system identification phase, they were subsequently found to be imprecise during real system testing. To refine these coefficients, free-fall tests were conducted, which entailed releasing the double pendulum from a slight angular displacement and allowing it to descend under the force of gravity. Given the coupling influence of the two links, one joint was immobilized during the friction coefficient tuning of the other: the elbow joint remained static while estimating the shoulder joint's coefficient, and vice versa. Manual adjustments to the friction coefficients were made until the position output of the tested joint resulted in a sine wave without decay.

Measurement noise:

Measurement noise has been identified as the second most critical factor. The position displacement is measured with high accuracy using built-in encoders in the AK80-6 motors. However, the velocity measurement, which is derived as the first derivative of the position measurement, tends to introduce a relatively higher error. In the ideal environment, measurement error for both position and velocity is assumed to be non-existent. Nonetheless, this error is considerable in real-world applications.

To tackle this problem, the measurement error has been modeled as a normal distribution, with the mean representing the true velocity value and a manually adjustable standard deviation. The measurement noise vector is denoted by $\varepsilon = [\Delta p_1, \Delta p_2, \Delta v_1, \Delta v_2]^T$, and is assumed to follow a multivariate normal distribution:

$$\varepsilon \sim \mathcal{N}(\mu, \Sigma) \tag{5.2}$$

where μ is the vector of means, and Σ is the 4×4 covariance matrix, representing the uncertainty spread of the noise across each dimension. The measurements for the four states are presumed to be independent, rendering Σ a diagonal matrix. In practice, $\mu = 0$ and $\Sigma = \text{diag}(0, 0, 0.5, 0.5)$, indicating an omission of measurement noise on positions and a focus on the velocity measurement noise.

Latency:

Latency has been identified as the third significant factor. Given that any communication system requires time to transmit and receive data, and programs also require time to execute, latency is an inevitable aspect. Such latency presents a substantial challenge to control systems, especially those based on reinforcement learning. Reinforcement learning operates on the principles of Markov Decision Processes (MDPs), which follow the Markov property. According to this property, the future state of a MDP process is determined solely by the current state and action, and not by the sequence of states that preceded it. However, latency compromises the Markov property by inducing state mismatches and fostering dependencies on historical data. During free-fall tests, the maximum latency observed was 0.015 seconds. Therefore, this value has been established as the standard latency.

Torque responsiveness:

The fourth factor to be considered is torque responsiveness. In real hardware tests, it was observed that motors struggle to match the torque output with rapidly alternating control signals, particularly when there are significant differences between consecutive time steps. To model this phenomenon in noisy simulation, a discount coefficient c_{tr} is applied to the change in

control signals. The actual applied torque u_{real} is the sum of the previous torque u_{previous} plus the discounted change in torque $u_{\text{current}} - u_{\text{previous}}$. The calculation is expressed as follows:

$$u_{\text{real}} = u_{\text{previous}} + c_{tr} \cdot (u_{\text{current}} - u_{\text{previous}}) \tag{5.3}$$

When c_{tr} equals zero, the torque that is exerted on the system precisely mirrors the controller's output. A lower c_{tr} value makes it more difficult to apply significant changes to the control signal. Conversely, a policy that functions effectively with a low c_{tr} value indicates better torque smoothness. This also acts as an intuitive measure of the policy's capacity to yield smooth torque outputs. In most cases, c_{tr} is set to 0.85; however, to test the controllers' boundaries, values below 0.7 are also examined.

5.3.2 Noisy training based on domain randomization

For agents that succeeded in ideal simulations yet failed in noisy simulations, domain randomization has been identified as one method to enhance robustness.

Domain randomization [53], a technique conceived to narrow the gap between simulation and reality, aims to enable a model to operate effectively in real-world conditions, without the need for labeled real-world training data.

Initially emerging from the computer vision domain, domain randomization involves training a model not within a single, static simulation but rather within a diverse and perturbed environment. This is achieved by incorporating random variations into the ideal simulation. Techniques commonly employed in vision-based systems include changing the colors and textures of objects, modifying the lighting conditions, adjusting object shapes and sizes, introducing random noise to sensor data, and perturbing physical properties like friction or mass.

In the application of domain randomization to robotic manipulation tasks, disturbances introduced in the noisy simulation were employed to construct a noisy training process. This process resembles the previous ideal training process, but the environments for both the trial and evaluation phases were substituted with noisy environments. The training was warm-started with pretrained agents that had demonstrated success in ideal simulations yet performed poorly in noisy simulations.

The results of domain randomization in noisy training proved to be highly variable. Some agents significantly improved after a mere 5e6 iterations of noisy training, while others remained unchanged or worsened, with some even regressing to the point of failure in ideal simulations where they had once succeeded.

5.4 Results on real hardware

In this section, the results from the real hardware phase are presented. The content is divided into three parts. The first part discusses the procedure for selecting agents suitable for real-world testing. The second part displays the successful outcomes from an agent trained for the pendubot setup. The range of results is restricted to the pendubot setup due to the introduction of speed and position limits, which have significantly narrowed the range of possible policies. The third part discusses an unsuccessful agent in the acrobot setup. It passed ideal and noisy validations but cannot be transferred to real-world tests due to a violation of the 2π position limit.

5.4.1 Agent selection procedure for real world tests

An agent deemed suitable for real-system tests must undergo a process that includes training and validation in ideal environment, followed by validation in noisy environment. If the agent proves successful in noisy validation, it can proceed to further testing on the real system. If it does not succeed, we employ the domain randomization method in an attempt to enhance its performance. Should the retrained agent pass the noisy validation, it will then advance to real-system testing. Otherwise, it will be discarded.

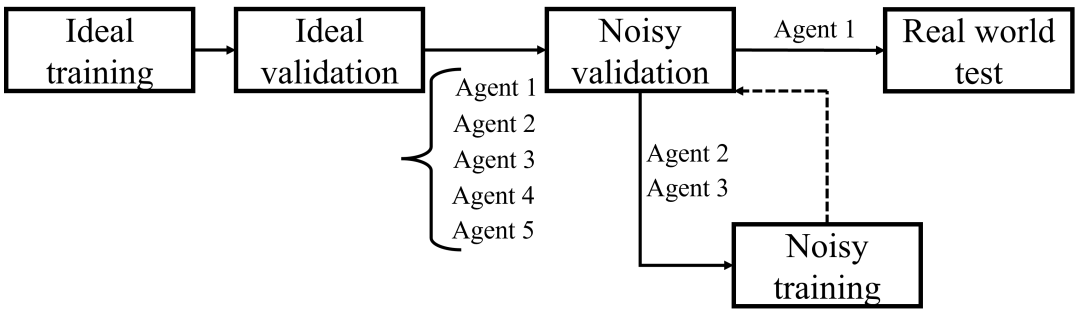


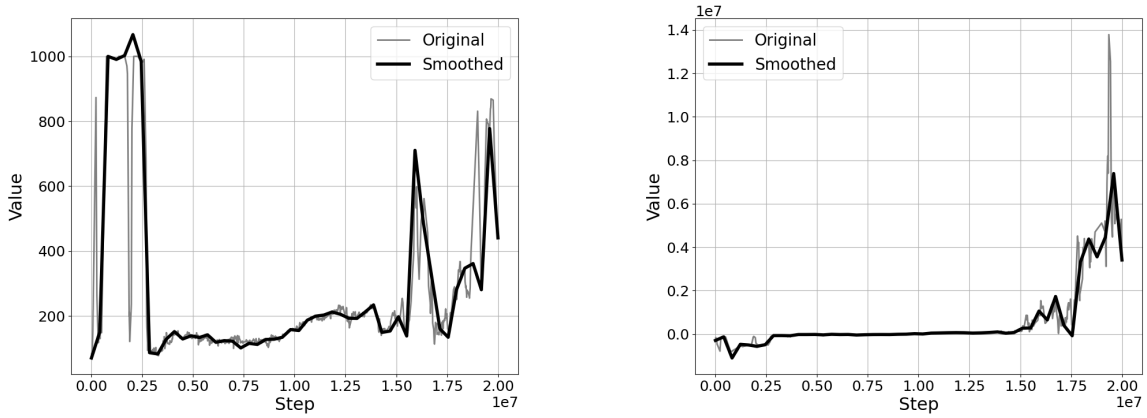
Figure 5.10: Agent selection procedure for real hardware tests

5.4.2 Pendubot results in real world

All real hardware experiments are based on design C ($l_1 = 0.2, l_2 = 0.3$). For the pendubot setup, obtaining a working agent is relatively straightforward. Slight modifications were made to the training process for the ideal simulation phase by deactivating the scaling mechanism mentioned in Section 4.1 and adding a termination condition that ends the training episode if the shoulder or elbow joint exceeds 2π, which means the agents are trained on unnormalized real physical state values with strict constraints on speed and velocity boundaries. Utilizing the parameters listed in Table 5.2, the training process successfully yielded a working model in the ideal simulation within 2e7 timesteps.

Robot	Quadratic Reward	Constant Reward	LQR
Pendubot	$Q_1 = 100$ $Q_2 = 100$ $Q_3 = 1.0$ $Q_4 = 1.0$ $R =$ 1e-2	$r_{line} = 1e3$ $r_{vel} = 0.0$ $r_{LQR} = 1e5$	$Q_1 = 0.97$ $Q_2 = 0.93$ $Q_3 = 0.39$ $Q_4 = 0.26$ $R = 0.11$

Table 5.2: Hyper parameters used for training pendubot agents for real world tests



(a) Mean episode length *(b) Mean episode reward*

Figure 5.11: Training curves of the working agent on pendubot

Figure 5.11 presents the training curve for the acquisition of an agent designated for real-world pendubot experiments, utilizing an ideal training process. As depicted in Figure 5.11a, the mean

episode length initially approaches the maximum of 1000, but then it rapidly descends to below 200 after 3e6 training steps. This descent indicates that the agent consistently attempts to rotate beyond 360 degrees in pursuit of higher rewards. A sign of emergence from this performance valley is observed at 15e6 training steps, with a subsequent significant increase in episode length. Correspondingly, the mean episode reward, as illustrated in Figure 5.11b, also shows a gradual increase before 15e6 time steps, followed by a rapid ascent thereafter.

The agent is put to the test in an ideal environment for validation. The results are 100% successful, as shown in Figure 5.12. The switching time between the SAC controller and the LQR controller occurs in less than 1 second. After taking over, the LQR controller is able to maintain balance around the goal state until the end of the experiment.

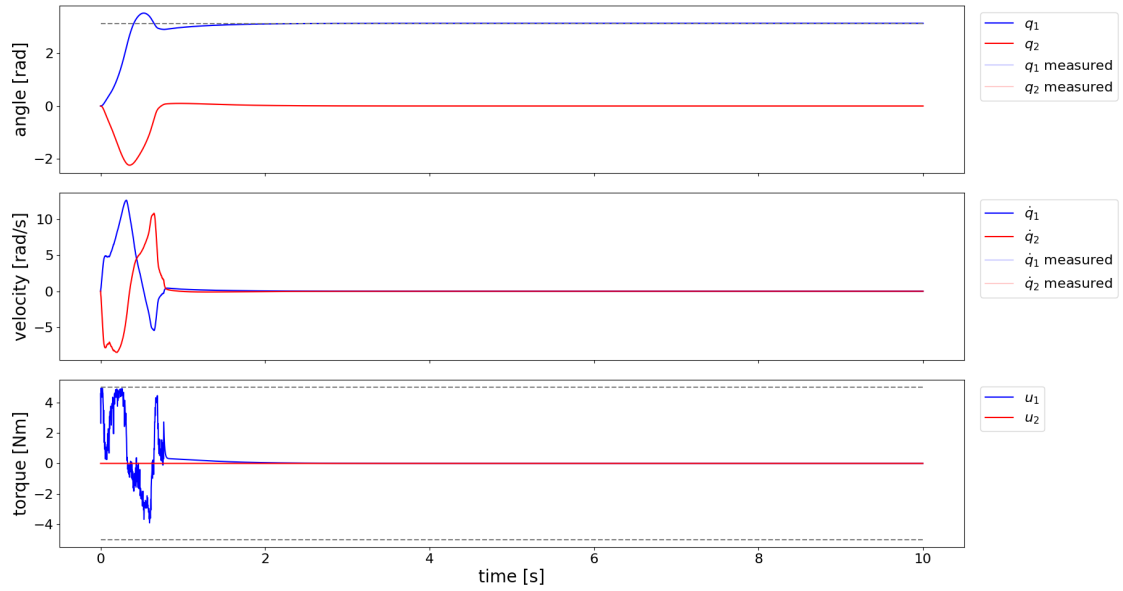


Figure 5.12: Pendubot result in the ideal environment

Subsequently, the agent was subjected to noisy validation. In a noisy simulation, the pretrained agent was found to succeed at rate of 40%, as demonstrated in Figure 5.13. In successful trials, the transition time between the SAC controller and the LQR controller occurred around 1 second. The LQR controller was still very effective in maintaining the system's stability. In unsuccessful trials, two scenarios occurred. Most often, the transition did not occur, and the system, remaining under the influence of the SAC controller, was inadequate for maintaining stability. In a few cases, although the transition did occur, the LQR controller failed to maintain the system's relative stable state around the upright position, resulting in experimental failure.

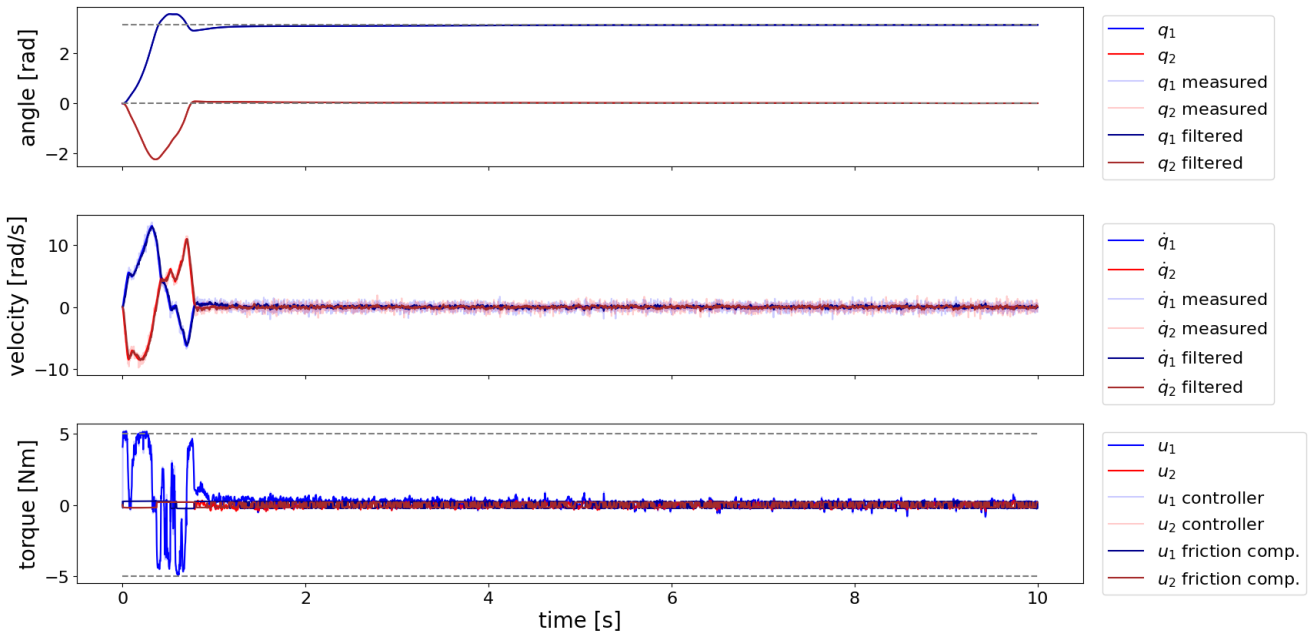


Figure 5.13: Pendubot result in the noisy environment

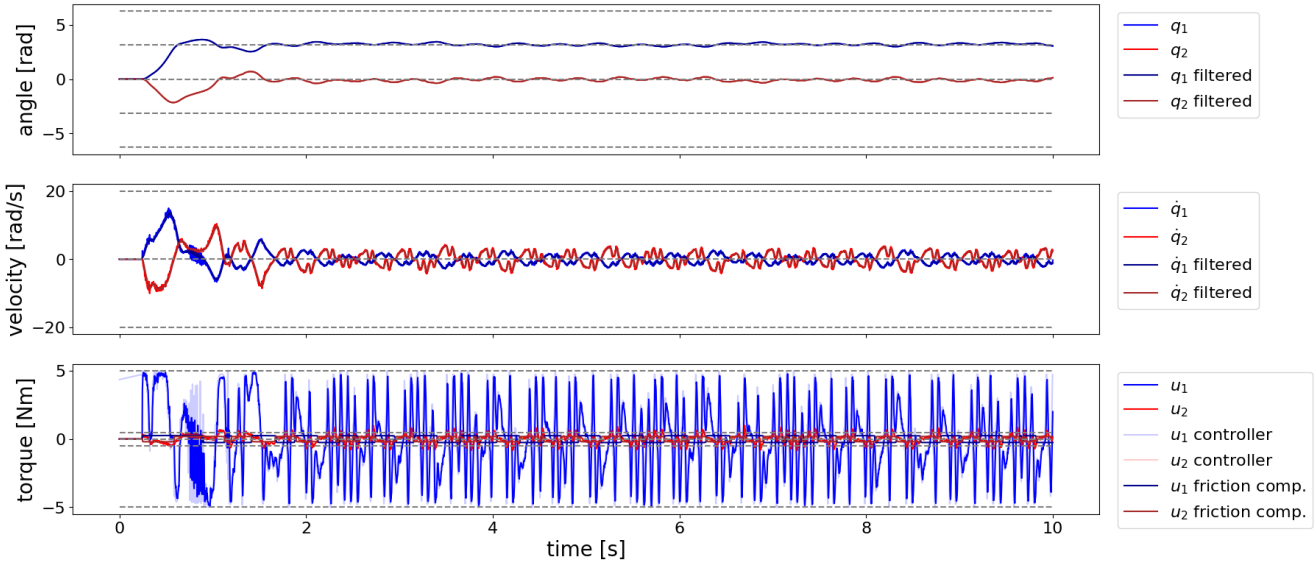


Figure 5.14: Result of a successful pendubot experiment on real system

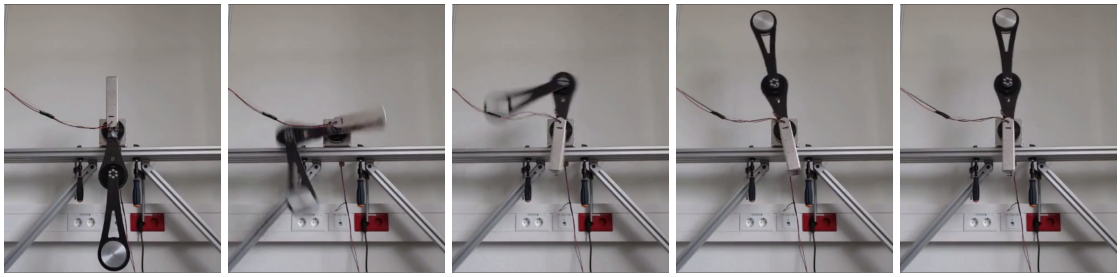


Figure 5.15: Snapshots in pendubot real world tests

Though the noisy validation only have 40% success rate, we consider the above shown agent ready for a real-world test, so we deploy it onto the test bench for the real world evaluation. A control frequency of 400Hz was utilized. The agent's average success rate in the real system was 40%, aligning with the success rate observed during noisy validation. A depiction of one such successful outcome is presented in Figure 5.14.

The transition to the LQR controller occurred smoothly at approximately one second. Upon assuming control, the LQR controller succeeded in sustaining the system's upright position, albeit with vibrations, until the completion of the experiment. Contrary to the results in both noisy and ideal simulations, the torque applied by the LQR controller was noticeably less smooth, resulting in greater amplitude of position and velocity vibrations. Given that the feature of self-stabilization had been evident during the ideal validation phase, an attempt was made to omit the LQR controller, allowing the agent to execute the swing-up and stabilize independently; however, this approach proved unsuccessful.

In unsuccessful trials, similar problems that occurred during noisy validation recurred. The system either failed to switch from the SAC controller to the LQR controller, or the LQR controller was unable to maintain stability. The latter scenario is more likely to occur in real-world tests than in noisy validation.

5.4.3 Acrobot results in real world

Training for the acrobot setup has encountered numerous challenges, which can be summarized into the following aspects:

Violation of Speed and Position Limit:

The introduction of new rules in IJCAI RealAIGym competition[21], with a speed limit set at 20 rad/s and a position limit set at 2π, has made it more challenging to train a functional agent in the acrobot setup. Attempts have been made to train agents using unscaled real physical state values with a termination condition aimed at keeping the agent within the position limit; however, this approach did not perform as well as it did with the pendubot setup. Despite the application of 5e7 timesteps (which takes approximately 5 hours), neither the mean episode length nor the mean episode reward exhibited growth. A return to the method of training based on scaled state values without termination conditions eventually yielded a working model in ideal validation, although the 2π position limit was violated.

Inconsistency in Training:

The training of acrobot agents has proven to be highly unstable. Despite the use of consistent and carefully tuned hyperparameters, training success has varied. There are instances when significant growth in mean episode reward suggests that training is progressing correctly, yet the resulting agent may perform inconsistently in ideal simulations.

Long Training Hours:

When compared to pendubot training, exploration in the acrobot setup often requires more time. While pendubot training typically produces a functional agent within 2e7 timesteps, acrobot training may take between 3e7 to 4e7 timesteps, with no guarantee that the resulting agent will pass an ideal validation. Such iterations, taking 3-4 hours each, are considerably time-consuming.

The most favorable outcome in the acrobot setup has been an agent that functions effectively in both ideal and noisy validations. It was trained using parameters detailed in Table 5.3, with an active scaling mechanism and in the absence of termination conditions, indicating that the agent was trained using normalized state values.

Robot	Quadratic Reward	Constant Reward	LQR
Pendubot	Q_1 = 100	$r_{line} = 1e3$	Q_1 = 1.92
	Q_2 = 90	$r_{vel} = 1e4$	Q_2 = 1.92
	Q_3 = 1.0	$r_{LQR} = 1e5$	Q_3 = 0.3
	Q_4 = 1.0		Q_4 = 0.3
	R = 1e-2		R = 0.82

Table 5.3: Hyper parameters used for training acrobot agents for real world tests

As shown in Figure 5.16, the reward for this agent experienced a smooth increase up to 2.5e7 timesteps and then a rapid ascent in the final 5 million timesteps. Because a termination condition was not implemented, the episode length remained consistently at 1000. This outcome was anticipated and led to a working agent in an ideal environment.

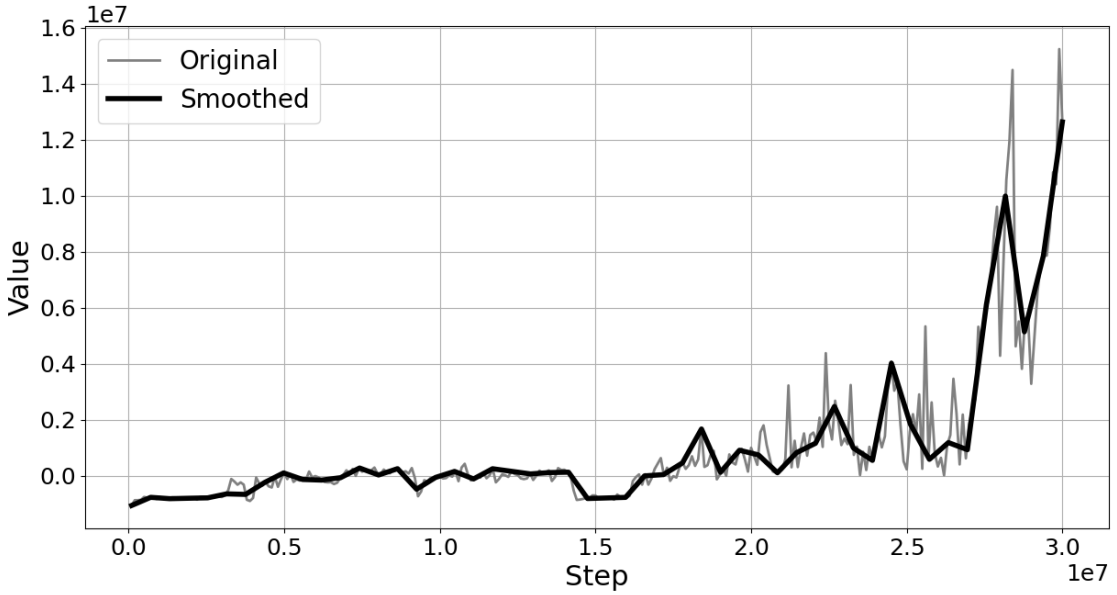


Figure 5.16: Training curve of best result in acrobot setup

The behavior of this agent during ideal validation is shown in Figure 5.17. The swing-up is performed by the SAC controller within 2 seconds. After the LQR controller takes over, the system maintains stability around the highest position for a prolonged period. The ideal validation is considered successful.

The agent operates successfully within a noisy simulation after ideal validation, eliminating the need for further noisy training. The success rate is approximately 50%. Figure 5.18 shows one of the successful tests.

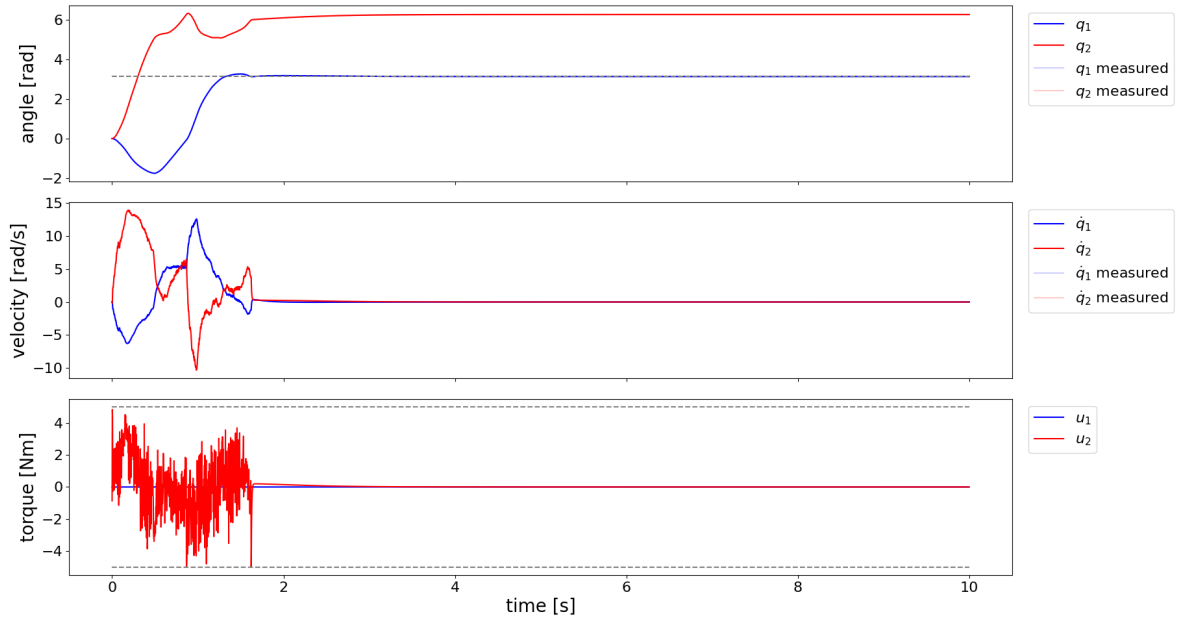


Figure 5.17: Acrobot result in ideal environment

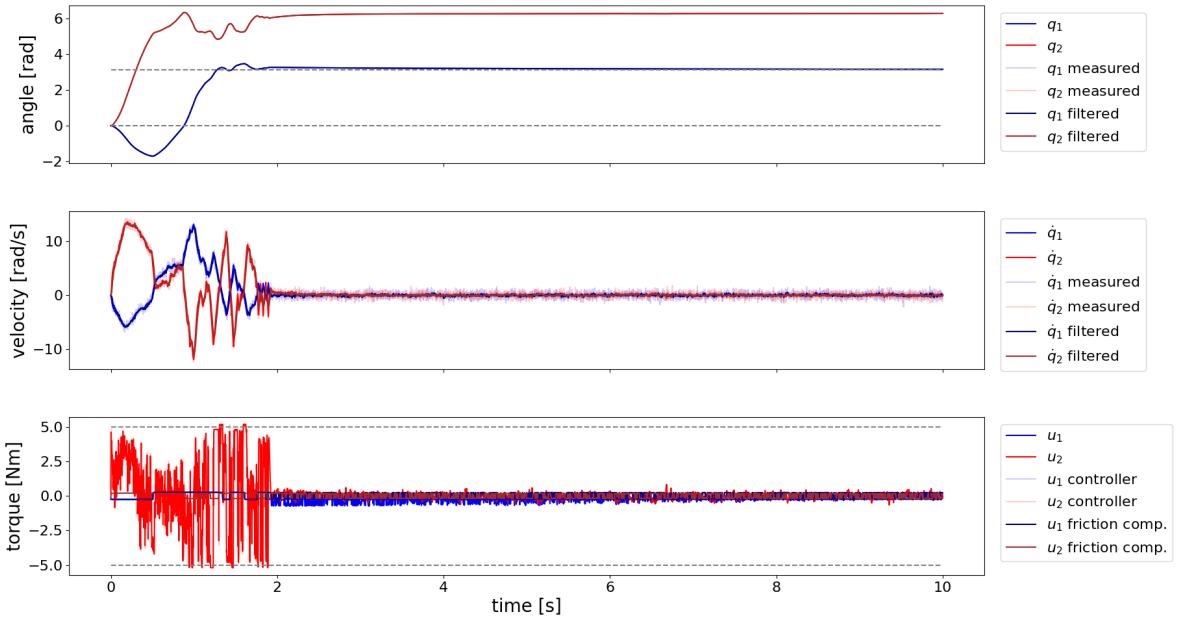


Figure 5.18: Acrobot result in noisy environment

In successful trials, the swing-up phase conducted by the SAC controller concludes within 2 seconds. Although the torque applied in ideal environment tests is already quite noisy, the torque in a noisy environment exhibits even greater fluctuations. Once the LQR controller takes effect, the system stabilizes around the goal state until the experiment concludes. In unsuccessful trials, the inability to switch to the LQR controller is the primary issue. But the agent has high recovery ability, the system sometimes misses the first transition oppotunity and making several rotations before it finds itself reaching a point for LQR controller to take over.

Due to the agent's violation of the 2π position limit, it has been deemed unsuitable for real-world tests. Consequently, no additional results from real-world tests in the acrobot setup are presented.

6 Discussion

In this chapter, the results of experiments conducted in both simulation and on the real system are delved into. The chapter is structured as follows: The first section discusses the simulation leaderboard,the second section discusses the robustness leaderboard, and the third section discusses real hardware leaderboard.

6.1 Interpretation of simulation leaderboard

In Table 6.1 and Table 6.2, the performance leaderboard results for both the pendubot and acrobot in simulation experiments are presented. Three types of controllers are listed for comparison, namely model-free reinforcement learning(MFRL) based controller, model-based reinforcement learning(MBRL) based controller and trajectory based controller.

The SAC+LQR controller is our design and is a representative of model-free reinforcement learning method.

MC-PILCO [4], which stands for Monte Carlo Probabilistic Inference for Learning Control, is a model-based reinforcement learning method. It utilizes probabilistic models to predict the system's dynamics and employs Monte Carlo methods to optimize control policies based on these predictions. This method was implemented by a team from the University of Padova [34].

tvLQR is an extension of the standard Linear Quadratic Regulator (LQR) control design. It is a trajectory based control method, tailored for systems with time-dependent state-space matrices or where the optimal control needs to be dynamic. Representing the optimal control method, it was implemented by a separate team from DFKI RIC[56].

Criteria	SAC+LQR	MC-PILCO	tvLQR
Swingup Success	success	success	success
Swingup time [s]	**0.65**	1.43	4.2
Energy [J]	9.4	12.67	**9.06**
Max. Torque [Nm]	5.0	**2.4**	2.82
Integrated Torque [Nm]	**2.21**	3.48	2.57
Torque Cost [N²m²]	8.58	7.77	**2.0**
Torque Smoothness [Nm]	0.172	0.07	**0.031**
Velocity Cost [m²/s²]	**44.98**	94.68	137.31
RealAI Score	0.801	**0.891**	0.827

Table 6.1: Performance scores of various controllers for the Pendubot experiment.

Criteria	SAC+LQR	MC-PILCO	tvLQR
Swingup Success	success	success	success
Swingup time [s]	2.06	**1.1**	3.98
Energy [J]	29.24	**9.81**	10.92
Max. Torque [Nm]	5.0	**2.82**	5.0
Integrated Torque [Nm]	4.57	**1.27**	2.27
Torque Cost [N²m²]	12.32	**2.27**	2.47
Torque Smoothness [Nm]	0.954	**0.057**	0.077
Velocity Cost [m²/s²]	193.78	242.44	**100.34**
RealAI Score	0.722	**0.869**	0.8

Table 6.2: Performance scores of various controllers for the Acrobot experiment.

All three controllers are successful with both the Pendubot and Acrobot setups.

In the Pendubot setup, the performance of the SAC+LQR controller is commendable, particularly with a swift swing-up time of 0.65s. The energy consumption of the SAC+LQR controller (9.4J) is significantly lower than that of the MC-PILCO controller (12.67J) and is nearly on par with the tvLQR controller (9.06J). The integrated torque of SAC+LQR controller is also the lowest. Additionally, its overall RealAI score is competitive, closely trailing the scores of MC-PILCO and tvLQR. However, a notable drawback is its torque smoothness; it performs the worst among the three controllers, being 2.46 times that of MC-PILCO and 5.55 times that of tvLQR.

For the Acrobot setup, the SAC+LQR controller loses its edge in both swing-up time and energy consumption. Its deficit in torque smoothness becomes even more pronounced, leading to a considerably lower RealAI score compared to the other two controllers.

In general, the SAC+LQR controller demonstrates competitive performance in simpler tasks, such as the pendubot, especially excelling in swing-up time. However, when faced with a more complex challenge like the Acrobot, its performance declines. The MC-PILCO consistently delivers the best overall performance across both setups and is notable for its remarkably low maximum torque input and consistent torque smoothness. Conversely, the tvLQR, a trajectory based method, highlights its effectiveness in both scenarios. While its swing-up time is relatively extended, its energy consumption and torque smoothness are commendably low, leading to a moderate RealAI score.

6.2 Interpretation of robust leaderboard

The results of robustness leaderboard is shown in Table 6.3. A visualization is shown in Figure 6.1. In comparison, the SAC+LQR controller achieves a moderate overall robustness score among the three controllers. It exhibits a higher resistance to model inaccuracy (71.9% for pendubot and 76.7% for acrobot) compared to MC-PILCO (45.2% for pendubot and 40.5% for acrobot) and tvLQR (75.2% for pendubot and 59.0% for acrobot). While the other two controllers demonstrate a noticeable decline when tackling the more complex acrobot task, the performance of the SAC+LQR remains consistent. Additionally, SAC+LQR offers better resistance against velocity measurement noise compared to MC-PILCO, though the top score in this category is held by tvLQR. Apart from MC-PILCO, the other two controllers display consistent and strong robustness regarding torque noise and torque response. When considering time delay, tvLQR outperforms both SAC+LQR and MC-PILCO owing to its nature as a trajectory-based controller, which is less affected by the Markov decision process.

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.

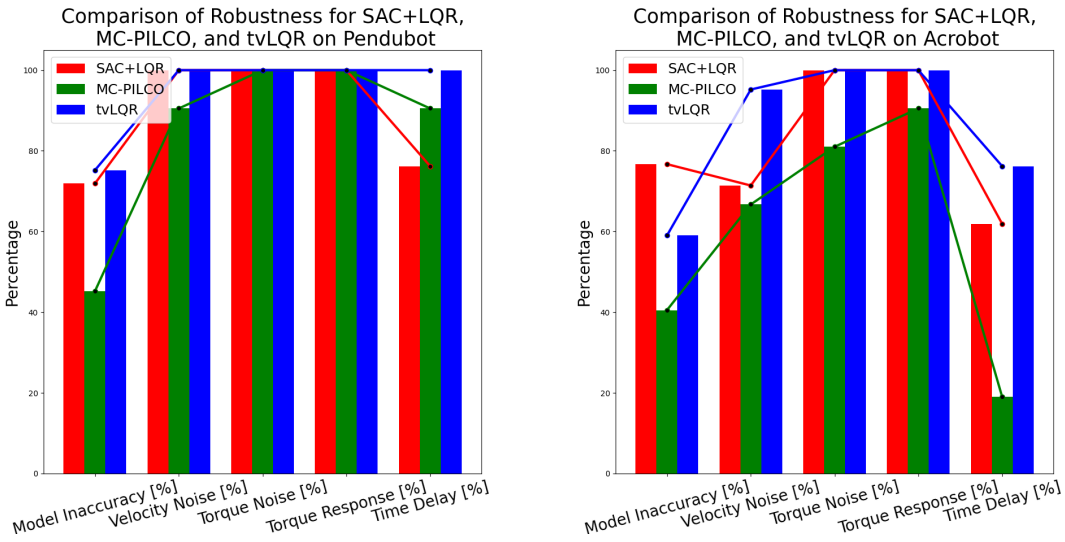


Figure 6.1: Visualization of Robustness Scores on Pendubot and Acrobot

In general, tvLQR achieves the best robustness scores for both pendubot and acrobot setups, followed by SAC+LQR, with MC-PILCO ranking last. While SAC+LQR boasts consistency in robustness across both setups, time delay remains a significant issue, limiting the robustness of RL-based methods.

6.3 Interpretation of real system leaderboard

The results of real system performance leaderboard is presented in Table 6.4. For the pendubot, SAC+LQR achieved a swing-up success rate of 40%, while MC-PILCO had a perfect score of 100%, and tvLQR scored 80%. For the acrobot, SAC+LQR did not achieve success, MC-PILCO again scored 100%, and tvLQR achieved full success as well. The swing-up time was fastest with SAC+LQR for the pendubot at 0.67 seconds, and for the acrobot, MC-PILCO had the fastest time at 1.55 seconds.

Apart from the swing-up time, MC-PILCO demonstrates superior performance on the pendubot, while tvLQR has the advantage for the acrobot. MC-PILCO's energy consumption on the pendubot is markedly lower, with marginally better maximum torque and a significant lead in

integrated torque and torque cost metrics. The velocity cost further showcases MC-PILCO's high efficiency, contributing to its leading average RealAI score of 0.839.

Criteria	SAC+LQR		MC-PILCO		tvLQR	
	Pendubot	Acrobot	Pendubot	Acrobot	Pendubot	Acrobot
Swingup Success	4/10	0/10	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 [N²m²]	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 [m²/s²]	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.

In the acrobot trials, the scores are close between MC-PILCO and tvLQR. However, tvLQR outperformed MC-PILCO in terms of energy consumption and most torque-related criteria by a substantial margin. Additionally, tvLQR's torque smoothness is notably superior to that of MC-PILCO. tvLQR also achieved the highest average RealAI score of 0.821 for the acrobot.

In summary, while MC-PILCO displayed high efficiency and success rates for the pendubot, tvLQR excelled in torque smoothness and energy efficiency, particularly for the acrobot system. The SAC+LQR approach demonstrated rapid swing-up times for the pendubot but failed to register success for the acrobot.

7 Conclusion and Future work

In this chapter, a conclusion of our work is presented, summarizing the controller design, experiments in simulation and the real world, and the leaderboard results. This is followed by a discussion of the future work that could not be completed during the six-month master's thesis period.

7.1 Conclusion

Combining an SAC-derived agent with an LQR controller is an effective method for performing swing-up and stabilization tasks for an underactuated double pendulum system in simulation, and it is partially effective in real-world tests.

During the ideal simulation phase, the training of an agent capable of swinging up and entering the Region of Attraction (RoA) of the LQR controller is stabilized using our custom three-stage reward function. The training for the pendubot setup generally requires 2e7 timesteps, and for the acrobot, it varies from 3e7 to 5e7 timesteps in total. Notably, the most significant challenge lies in hyperparameter tuning to ensure stable training that produces a functional agent. While the training for the pendubot and acrobot only takes a few hours, the tuning process can extend for several days for each design variation. The LQR controller exhibits flawless performance in ideal simulations; upon taking control, it quickly guides the system to the desired state and maintains stability until the experiment concludes. In comparison, the acrobot setup presents more challenges than the pendubot due to the longer training requirements and less consistent outcomes.

In real-world tests, which confront real-world complexities and added constraints such as speed and position limits, our SAC+LQR method encounters significant challenges. It delivers only one working solution for the pendubot, achieving a success rate of 40%. For the acrobot setup, despite the presence of multiple well-performing candidates, each one is ultimately discarded due to exceeding the 2π position limit. The sim-to-real gap substantially affects the performance of the SAC+LQR controllers when they are tested exclusively in an ideal simulation. The SAC agents tend to utilize control signals with less smoothness and struggle to determine the precise moment for the LQR controller to assume control amid various disturbances. Moreover, LQR

controllers do not always perform perfectly in real-world applications. There are possibilities of failing to maintain stability after taking over.

An attempt to bridge the sim-to-real gap includes the establishment of a noisy simulation for validation and a noisy training process to enhance robustness. This noisy simulation builds upon the ideal simulation, incorporating real-world features such as friction, measurement noise, latency, and torque responsiveness. The noisy training process employs domain randomization techniques, aiming to improve robustness by exposing agents to variable environments. Furthermore, agents that are proven effective in ideal simulations must undergo a selection process (Figure 5.10) before being considered suitable for real-world tests.

In terms of final RealAI scores, the SAC+LQR controller ranks last among the three methods discussed in terms of performance in both simulation and real-world tests. However, the SAC+LQR controller ranks medium in robustness scores.

In conclusion, the SAC+LQR controller performs adequately in simulation environments but exhibits flaws in real-world applications. Our current methods to bridge the sim-to-real gap lack effectiveness and require further improvements.

7.2 Future Work

Due to the ineffectiveness of our method in real-world tests, several aspects of future work are worth exploring.

Modify the Training Process for More Accurate Behavior Guidance:

The unsuccessful training of an agent for the acrobot setup within speed and position limits has highlighted the necessity for an improvement in behavior guidance during training. Our current reward function only indicates to the agent to swing up and enter the Region of Attraction (RoA) of a predefined LQR controller; it doesn't provide detailed instructions on how the swing-up should be executed. Future work could base the reward function and termination conditions on mirroring a feasible trajectory within constraints.

Model-Based Reinforcement Learning:

As indicated in Tables 6.1,6.2 6.3, and 6.4, the MC-PILCO controller, as a representative of model-based reinforcement learning (MBRL) methods, delivers astonishing results. Although

MBRL methods are still in their infancy, the idea of combining transition model information with pure trial-and-error shows high potential for solving complex issues like chaotic system control in real-world applications. This direction holds the most promise for achieving significant improvement in our current control problems with pendubot and acrobot setups.

More Effective Sim-to-Real Methods

The results presented in Table 6.4 indicate a considerable need for improvements in real-world tests when SAC+LQR control is employed. The sim-to-real gap poses a challenge that limits the practical application of the SAC+LQR controller in real-world scenarios. Addressing this issue may involve several potential strategies:

Firstly, the integration of more accurate real-world features into the learning process should be considered to enable agents to adapt to real-world complexities through trial and error, thereby enabling a smoother transition to actual systems.

Secondly, the training of agents could be approached using direct real-world data, or by combining agent training with on-site testing on actual hardware, necessitating the use of highly sample-efficient algorithms.

Lastly, the development of a mapping mechanism for translating results from idealized environments to the real world could be advantageous. Such a mechanism could be embodied in a neural network-based mapping function that processes state information from both simulated and real environments and actions generated for the ideal environment, outputting actions suitable for the real-world system.

Bibliography

[1] J. Achiam. *Spinning up in deep reinforcement learning*. 2018.

[2] *AK80-6 Robotic Actuator*. https://www.cubemars.com/goods-981-AK80-6.html. Accessed: 2023-11-03. 2023.

[3] T. Albahkali, R. Mukherjee, and T. Das. "Swing-up control of the pendubot: an impulse–momentum approach". In: *IEEE Transactions on Robotics* 25.4 (2009), pp. 975–982.

[4] F. Amadio, A. Dalla Libera, R. Antonello, D. Nikovski, R. Carli, and D. Romeres. "Model-based policy search using monte carlo gradient estimation with real systems application". In: *IEEE Transactions on Robotics* 38.6 (2022), pp. 3879–3898.

[5] J. L. Ba, J. R. Kiros, and G. E. Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[6] J. T. Betts. "Survey of numerical methods for trajectory optimization". In: *Journal of guidance, control, and dynamics* 21.2 (1998), pp. 193–207.

[7] L. Biagiotti and C. Melchiorri. *Trajectory planning for automatic machines and robots*. Springer Science & Business Media, 2008.

[8] W. Bickley. "Piecewise cubic interpolation and two-point boundary problems". In: *The computer journal* 11.2 (1968), pp. 206–208.

[9] P. Biswal and P. K. Mohanty. "Development of quadruped walking robots: A review". In: *Ain Shams Engineering Journal* 12.2 (2021), pp. 2017–2031.

[10] A. Bogdanov. "Optimal control of a double inverted pendulum on a cart". In: *Oregon Health and Science University, Tech. Rep. CSE-04-006, OGI School of Science and Engineering, Beaverton, OR* (2004).

[11] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[12] B. S. Cazzolato, Z. Prime, et al. "On the dynamics of the furuta pendulum". In: *Journal of Control Science and Engineering* 2011 (2011).

[13] X. Cui and H. Shi. "A*-based pathfinding in modern computer games". In: *International Journal of Computer Science and Network Security* 11.1 (2011), pp. 125–130.

[14] DFKI RIC Underactuated Lab. *Double Pendulum Dynamics*. Website. 2023. URL: https://dfki-ric-underactuated-lab.github.io/double_pendulum/dynamics.html.

[15] W. Ditto and T. Munakata. "Principles and applications of chaotic systems". In: *Communications of the ACM* 38.11 (1995), pp. 96–102.

[16] ESD Electronics. *CAN-USB/2*. https://esd.eu/en/products/can-usb-2. Accessed: yyyy-mm-dd. 2023.

[17] 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* 206.4 (1992), pp. 263–269.

[18] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni. "Path planning and trajectory planning algorithms: A general overview". In: *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches* (2015), pp. 3–27.

[19] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni. "Trajectory planning in robotics". In: *Mathematics in Computer Science* 6 (2012), pp. 269–279.

[20] A. Gasparetto and V. Zanotto. "Optimal trajectory planning for industrial robots". In: *Advances in Engineering Software* 41.4 (2010), pp. 548–556.

[21] German Research Center for Artificial Intelligence (DFKI). *Real AI Gym*. https://dfki-ric-underactuated-lab.github.io/real-ai-gym/. Accessed: yyyy-mm-dd. 2023.

[22] S. Gillen, M. Molnar, and K. Byl. "Combining deep reinforcement learning and local control for the acrobot swing-up and balance task". In: *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE. 2020, pp. 4129–4134.

[23] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[24] T. Hiraoka, T. Imagawa, T. Hashimoto, T. Onishi, and Y. Tsuruoka. "Dropout q-functions for doubly efficient reinforcement learning". In: *arXiv preprint arXiv:2110.02034* (2021).

[25] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4.26 (2019), eaau5872.

[26] J. Iqbal, M. Ullah, S. G. Khan, B. Khelifa, and S. Ćuković. "Nonlinear control systems-A brief overview of historical and recent advances". In: *Nonlinear Engineering* 6.4 (2017), pp. 301–312.

[27] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control". In: *arXiv preprint arXiv:1708.04133* (2017).

[28] H. Khalil. *Nonlinear Systems*. Pearson Education. Prentice Hall, 2002. URL: https://books.google.de/books?id=t_d1QgAACAAJ.

[29] J. Kober, J. A. Bagnell, and J. Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[30] V. Konda and J. Tsitsiklis. “Actor-critic algorithms”. In: *Advances in neural information processing systems* 12 (1999).

[31] B. Kouvaritakis and M. Cannon. “Model predictive control”. In: *Switzerland: Springer International Publishing* 38 (2016).

[32] A. Kumar, Z. Fu, D. Pathak, and J. Malik. “Rma: Rapid motor adaptation for legged robots”. In: *arXiv preprint arXiv:2107.04034* (2021).

[33] N. Lehtomaki, N. Sandell, and M. Athans. “Robustness results in linear-quadratic Gaussian based multivariable control designs”. In: *IEEE Transactions on Automatic Control* 26.1 (1981), pp. 75–93.

[34] D. Libera, A. Turcato, N. Giacomuzzo, G. Carli, R. Romeres, A. D. Libera, N. Turcato, G. Giacomuzzo, et al. “Athletic Intelligence Olympics challenge with Model-Based Reinforcement Learning”. In: 2023. URL: https://api.semanticscholar.org/CorpusID:261487429.

[35] Y. Liu and H. Yu. “A survey of underactuated mechanical systems”. In: *IET Control Theory & Applications* 7.7 (2013), pp. 921–935.

[36] T. Luukkonen. “Modelling and control of quadcopter”. In: *Independent research project in applied mathematics, Espoo* 22.22 (2011).

[37] K. M. Lynch and F. C. Park. *Modern robotics*. Cambridge University Press, 2017.

[38] L. J. Maywald, F. Wiebe, S. Kumar, M. Javadi, and F. Kirchner. “Co-optimization of Acrobot Design and Controller for Increased Certifiable Stability”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 2636–2641.

[39] C. Oestreicher. “A history of chaos theory”. In: *Dialogues in clinical neuroscience* (2007).

[40] X. B. Peng, P. Abbeel, S. Levine, and M. Van de Panne. “Deepmimic: Example-guided deep reinforcement learning of physics-based character skills”. In: *ACM Transactions On Graphics (TOG)* 37.4 (2018), pp. 1–14.

[41] M. L. Puterman. “Markov decision processes”. In: *Handbooks in operations research and management science* 2 (1990), pp. 331–434.

[42] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[43] S. Saeedvand, M. Jafari, H. S. Aghdasi, and J. Baltes. “A comprehensive survey on humanoid robot development”. In: *The Knowledge Engineering Review* 34 (2019), e20.

[44] S. Sastry. *Nonlinear systems: analysis, stability, and control*. Vol. 10. Springer Science & Business Media, 2013.

[45] W. Schwarting, J. Alonso-Mora, and D. Rus. "Planning and decision-making for autonomous vehicles". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 187–210.

[46] T. Shinbrot, C. Grebogi, J. Wisdom, and J. A. Yorke. "Chaos in a double pendulum". In: *American Journal of Physics* 60.6 (1992), pp. 491–499.

[47] L. Smith, I. Kostrikov, and S. Levine. "A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning". In: *arXiv preprint arXiv:2208.07860* (2022).

[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[49] R. A. Struble. *Nonlinear differential equations*. Courier Dover Publications, 2018.

[50] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[51] R. Tedrake. "Underactuated robotics". In: *Algorithms for Walking, Running, Swimming, Flying, and Manipulation* (2022).

[52] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. "LQR-trees: Feedback motion planning via sums-of-squares verification". In: *The International Journal of Robotics Research* 29.8 (2010), pp. 1038–1052.

[53] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[54] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, et al. *Gymnasium*. Mar. 2023. DOI: 10.5281/zenodo.8127026. URL: https://zenodo.org/record/8127025 (visited on 07/08/2023).

[55] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, et al. "Benchmarking model-based reinforcement learning". In: *arXiv preprint arXiv:1907.02057* (2019).

[56] F. Wiebe, S. Kumar, L. Shala, S. Vyas, M. Javadi, and F. Kirchner. "An Open Source Dual Purpose Acrobot and Pendubot Platform for Benchmarking Control Algorithms for Underactuated Robotics". In: *IEEE Robotics and Automation Magazine* (2023). under review.

[57] X. Xin and M. Kaneda. "New analytical results of the energy based swinging up control of the Acrobot". In: *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*. Vol. 1. IEEE. 2004, pp. 704–709.

[58] M. Yamakita, M. Iwashiro, Y. Sugahara, and K. Furuta. “Robust swing up control of double pendulum”. In: *Proceedings of 1995 American Control Conference-ACC’95*. Vol. 1. IEEE. 1995, pp. 290–295.

[59] Y. Zheng, S. Luo, and Z. Lv. “Control double inverted pendulum by reinforcement learning with double cmac network”. In: *18th International Conference on Pattern Recognition (ICPR’06)*. Vol. 4. IEEE. 2006, pp. 639–642.