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

Declaration

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

Garching, 15.11.2023

Chi Zhang

Project Definition

Abstract

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

Background

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

Acknowledgement

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

Project Note

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

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

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

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

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

Garching, 15.11.2023

_____
Chi Zhang

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

Contents

1 Introduction

Nonlinear systems, as their name suggests, do not exhibit linear relationships between inputs and outputs. This inherent non-linearity means the system's response to changes in input can be complex and often unpredictable. In the real world, nearly all systems display some degree of nonlinearity. This nonlinearity can manifest in a variety of phenomena. For example, in systems with multiple inputs and outputs, the interdependencies between variables can become intricate, leading to coupling issues. Another frequent challenge is chaotic behavior, where even minor alterations in initial conditions can lead to vastly different outcomes. Addressing these nonlinearities is crucial when managing control tasks, especially in real-world systems.

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. This necessitates the use of approximations and numerical methods. Efficiently and accurately executing these methods presents a central challenge in nonlinear control. Throughout history, control engineers have devised a wide range of strategies to manage these 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 under partial human oversight. These 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 power of artificial intelligence, numerous innovative control approaches are emerging each year. Many products in modern robotics have achieved remarkable success, some well-known instances include quadruped robotics[10], autonomous vehicles[38], quadcopters[32], and humanoid robots[37].

(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 dependable nonlinear control strategies to enhance motion

capabilities. Typically, the procedure for planning intricate movements within these systems adheres to a two-phase method[8], encompassing trajectory planning and trajectory tracking. This structured approach ensures precision and stability in robotic system control.

Trajectory planning[17][18] 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 is trajectory optimization[7], which seeks to minimize a cost function accounting for travel time, energy consumption, and motion smoothness. Techniques such as gradient descent or genetic algorithms are often utilized for optimization, with adherence to constraints like maximum velocities and accelerations being a crucial aspect of the process.

Upon successful trajectory planning, the next step is trajectory tracking, which entails 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.

In industrial robot control[19], the practice typically involves dividing the control process into distinct phases of offline planning and execution. This segmentation is primarily due to the non-critical requirement for real-time responsiveness in such applications. Take, for instance, the planning phase, where an algorithm such as A* is employed to determine an optimal route based on a predefined task, navigating around obstacles and minimizing travel distance, which results in a set of discrete waypoints[14]. Following this, cubic interpolation techniques are applied to these waypoints, crafting a smooth and continuous trajectory that the robot can realistically follow, ensuring both fluid motion and compliance with the robot's kinematic constraints[9]. The process subsequently moves to the execution phase, where a robust and precise control strategy is implemented. This strategy is crucial as it guarantees the robot's meticulous adherence to the pre-established trajectory, maintaining both accuracy and reliability throughout the operation[33].

Yet, in dynamic domains like automotive and flight control, the demand for real-time responsiveness takes center stage. For example, Model Predictive Control (MPC)[27] stands as a prime illustration of this critical requirement, seamlessly integrating online trajectory planning and execution into a unified framework. It initiates the process by forecasting a series of future control actions as part of the planning phase. Subsequently, during execution, these control inputs are refined to minimize any discrepancies between the real-time system state and the intended trajectory, ensuring precise alignment. MPC excels in its ability to simultaneously create, optimize, and follow trajectories, all while dynamically adjusting in real-time to accom-

modate disturbances and uncertainties. This capability is vital for maintaining precise control and adaptability in fast-paced and complex environments.

The traditional trajectory generation 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 unforeseen changes or dynamic environments. If the environment changes significantly, 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.

Recognizing the limitations inherent in traditional trajectory generation and tracking methodologies is essential for fostering the development of more advanced and efficient strategies in trajectory planning and control. This insight becomes particularly crucial in contexts that require rapid responsiveness and a capacity to skillfully navigate unpredictable changes or dynamic environments.

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) is a subset of machine learning centered on agents learning optimal behavior through trial-and-error interactions with their environment. Essentially, the 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 and adjust its policy accordingly to maximize cumulative rewards over time.

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

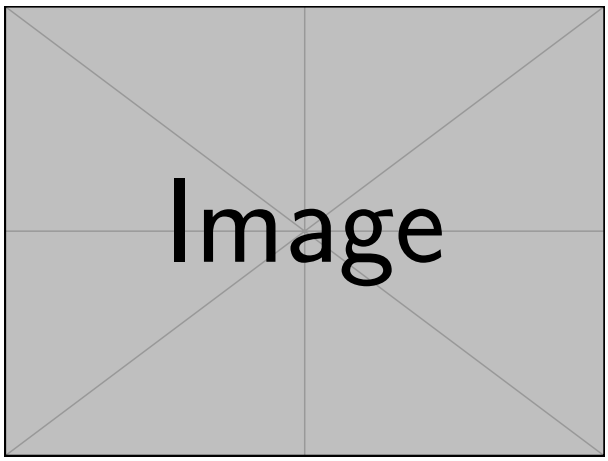


Figure 1.2: Markov chain

The ultimate objective in RL is to find an optimal policy, a mapping from states to actions, that dictates the best action to take in each state to achieve maximum long-term rewards. The learning process is driven by the tuple (s, a, r, s'), representing the current state, action taken, received reward, and subsequent state. Over time, through exploration of the state-action space and exploitation of acquired knowledge, the agent refines its policy, converging towards optimal behavior.

A key challenge in RL is balancing exploration and exploitation. Exploration involves trying out different actions to discover their potential outcomes, essential for acquiring new knowledge. Exploitation, on the other hand, entails leveraging existing knowledge to make optimal decisions. Striking the right balance between these two strategies is crucial for the agent to effectively navigate its environment and learn from received rewards.

In the field of robotic control, the integration of reinforcement learning has become increasingly popular due to the numerous distinct advantages this combined 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 thrives 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 intricate 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 spaces and control intricate relationships between states and actions in complex scenarios.

Reinforcement learning distinguishes itself as a straightforward control method with high adaptability, setting itself apart from traditional trajectory generation 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 accurately predefined 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[31] 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 s 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-related tasks, 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:

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

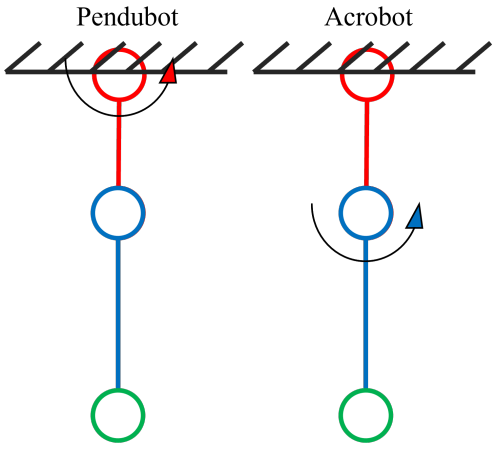
- **Case 2:**

  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[43].

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[39]. 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[50], and it continues to be a crucial testbed for validating the effectiveness of newly designed control algorithms[49][51][4].

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.

1.3 Contribution

In this paper, the main contribution is as follows: An effective control strategy has been developed to achieve two key objectives with the double pendulum. The first task involves swinging the double pendulum from its lowest point to its highest point. The second task is to ensure stability at the highest point for an extended time period.

To address the swing-up task, a well-known model-free reinforcement learning algorithm called Soft Actor-Critic (SAC) was utilized[21]. 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[29]. Once the RoA is reached by the system, a seamless transition to the LQR controller is made to maintain stability around the highest point.

1.4 Content

This thesis comprises six chapters, namely: introduction, state of the art, methodology, agent training and experiments in a simulation environment, experiments on real hardware, discussion, and future work. At the end of Chapter One, the content of the following chapters is outlined below.

- **Chapter 2: State of the Art**:
  - This chapter begins by providing an explanation of the fundamental theories related to double pendulum dynamics. It is followed by an overview of recent advancements in the field of robotic control, with a specific focus on learning-based control methods.
- **Chapter 3: Methodology**:
  - This chapter delves into the methodology, covering fundamental aspects of reinforcement learning, with a specific focus on the SAC algorithm. It explains the reward function used for training, the training procedure, and introduces the concept of the LQR controller and the composition of the combined controller.

  - Additionally, we introduce the evaluation metrics used to assess the performance and robustness of the newly designed control strategies in both simulated environments and real-world experiments.

- **Chapter 4: Agent training and experiments in simulation environment**:

  - In this chapter, we present the results obtained from simulations, showcasing the performance and behavior of the designed control strategy.

- **Chapter 5: Experiments on real hardware**:

  - This chapter reports the outcomes of experiments conducted on the hardware, providing insights into our approach to addressing the sim2real transfer problem.

- **Chapter 6: Discussion and Future Work**:

  - The final chapter engages in a discussion about the obtained results and explores potential future research and development directions.

2 State of the art

In this chapter, we will explore the state of the art. In the first section, we will first discuss the important variations of double pendulum that is widely used by the research society. Subsequently, we will delve into the basic dynamics of a double pendulum system. In the second section, we will review some of the most renowned works related to learning-based control in the field of robotics.

2.1 Theory

In this section, we will first introduce the various variations of double pendulum setups commonly used in research society. Subsequently, we will explore the fundamental principles governing double pendulum dynamics.

2.1.1 Variations of double pendulum

Broadly speaking, a double pendulum system is a mechanical structure consisting of two pendulum arms or masses suspended in such a way that they can swing freely and independently from each other. These pendulum arms are typically connected in a series, where the motion of the second pendulum is influenced by the motion of the first pendulum.

There are several variations of the double pendulum setup, and one of the most famous ones is the Double Inverted Pendulum on a Cart (DIPC)[11]. The DIPC system is a modification of the pendubot setup, as it is actuated solely at the shoulder joint. The key distinction lies in the actuation mechanism, which employs a prismatic joint instead of a revolute joint.

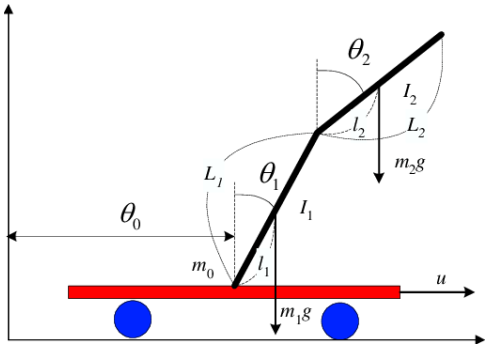


Figure 2.1: Double inverted pendulum on a cart[11]

Another significant variation is the Furuta pendulum[13], initially developed at the Tokyo Institute of Technology by Furuta and his colleagues[16]. This system comprises a driven arm that rotates horizontally and a pendulum attached to this arm, allowing free vertical rotation. Due to the influence of gravitational forces and the coupling stemming from Coriolis and centripetal forces, the Furuta pendulum is characterized by its underactuation and highly nonlinear behavior.

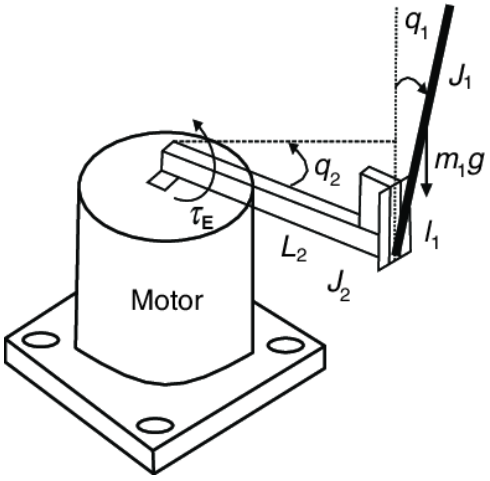


Figure 2.2: Furuta pendulum[2]

The double pendulum utilized in this thesis represents a third variation. It consists of 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 two links are

connected to the world frame via revolute joints as well. 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.

2.1.2 Dynamics of underactuated double pendulum system

As shown in Figure 2.3, we model the dynamics of the double pendulum with 15 parameters which include 8 link parameters namely masses (m_1, m_2) , lengths (l_1, l_2) , center of masses (r_1, r_2) , inertias (I_1, I_2) for the two links, and 6 actuator parameters namely motor inertia I_r , gear ratio g_r , coulomb friction (c_{f1}, c_{f2}) , viscous friction (b_1, b_2) for the two joints and gravity.

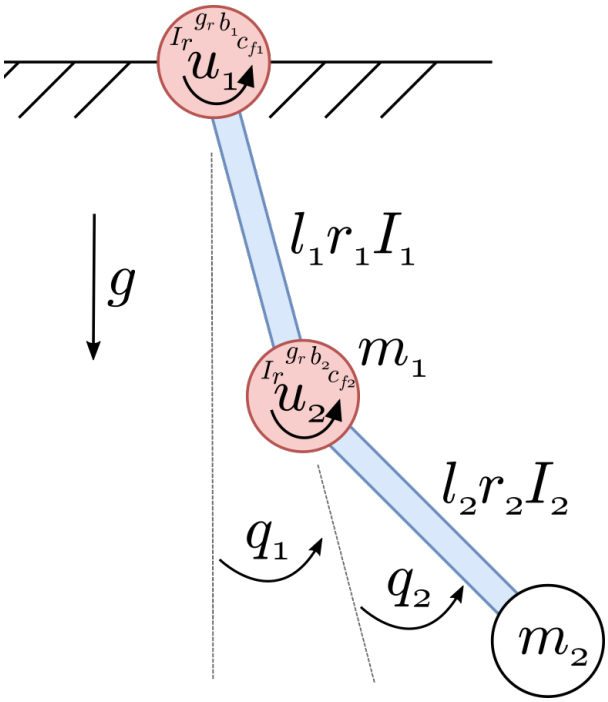


Figure 2.3: Double pendulum dynamics

The generalized coordinates $\mathbf{q} = [q_1, q_2]^T$ are the joint angles measured from the free hanging position. The state vector of the systems contains the position coordinates and their time

derivatives: $\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T$. The torque applied by the actuators are $\mathbf{u} = [u_1, u_2]$. The equation of motion for the dynamics of a dynamical system can be derived following the blow steps:

Step 1. Define the Lagrangian (L):

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

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

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:

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

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

Step 5. Solve the Equations of Motion:

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

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} = Du + G(q) - F(\dot{q}) \tag{2.5}$$

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

$$\begin{aligned} \dot{x} &= f(x,u) \\ &= \begin{bmatrix} \dot{q} \\ M^{-1}(Du - C(q,\dot{q})\dot{q} + G(q) - F(\dot{q})) \end{bmatrix} \end{aligned} \tag{2.6}$$

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

$$\begin{cases} x_1 = l_1 \sin(q_1) \\ y_1 = -l_1 \cos(q_1) \end{cases} \tag{2.7}$$

$$\begin{cases} x_2 = l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) \\ y_2 = -l_1 \cos(q_1) - l_2 \cos(q_1 + q_2) \end{cases} \tag{2.8}$$

Put Equation 2.7 and 2.8 into 2.2, 2.3, 2.4, 2.5,we can get the mass matrix (with $s_1 = \sin(q_1), c_1 = \cos(q_1), \ldots$)

$$\mathbf{M} = \begin{bmatrix} I_1 + I_2 + l_1^2 m_2 + 2 l_1 m_2 r_2 c_2 + g_r^2 I_r + I_r & I_2 + l_1 m_2 r_2 c_2 - g_r I_r \\ I_2 + l_1 m_2 r_2 c_2 - g_r I_r & I_2 + g_r^2 I_r \end{bmatrix} \tag{2.9}$$

The Coriolis matrix:

$$\mathbf{C} = \begin{bmatrix} -2\dot{q}_2 l_1 m_2 r_2 \sin(q_2) & -\dot{q}_2 l_1 m_2 r_2 \sin(q_2) \\ \dot{q}_1 l_1 m_2 r_2 \sin(q_2) & 0 \end{bmatrix}, \tag{2.10}$$

The gravity vector:

$$\mathbf{G} = \begin{bmatrix} -g m_1 r_1 \sin(q_1) - g m_2 \left(l_1 \sin(q_1) + r_2 \sin(q_{1+2}) \right) \\ -g m_2 r_2 \sin(q_{1+2}) \end{bmatrix}, \tag{2.11}$$

The friction vector:

$$\mathbf{F} = \begin{bmatrix} b_1 \dot{q}_1 + c_{f1} \arctan(100\,\dot{q}_1) \\ b_2 \dot{q}_2 + c_{f2} \arctan(100\,\dot{q}_2) \end{bmatrix} \tag{2.12}$$

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

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

$$\mathbf{D}_{full} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{D}_{pendu} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathbf{D}_{acro} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.13}$$

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[35], 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.

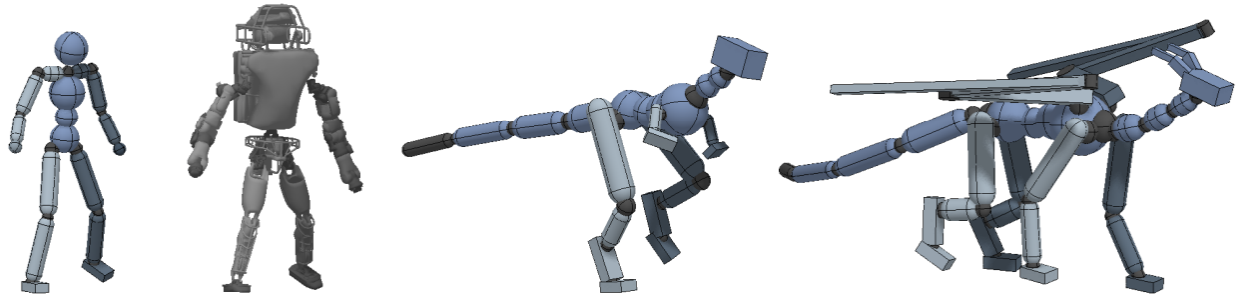


Figure 2.4: 3D characters in Deepmimic project[35]

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[40] conducted by researchers from UC Berkeley. They employ an algorithmic framework closely related to DroQ [22], an extension of the SAC algorithm [21] incorporating dropout [41] and layer normalization [6]. 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. [28], 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[47], provides a comprehensive comparison by benchmarking model-based reinforcement learning.

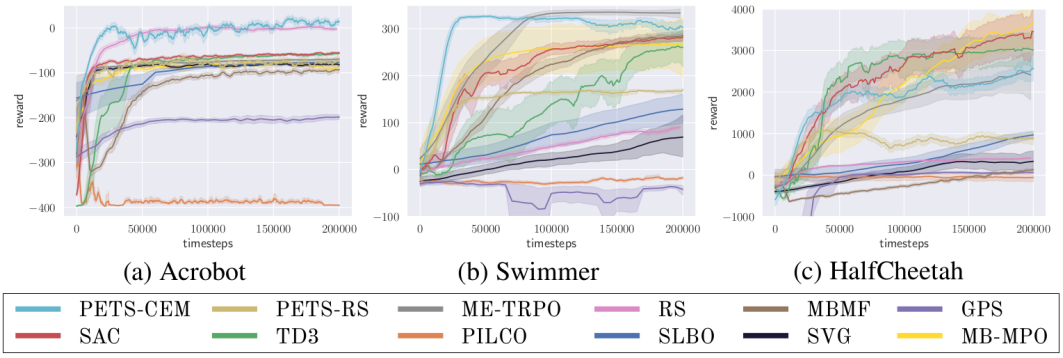


Figure 2.5: Benchmarking model-based reinforcement learning[47]

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[12], 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 [23]. 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.

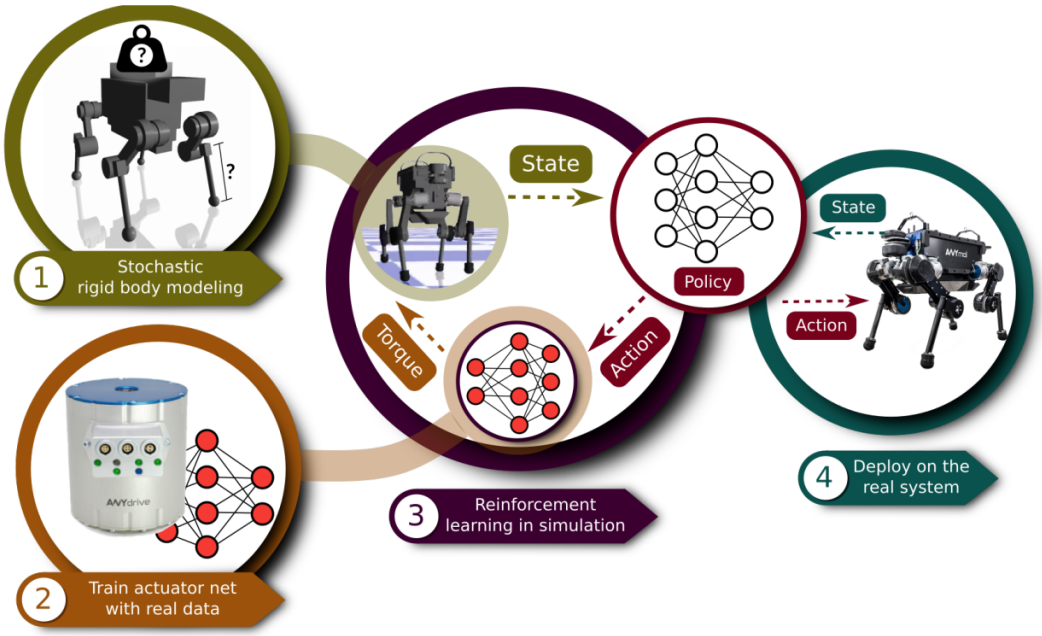


Figure 2.6: Sim-to-real framework for RL-based quadruped controller[23]

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.

3.1 Soft actor critic

Within the landscape of reinforcement learning, the Soft Actor Critic (SAC)[21] 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[26].

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 using:

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

The policy is updated using one step of gradient ascent:

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

where $\widetilde{a}_\theta(s)$ is a sample of action derived from $\pi_\theta(\cdot|s)$ which is differentiable with respect to θ. The actor and critic neural networks' parameters are updated, resulting in policy adaptation.

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)[29] 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 $x_{op} = [\pi, 0, 0, 0]^T$. 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 matrices Q and R must be chosen to be symmetric and positive definite, which means $Q = Q^T$ with all its eigenvalues positive, and similarly, $R = R^T$ with all positive eigenvalues.

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

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

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.[20], 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 aligns with our task.

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[34].

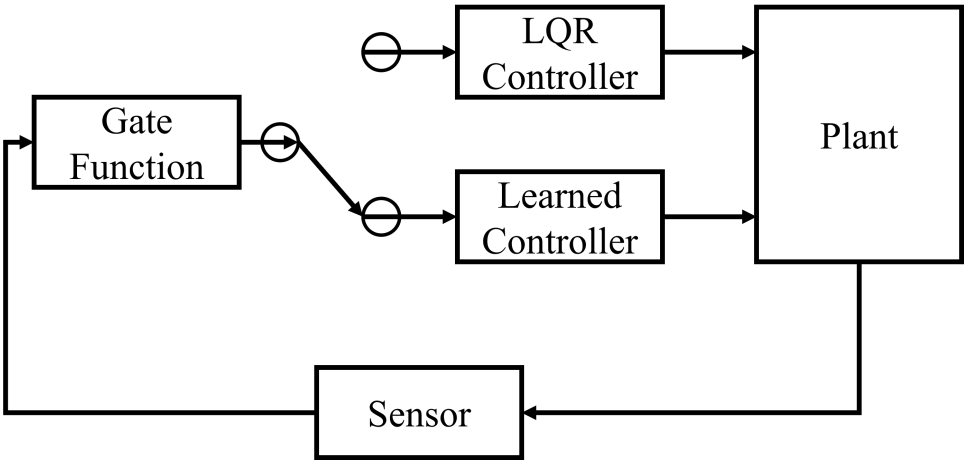


Figure 3.1: Combined controller

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[25].

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

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[34] but with a sums of squares method[44]. The resulting shape of the RoA in a 4D state space resembles an ellipsoid.

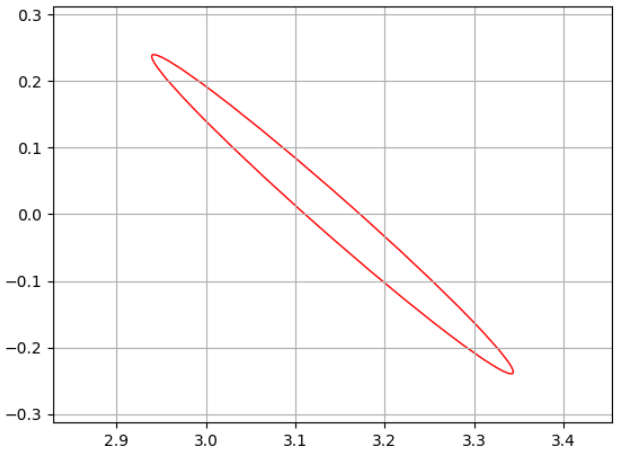


Figure 3.2: Region of Attraction

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[46] 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. The manifestation of failure is entirely random.

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[46]. 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 and reliable 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.

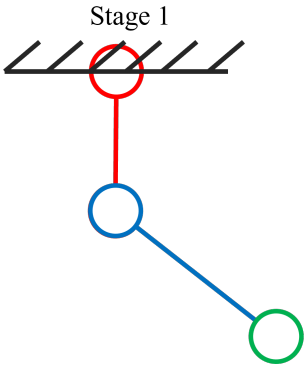


Figure 3.3: Swing up stage 1

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.

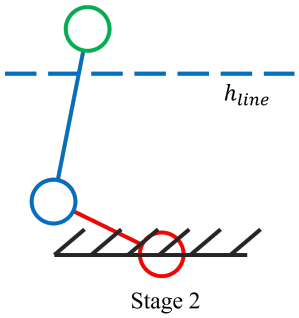


Figure 3.4: Swing up stage 2

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.

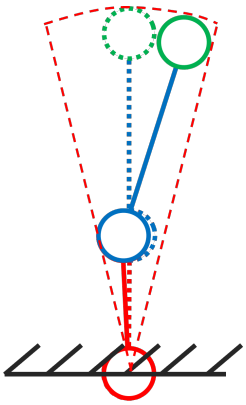


Figure 3.5: Swing up stage 3

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[48]. 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, we also consider robustness metrics. 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 environment

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

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

4.1 Ideal training setup

Stable Baseline 3(SB3)[36] 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[24]. 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[46] 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.

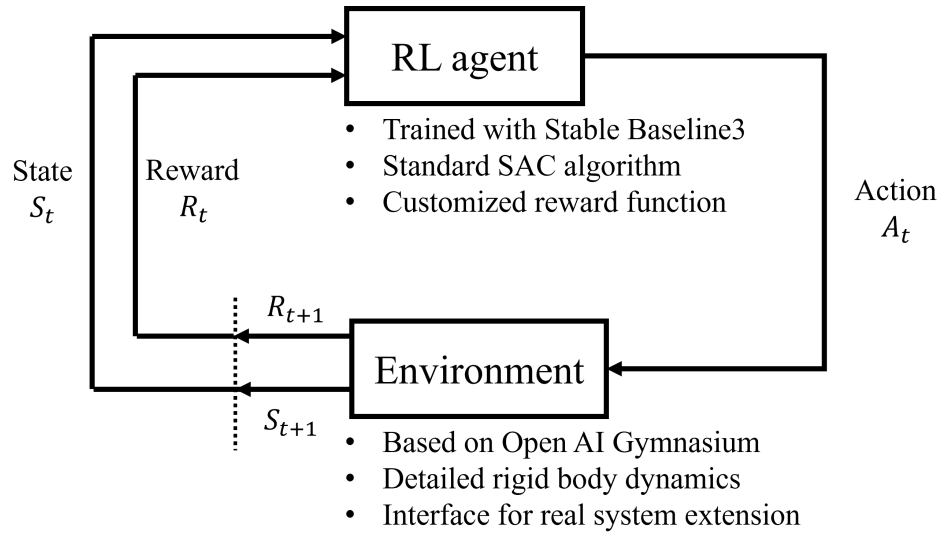


Figure 4.1: Interaction between reinforcement learning agent and customized environment

To construct the customized environment, we begin with the dynamic function that captures the nonlinear dynamics of the underactuated double pendulum from this respository[48]. The dynamic function uses the current state observation and the action determined by the control policy, producing the next state via a Runge-Kutta integrator.

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

It is widely recognized in the field of reinforcement learning that algorithms tend to converge more effectively when utilizing normalized state and action spaces [42]. 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.

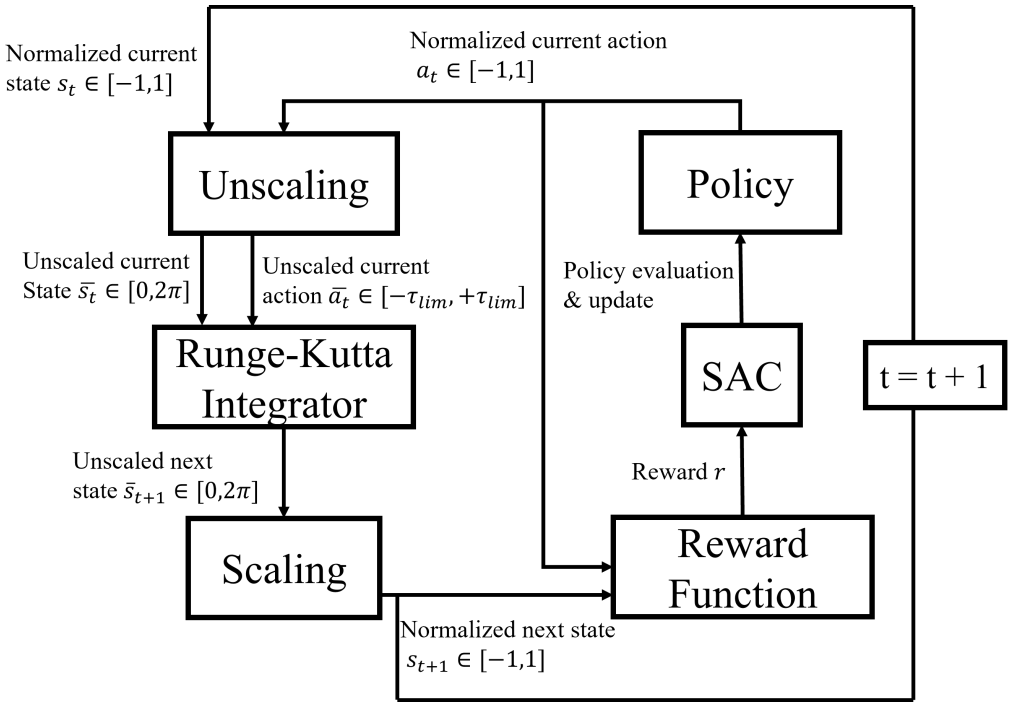


Figure 4.2: Detailed scaling pipeline in training process

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 of 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 Ideal training process

During the training of the SAC controller for both the acrobot and pendubot, the primary focus was on tuning several key hyperparameters. These encompassed the learning rate, control frequency, episode length, and learning timestep. Effort was invested in adjusting hyperparameters for the reward function and the LQR controller, given their pivotal roles in the learning process, as detailed in Table 4.2.

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

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

Table 4.2: Hyper parameters used for the SAC training and the LQR controller.[may change]

One of the successful learning curves for a total of 2e7 timesteps for the pendubot is illustrated below. This experiment, like all other simulation experiments done on the pendubot setup, uses design A, namely, $l_1 = 0.3m$ and $l_2 = 0.2m$. As depicted in the figure, from 0 to 1e7 timesteps, the learning curve experiences a gradual ascent. Between 1e7 and 1.2e7 timesteps, the learning curve surges rapidly, peaking at around one million in reward. After 1.2e6 timesteps, the curve begins to stabilize with occasional fluctuations, and no substantial increase in reward is observed.

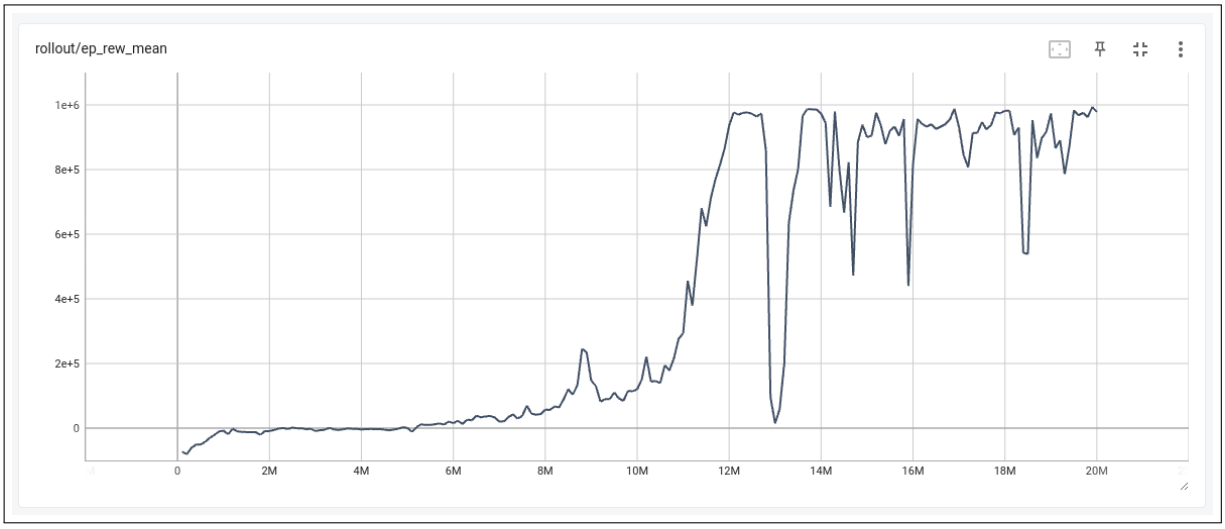


Figure 4.3: Pendubot learning curve

The training on the acrobot are based on design C, with l_1 being 0.2m and l_2 being 0.3m. As shown in the successful learning curve over a total of 3e7 timesteps for the acrobot, the curve experienced relatively steady growth until 1.8e7 timesteps. After a sudden drop in reward between 1.8e7 and 2e7 timesteps, the learning curve began to increase drastically, with two major setbacks. By 3e7 timesteps, the learning curve had not reached its peak. We extended the learning period to 5e7 using a warm start from the model obtained at 3e7 timesteps, and the learning curve began to stabilize around 4e7 time steps.

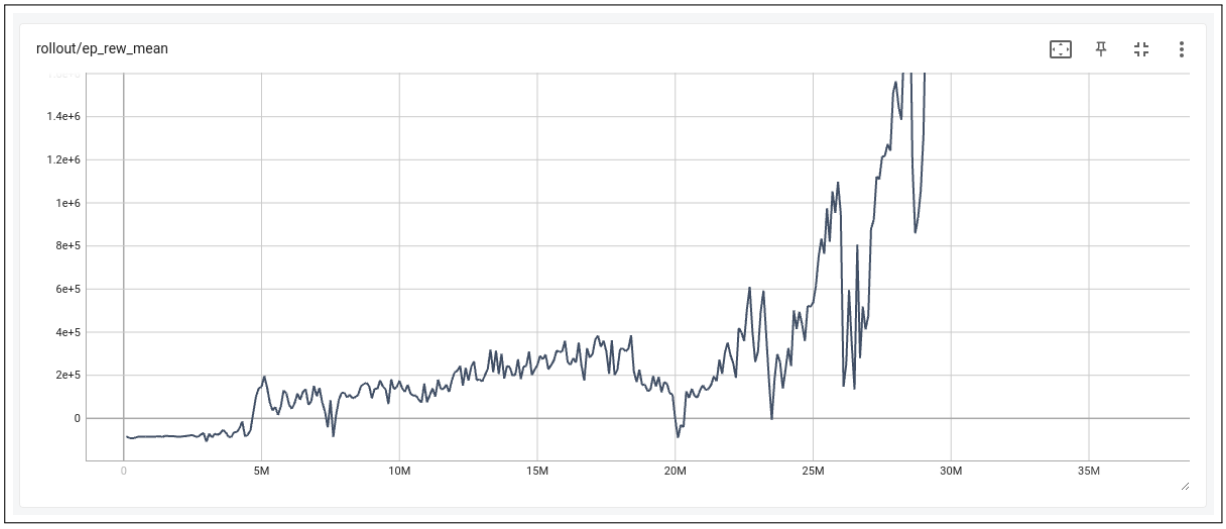


Figure 4.4: Acrobot learning curve

4.3 Ideal simulation results

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

4.3.1 Pendubot simulation in ideal environment

The testing environment, identical to the customized learning environment employed during the evaluation phase in training, has been established to validate the training outcomes and replicate the agent's learned behavior from the reinforcement learning process. This testing environment is considered ideal as it excludes disturbances such as friction and damping, focusing solely on the effects of gravity, joint torque, and mechanical constraints. In this environment, zero

initialization is applied instead of the noisy initialization. The system begins with an initial state of $[0,0,0,0]^T$, representing its lowest point with zero velocity, and initiates the swing-up using the control policy exclusively derived from SAC.

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

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

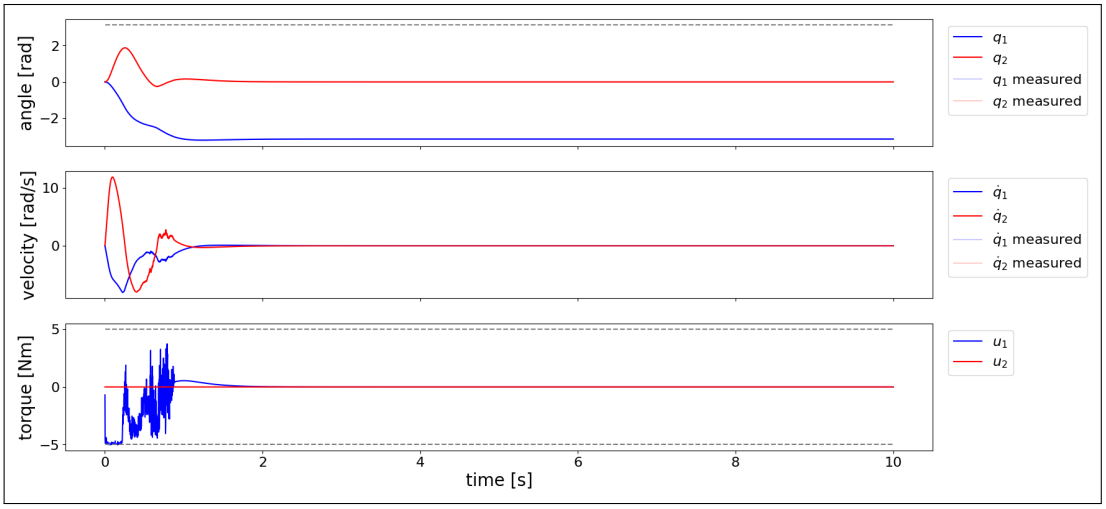


Figure 4.5: Pendubot simulation result

4.3.2 Acrobot simulation in ideal environment

Just like the pendubot simulation, experiments on the acrobot setup are conducted in an ideal environment identical to the evaluation environment during the training phase. The acrobot begins its swing from the downward position with zero velocity, with the final goal of stabilizing around its highest point.

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

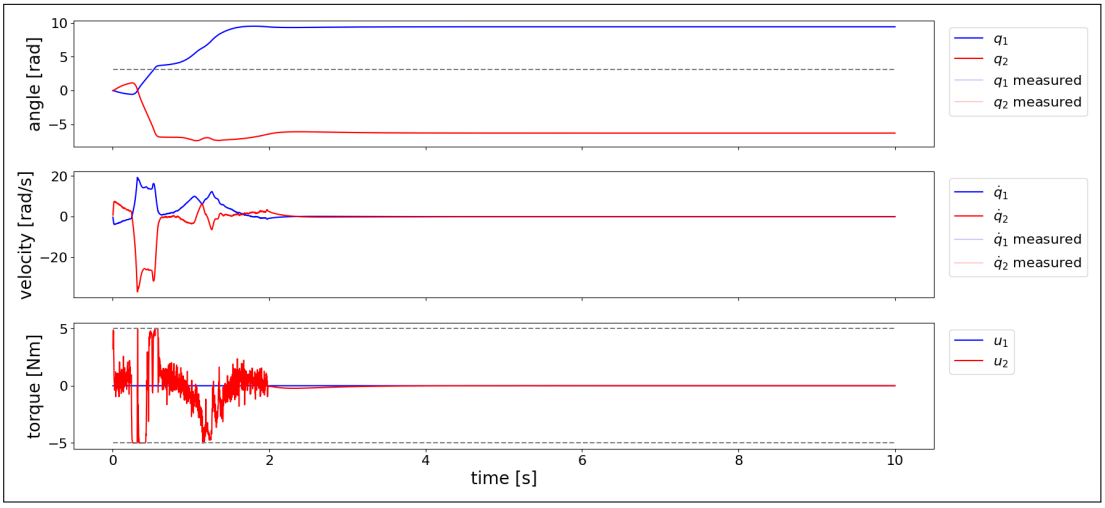


Figure 4.6: Acrobot simulation result

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

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 sufficient time steps (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 the figure below and discussed in section 4.3.1, the stabilization by the RL-learned policy, while using the same agent and model parameters for demonstration, is less smooth compared to the LQR-based stabilizer, with the torque fluctuating at an amplitude of approximately 1.5 Nm and minimal fluctuations in position and velocity.

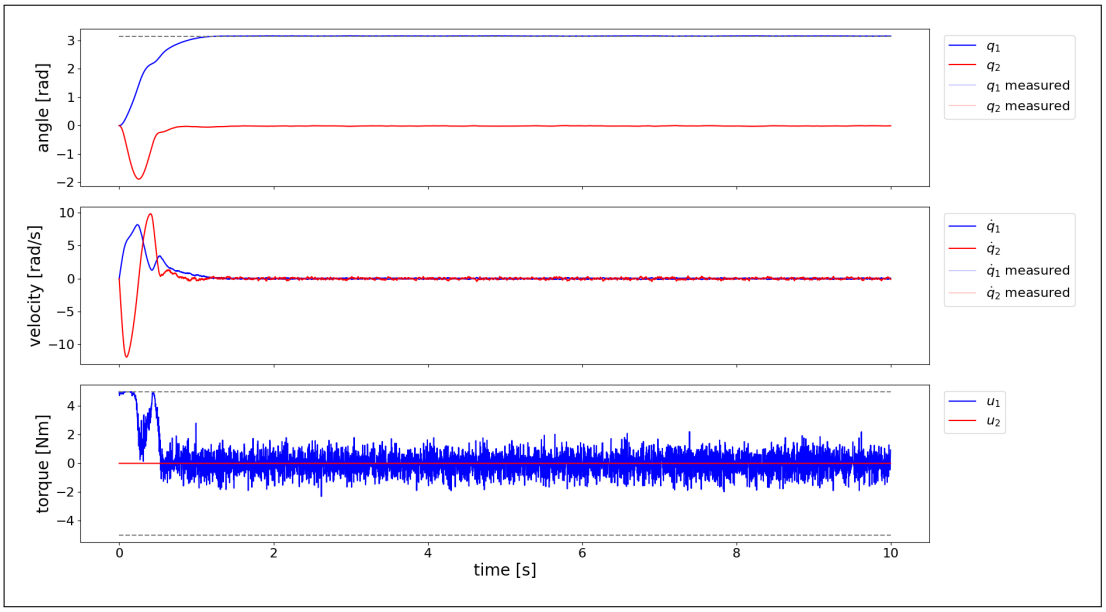


Figure 4.7: Self stabilization of pendubot in simulation

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 the figure that follows. 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. Although the stabilization provided by the RL-based controller is less smooth than that of the LQR-based controller, the inclination towards self-stabilization is distinctly apparent.

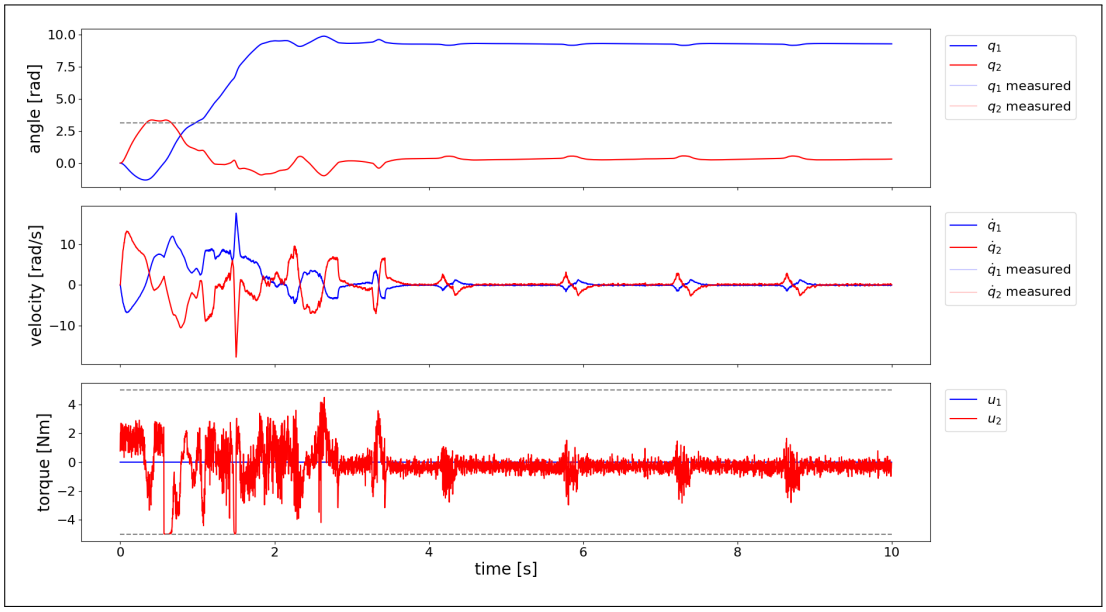


Figure 4.8: Self stabilization of acrobot in simulation

5 Experiments on real hardware

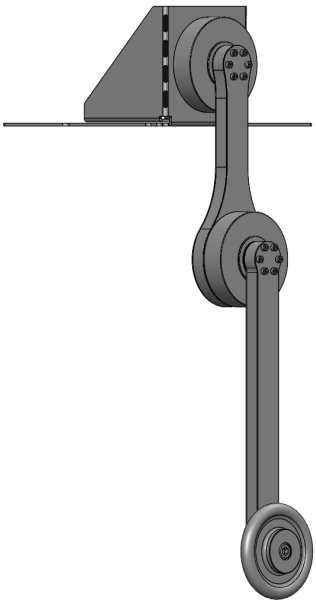
In this chapter, we discuss experiments conducted on the hardware system. The content is organized into four sections. The first section provides an overview of the hardware setup for the double pendulum system. The second section delves into the system identification of the hardware. The third section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

5.1 Hardware setup

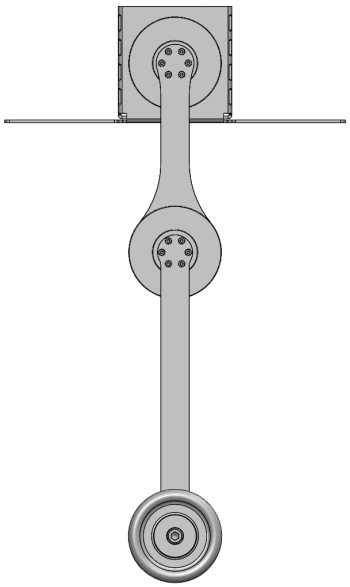
To be clear upfront, the hardware design and manufacturing were completed by previous researchers and coworkers at the Underactuated Lab of the German Research Center for Artificial Intelligence GmbH, Robotics Innovation Center branch. My role in this section is to understand the logic of all the involved subsystems and set up the test environment using 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.

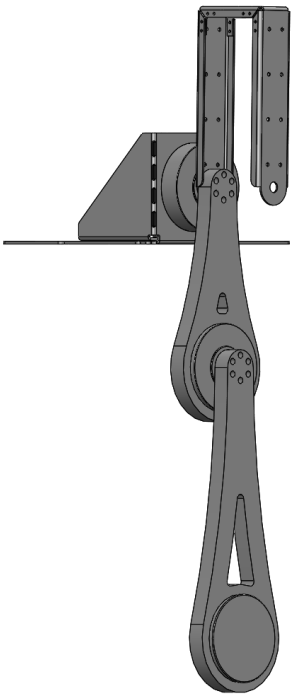
The mechanical design of the double pendulum underwent two iterations. The initial design was characterized by rotational imbalances due to a slightly misaligned motor axis and homogeneous link sizes, leading to vibrations and potential failures. Significant improvements were made in the second iteration: 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.



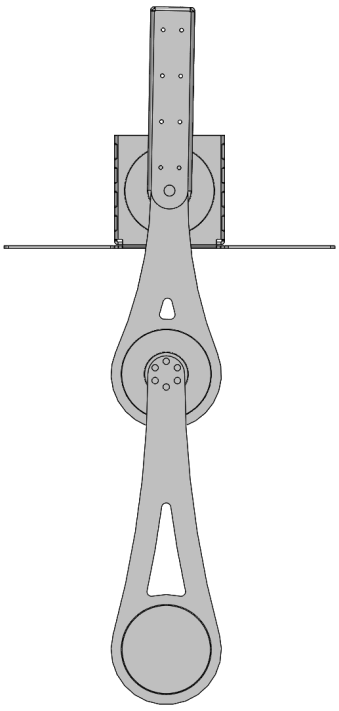
(a) Iteration 1 in isometric view



(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[3]

The AK80-6 V100 motors from CubeMars were selected for their ease of mounting, which is facilitated from both the front and rear ends[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.

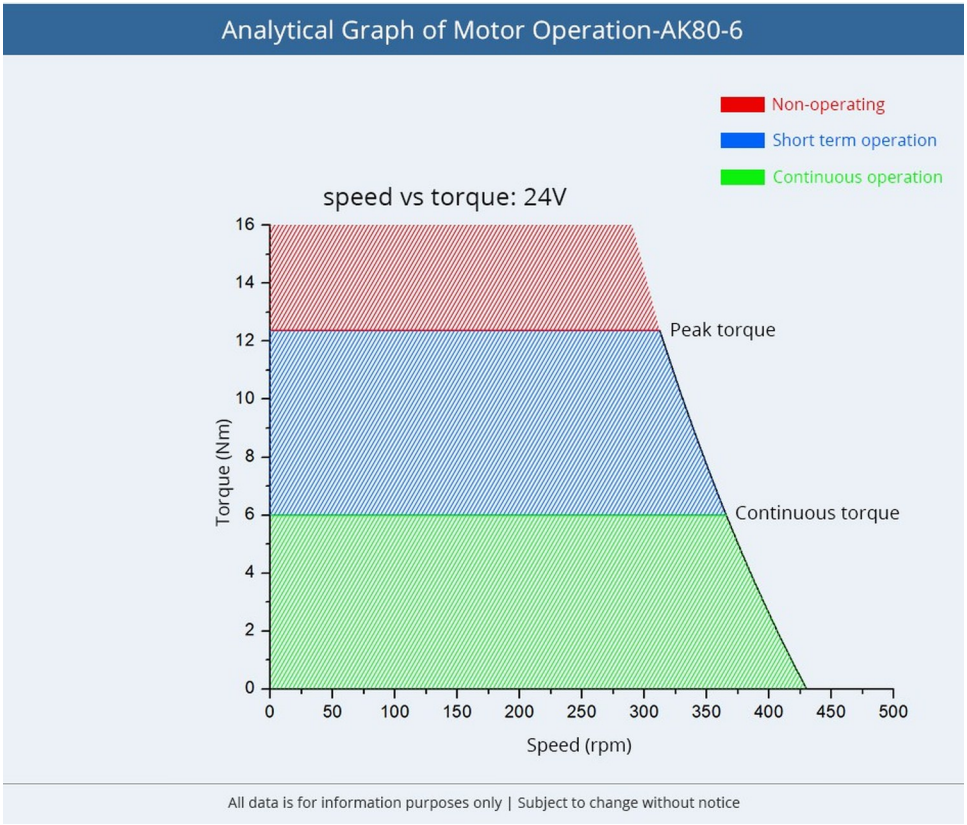


Figure 5.3: Torque speed curve of AK80-6 V100 motor[3]

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 the proposed configuration, 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.

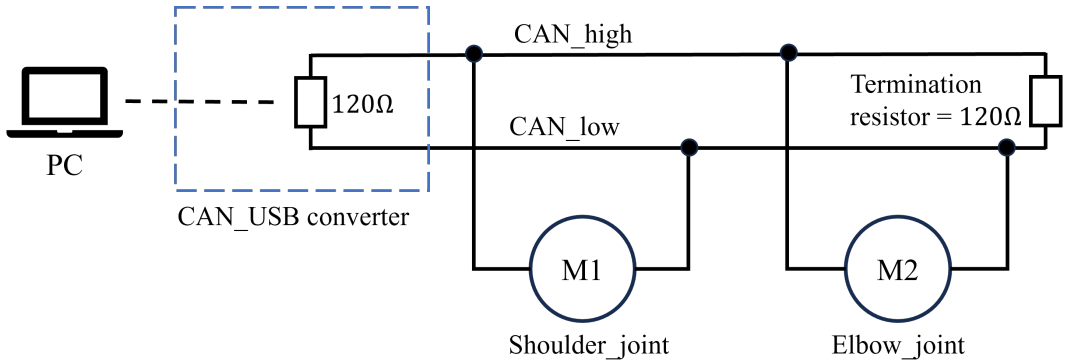


Figure 5.4: CAN connection diagram

To achieve a higher control frequency of approximately 500 Hz, the CAN-USB/2 product from ESD GmbH Hannover [15] 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[15]

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:

At the software level, speed and position limits have been defined. Due to the potential for vibrations and rotational imbalances that may occur at high speeds, which could lead to structural disassembly, a maximum speed of 20 rad/s has been instituted. Should the speed surpass this threshold, an automatic system halt will be initiated, analogous to the actuation of the emergency stop mechanism. Position limits have been established at 2π for both joints. Excessive rotations risk the entanglement of CAN and power cables, which could result in interference and possible cable damage. A schematic of the entire wiring system is illustrated in the 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 apparatus, significantly mitigating the potential for accidents.

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

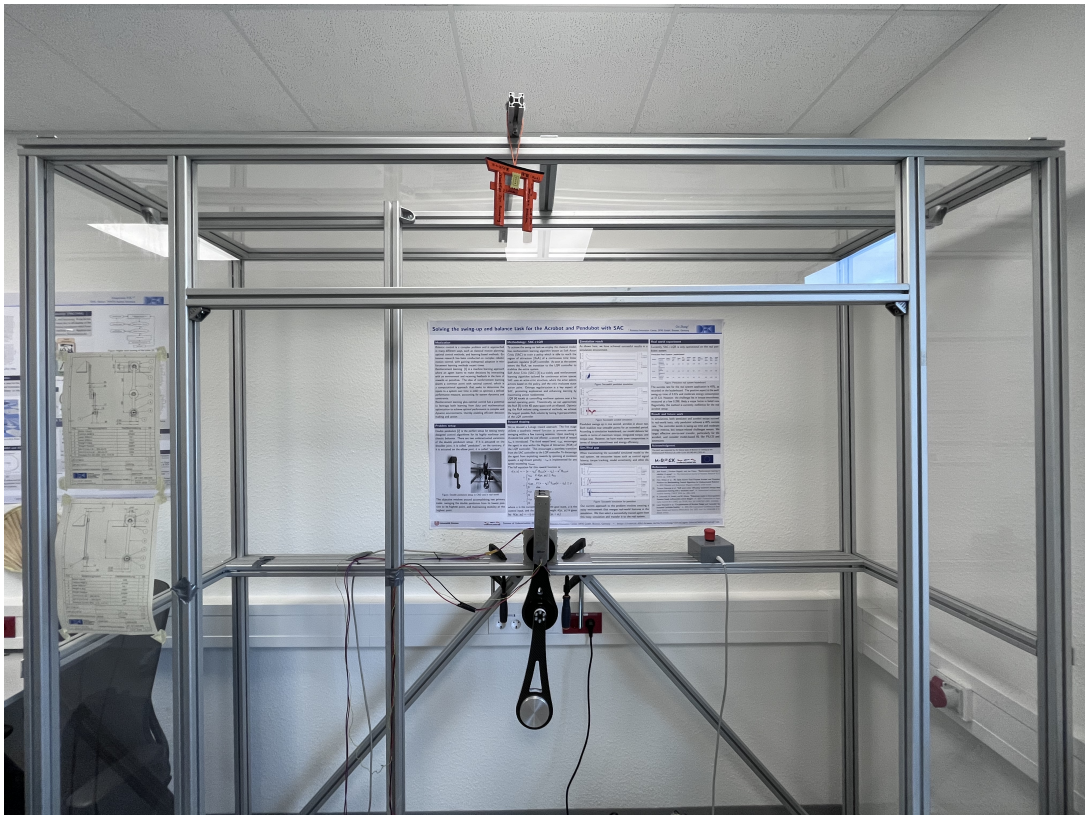


Figure 5.7: Physical enclosure for protection

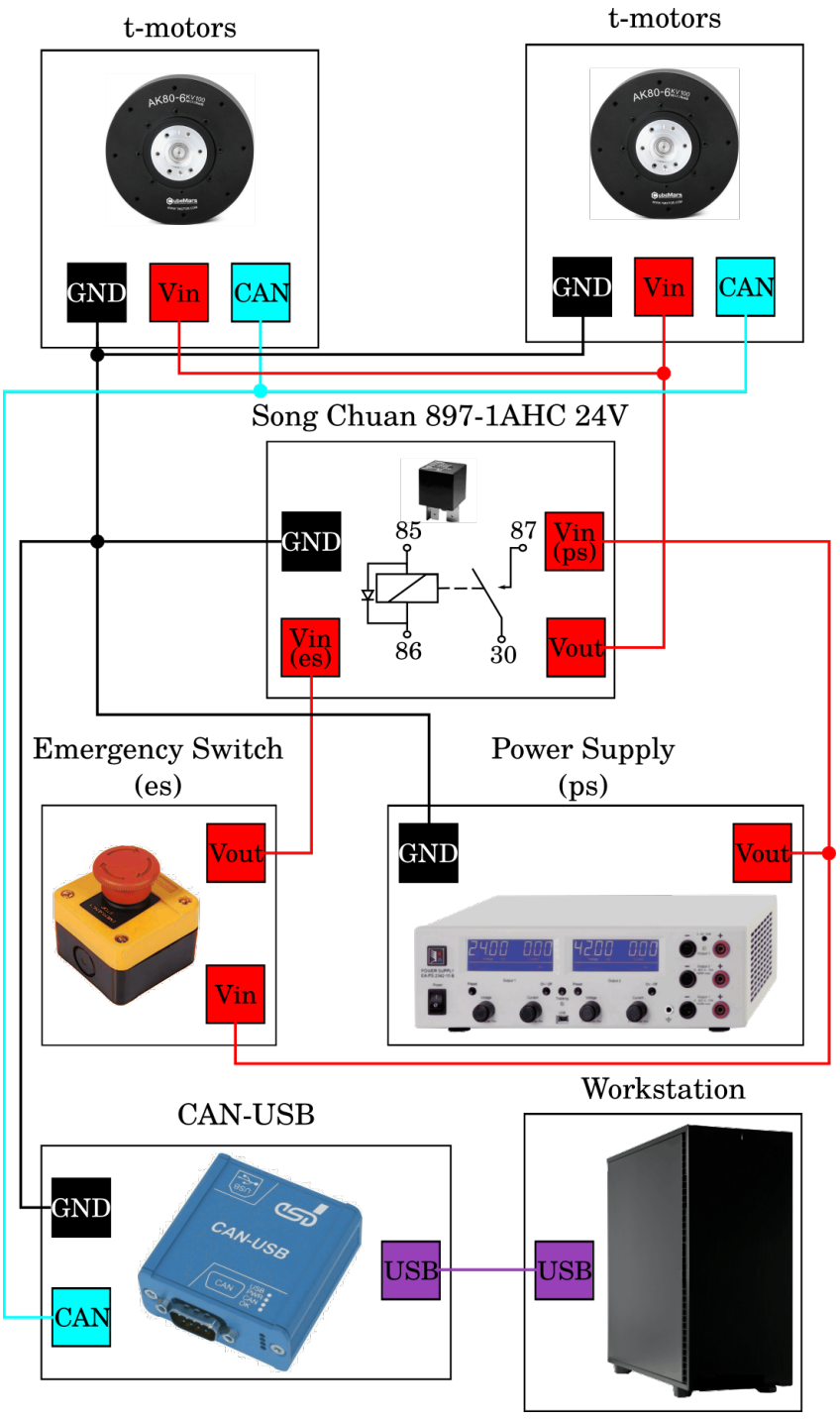


Figure 5.8: Wiring diagram[48]

5.2 System identification

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

Out of all model parameters, 15 are selected. While the naturally provided parameters g and g_r are held constant, the easily measurable parameters l_1 and l_2 are determined by the design(Table 4.1). The remaining system parameters need to be identified. , which are

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

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

The identified model parameters are shown in table below:

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 sim2real transfer, multiple agents were trained using the Soft Actor-Critic (SAC) algorithm under similar setups in ideal environment. These agents were then subjected to validation within a noisy environment. Only those agents demonstrating robustness to perturbations within this environment were advanced to testing on the actual system. Agents that did not withstand the noisy environment were deemed insufficiently robust and subsequently discarded.

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

Friction:

Friction was identified as the predominant factor. The agent training took place in a simulated environment devoid of friction, which does not reflect real-world conditions. To address this discrepancy, a strategy for friction compensation was employed, initially involving modeling based on Coulomb's law of friction.

This frictional force counteracts the relative motion between contact surfaces. Friction compensation was applied by exerting torque in the same direction as the angular motion, thereby supplying the system with the necessary energy to counteract 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 simulation 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:

$$\varepsilon \sim \mathcal{N}(\mu, \Sigma) \tag{5.1}$$

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 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:

$$u_{\text{real}} = u_{\text{previous}} + c_{tr} \cdot (u_{\text{current}} - u_{\text{previous}}) \tag{5.2}$$

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 [45], 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 Real hardware results

In this section, results from real hardware tests are presented. Initially, the procedure for selecting agents suitable for real-world testing is introduced. Subsequently, the successful outcomes from an agent trained for the pendubot setup are displayed. The range of results is restricted to the pendubot setup because the introduction of speed and position limits has significantly narrowed the range of possible policies. In the acrobot experiments, although an agent was deemed worthy of real-world testing, it exceeded the position limits and was therefore discarded.

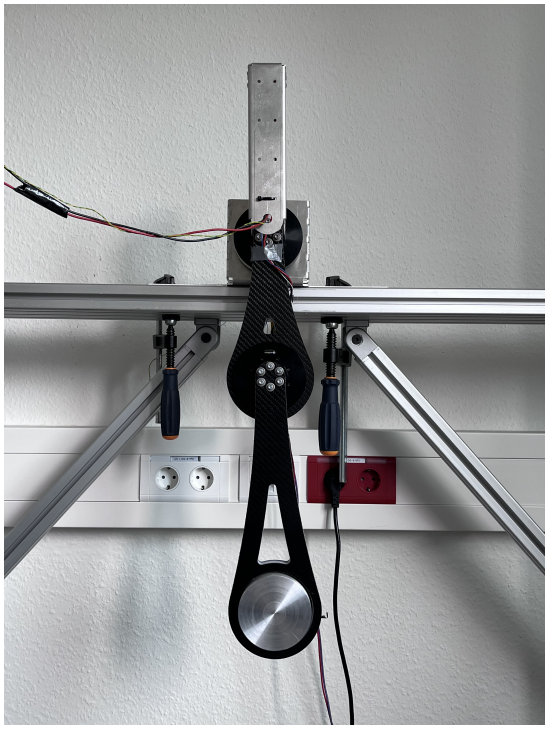


Figure 5.9: Double pendulum in real system

An agent deemed suitable for real-system tests must undergo a process that includes ideal training and validation, followed by noisy validation. 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.

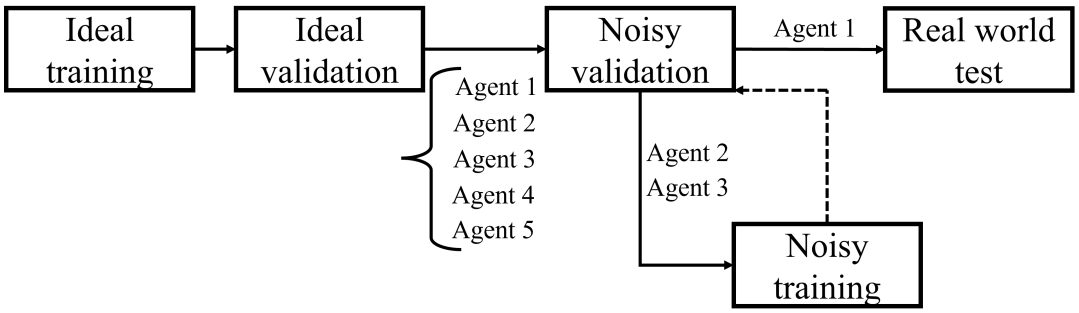


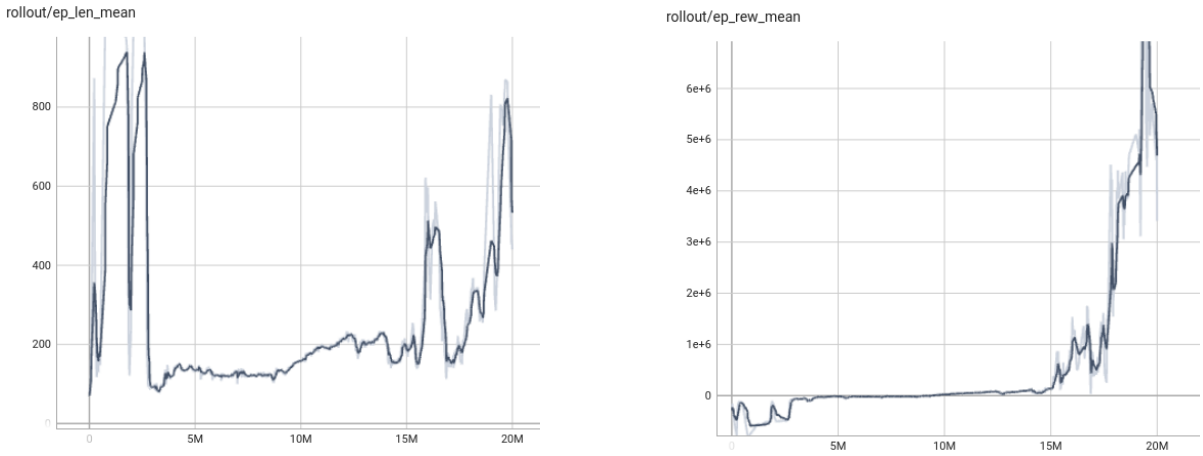
Figure 5.10: Agent selection procedure for real hardware tests

All real hardware experiments are based on design C ($l_1 = 0.2, l_2 = 0.3$).For pendubot setup, obtaining a working agent is straightforward. We did a little modification on the training process for ideal simulation phase by activating the scaling mechanism mentioned in section 4.1 and add a termination condition that terminates the training episode whenever the shoulder or elbow joint exceeds 2π. The training successfully delivered a working model in ideal simulation with 2e7 timesteps using parameters in Table 5.2.

Robot	Quadratic Reward	Constant Reward	LQR
Pendubot	Q_1 = 100	$r_{line} = 1e3$	Q_1 = 1.92
	Q_2 = 100	$r_{vel} = 0.0$	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 5.2: Hyper parameters used for training agents for real world experience

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 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 gradual emergence from this 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.



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

Figure 5.11: Training curves of the working agent on pendubot

The result in the ideal environment is shown in Figure 5.12.

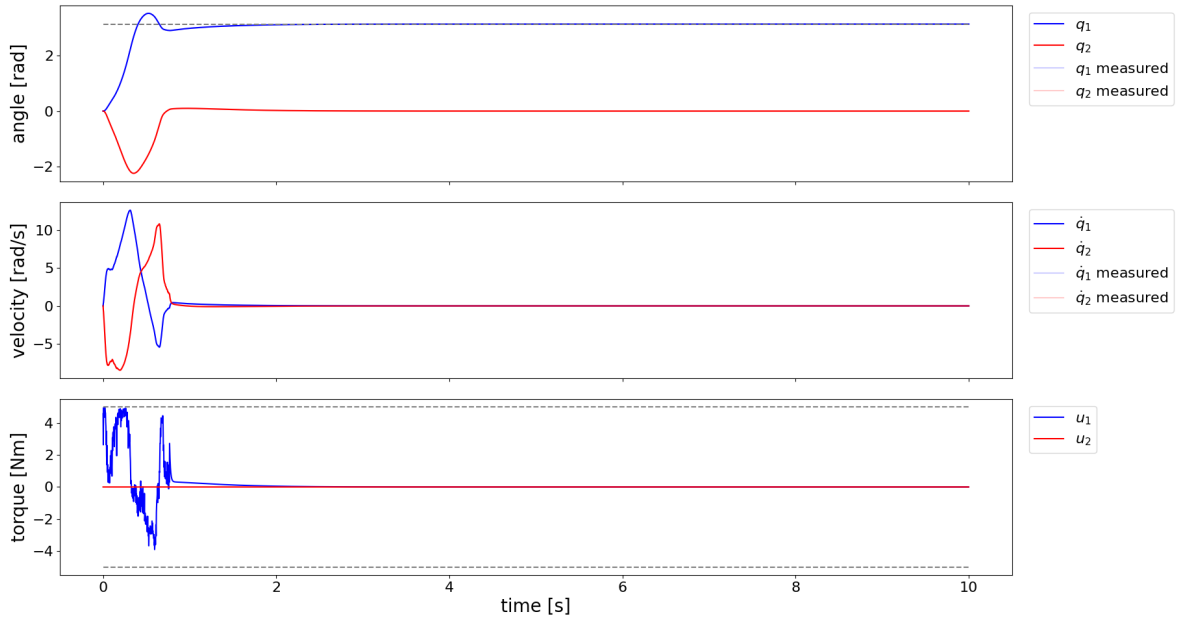


Figure 5.12: Pendubot result in the ideal environment

When tested in a noisy simulation, the pretrained agent did not perform adequately. Consequently, the agent was subjected to an additional 2e7 timesteps of training within a noisy environment. The resultant agent successfully passed the noisy validation with an approximate success rate of 40%. The result of a successful trial is shown in Figure 5.13.

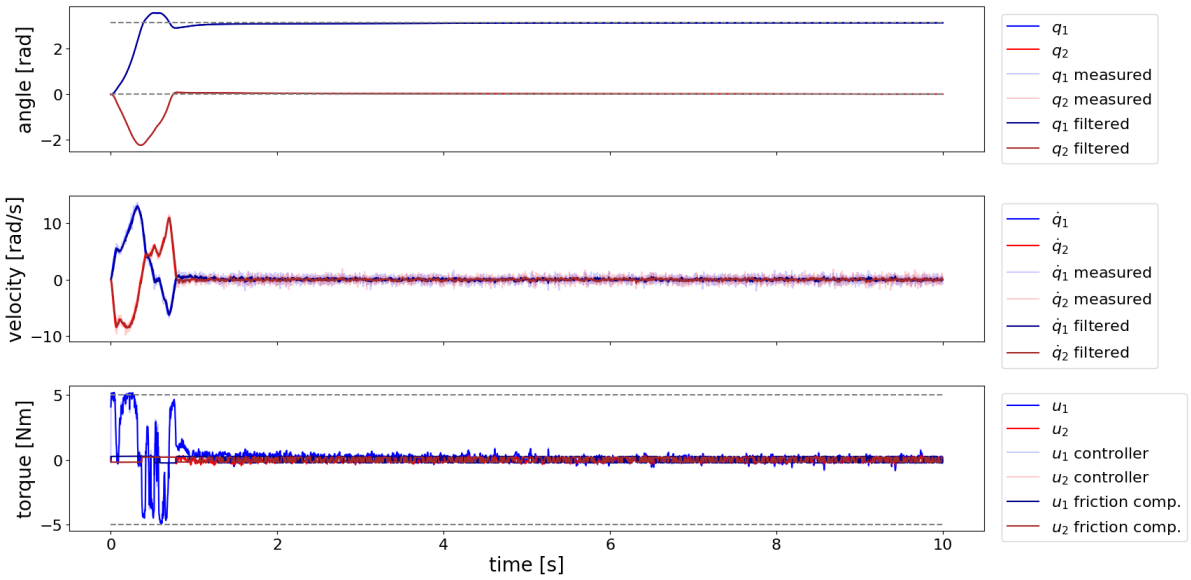


Figure 5.13: Pendubot result in the noisy environment

We consider the agent ready for a real-world test, so we deploy it onto the test bench for the final 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.

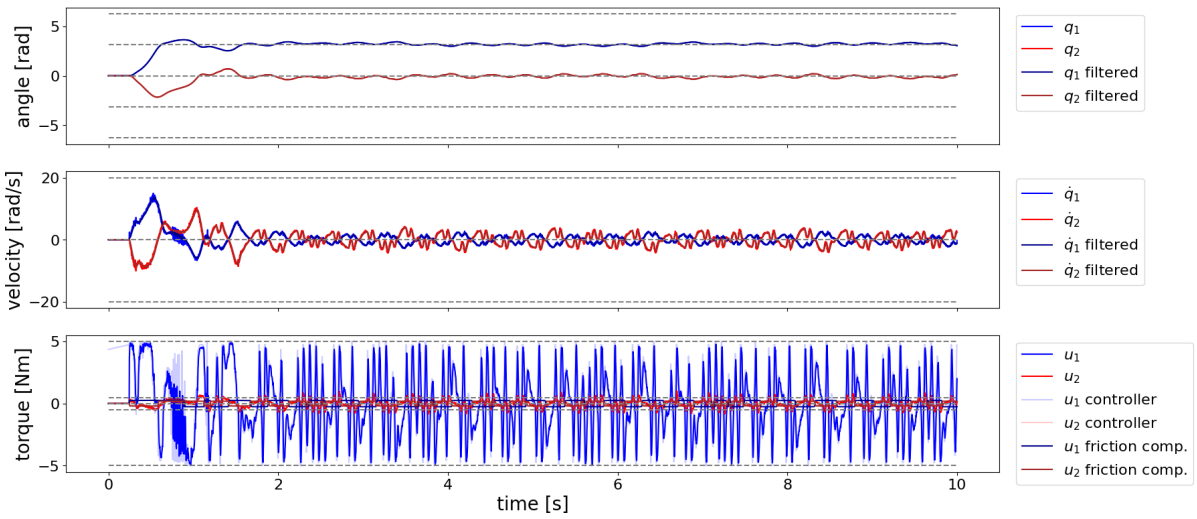


Figure 5.14: Result of a successful pendubot experiment on real system

6 Discussion and Future Work

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

6.1 Interpretation of simulation leaderboard

In the table below, the performance leaderboard results for both the pendubot and acrobot in simulation experiments are presented. Three major types of controllers are listed for comparison. The SAC+LQR controller is our design and is based on the model-free reinforcement learning method. MC-PILCO [5] [30], 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 using a remote testing system. tvLQR is an extension of the standard Linear Quadratic Regulator (LQR) control design. It is 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.

Criteria	SAC+LQR		MC-PILCO		tvLQR	
	Pendubot	Acrobot	Pendubot	Acrobot	Pendubot	Acrobot
Swingup Success	success	success	success	success	success	success
Swingup time [s]	0.65	2.06	1.43	1.1	4.2	3.98
Energy [J]	9.4	29.24	12.67	9.81	9.06	10.92
Max. Torque [Nm]	5.0	5.0	2.4	2.82	2.82	5.0
Integrated Torque [Nm]	2.21	4.57	3.48	1.27	2.57	2.27
Torque Cost [N^2m^2]	8.58	12.32	7.77	2.27	2.0	2.47
Torque Smoothness [Nm]	0.172	0.954	0.07	0.057	0.031	0.077
Velocity Cost [m^2/s^2]	44.98	193.78	94.68	242.44	137.31	100.34
RealAI Score	0.801	0.722	0.891	0.869	0.827	0.8

Table 6.1: Performance scores of various controllers for pendubot and acrobot experiments.

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). 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 non-learning-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

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. As previously predicted, time delay is the Achilles heel for RL-based methods, since high latency can disrupt the Markov decision process entirely.

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

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

This section is about explaining the hardware results. [to be filled]

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 [N^2m^2]	78.7	-	8.93	11.73	51.88	3.26
Torque Smoothness [Nm]	0.774	-	0.54	0.671	0.643	0.108
Velocity Cost [m^2/s^2]	114.04	-	84.61	118.38	242.34	109.77
Best RealAI Score	0.767	-	0.843	0.82	0.695	0.822
Average RealAI Score	0.298	-	0.839	0.817	0.547	0.821

Table 6.3: Real hardware performance scores of multiple controllers for pendubot and acrobot experiments.

6.4 Conclusion and future work

This section is to talk about things to be done.

Bibliography

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

[2] C. Aguilar-Ibañez, M. S. Suárez-Castañón, and O. O. Gutiérres-Frias. "The direct Lyapunov method for the stabilisation of the Furuta pendulum". In: *International Journal of Control* 83.11 (2010), pp. 2285–2293.

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

[4] 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.

[5] 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.

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

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

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

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

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

[11] 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).

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

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

[14] 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.

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

[16] 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.

[17] 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.

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

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

[20] 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.

[21] 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.

[22] 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).

[23] 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.

[24] 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).

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

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

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

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

[29] 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.

[30] 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.

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

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

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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).

[41] 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.

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

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

[44] 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.

[45] 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.

[46] 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).

[47] 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).

[48] 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.

[49] 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.

[50] 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.

[51] 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.

Appendix

A An appendix

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

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