DLR

Hochschule
Kempten
University of Applied Sciences

Master Thesis

Automation and Robotics

Control of elastic joint robots by combining the ESP concept with reinforcement learning

Christoph Saad
Matriculation No.: 360127

Examiner	Prof. Dr. Jürgen Brauer
Work submitted on	28.02.2023
Performed at	Deutsches Zentrum für Luft und Raumfahrt e.V.
Supervisors at DLR	Dipl.- Ing. Manuel Keppler, Antonin Raffin
Address of the author	Wiesenstr. 11, 87787, Wolfertschwenden christophsaad@gmail.com

I Confidentiality clause

Sperrvermerk

Die nachfolgende Arbeit enthält vertrauliche Informationen und Daten der Firma Deutsches Zentrum für Luft und Raumfahrt e.V.. Veröffentlichungen oder Vervielfältigungen - auch nur auszugsweise oder in elektronischer Form - sind ohne ausdrückliche schriftliche Genehmigung der Firma Deutsches Zentrum für Luft und Raumfahrt e.V. nicht gestattet. Die Sperrfrist gilt bis zum 31. März 2024. Die Arbeit darf bis zum Ablauf der Sperrfrist nur für Prüfungszwecke verwendet werden.

Confidentiality clause

The following work contains confidential information and data of the German Aerospace Center (Deutsches Zentrum für Luft und Raumfahrt e.V.). Publications or duplications - even in extracts or in electronic form - are not permitted without the express written permission of the German Aerospace Center. The blocking period is valid until March 31, 2024. Until the end of the blocking period, the work may only be used for examination purposes.

II Abstract

In this thesis, we aim on quantifying and improving the tracking ability of the ESP controller for the elastic joint robot. We take the David system as a base for our study. We also show the effect friction has on the tracking error of the controller. We use a RL based approach that gives a trajectory offset to compensate the error. We did run several training experiments on a single-DoF simulation model of an elastic joint to verify and validate our concept and pipeline. This concept showed some promising results improving the tracking ability of the controller, but also showed some stability limitations. We also notice that that the tracking error increases with the trajectory frequency. To overcome these limitations, we proposed a second approach that may resolve these issues by adding an interpolator to the pipeline.

III Acknowledgement

In this thesis I will be taking the last step to finish my master in automation and robotics. Therefore, I will take a moment to thank all the people who helped me along the way.

First, I would like to thank my professor Prof. Dr. Jürgen Brauer who helped me by providing guidance and feedback throughout the thesis.

I am also deeply grateful to Dipl. -Ing. Manuel Keppler, my supervisor at DLR for giving me the opportunity to do the research and for the support and during my thesis. I would also like to express my sincere gratitude to my colleagues Antonin Raffin and Xuming Meng who provided me with their technical knowledge and help, in the field of reinforcement learning and control respectively. I also would like to thank the rest of the David team, and colleagues at DLR who made me feel very welcome and kept me motivated during my time there.

Finally, I need to thank my family who supported me during my studies especially during my thesis and kept me motivated to achieve and accomplish my dreams.

Weßling, February 2023
Christoph Saad

IV Glossary

DLR: Deutschen Zentrum für Luft und Raumfahrt, German Space Center.

DoF: Degrees of Freedom.

ESP: Elastic Structure Preserving.

ESπ: Elastic Structure Preserving Impedance.

FSJ: Floating Spring Joint.

ILC: Iterative Learning Control Theory.

LN: Links and Nodes.

RMC: Robotics and Mechatronics.

MDPs: Markov Decision Processes.

ML: Machine Learning.

PD: Proportional-Derivative.

RL: Reinforcement Learning.

SAC: Soft Actor-Critic.

SB3: Stable-Baselines3.

SBX: Stable-Baselines Jax.

VSA: Variable Stiffness Actuators.

V Table of Contents

I Confidentiality clause II

II Abstract III

III Acknowledgement IV

IV Glossary V

V Table of Contents VI

VI Table of figures and tables VIII

1 Introduction 1

2 Definitions and Fundamentals 2
- 2.1 Physics Model . . . 2
  - 2.1.1 Elastic Link . . . 2
  - 2.1.2 Friction Model . . . 4
- 2.2 Controller . . . 7
  - 2.2.1 ESP . . . 7
  - 2.2.2 ESπ . . . 9
- 2.3 Reinforcement Learning . . . 10
  - 2.3.1 Basics . . . 11
  - 2.3.2 Exploration vs. Exploitation . . . 13
  - 2.3.3 Algorithms . . . 13

3 State of Science and Technology 15

4 Analysis and Evaluation of Requirements 17

5 Concept and Methodology 19
- 5.1 Baseline and reference . . . 19
- 5.2 Direct trajectory modification . . . 20
- 5.3 Pose modification and interpolation . . . 22

6 Implementation 23

6.1 Baseline and reference 23
6.2 Direct trajectory modification 24
6.3 Pose modification and Interpolation 32

7 Evaluation of the Results 35
7.1 Baseline and reference 35
7.2 Direct trajectory modification 36

8 Summary and Outlook 46

9 Bibliography 47

VI Table of figures and tables

List of Figures

1 The spring mechanism, gear box and main actuator of the FSJ. [3] 2
2 The variable stiffness mechanism of the FSJ. [3] 3
3 Dynamic model of the FSJ. [4] 3
4 Friction-velocity curve. 5
5 Friction-velocity curve for the simplified model. 6
6 Intermediary closed loop dynamics of a single DoF elastic joint. [4] 7
7 Closed loop dynamics: link-side damping and regulation. [4] 9
8 Closed loop dynamics for the ESπ controller. [5] 10
9 Agent-environment interaction loop example. 11
10 Agent trajectory modification concept pipeline. 20
11 Agent trajectory modification and interpolation concept pipeline. 22
12 Traditional trajectory tracking loop. 24
13 Friction-velocity curve parameters. 25
14 Trajectory oscillations as seen in the velocity curve. 30
15 Interpolation problem. 32
16 Root mean and maximal squared tracking error for experiments Group 4. . . . 35
17 Range and limit analysis of single DoF model. 37
18 Group 5 experiment relative error decrease during training. 38
19 Group 6 experiment relative error decrease during training. 39
20 Trajectory of experiments 0046. 40
21 Trajectory, control input and error of the last episode in Group 7. 41
22 Trajectory, control input and error of the last episode in Group 8. 42
23 Group 9 experiment relative error decrease during training. 43
24 Trajectory, control input and error of the last episode in Group 10. 45

List of Tables

1 Experiment parameters for Group 1 and Group 4. 26
2 Experiment parameters for Group 2. 27
3 Experiment parameters for Group 5. 28
4 Experiment parameters for Group 6. 29
5 Experiment parameters for Group 7. 30

6 Experiment parameters for Group 8. 31
7 Experiment parameters for Group 9. 31
8 Experiment parameters for Group 10. 31

1 Introduction

The DLR, short for Deutschen Zentrum für Luft und Raumfahrt, meaning German Space Center, is one of the leading research institutes in various fields worldwide. One of the institutes under the DLR is the institute of Robotics and Mechatronics [1] at the Robotics and Mechatronics Center (RMC) in Weßling near Munich. As the name suggests, the main topic of research at the institute is robots, including legged-, mobile-, flying-, medical- and compliant robots. One interesting project at the institute is the David robot [2]. David is an anthropomorphic robot with variable stiffness actuators (VSA), and 41 degrees of freedom build with the purpose of achieving human capabilities.

For the arms, new VSA were developed by the DLR as a floating spring joint (FSJ) [3] which allows the mechanical compliance of the robot arms by dynamically changing the stiffness of the elastic spring between the motor and link. To control the arms a new type of controller called elastic structure preserving (ESP) controller [4] had to be developed and latter improved becoming the elastic structure preserving impedance (ESPi/ESπ) controller [5]. Both the ESP and ESπ controllers prove to be stable and show a high positioning accuracy. However, the trajectory tracking may be in some cases less optimal. While the tracking error appears to be close to zero within the simulation and without any friction model, when friction is introduced, both in the simulation and on the real hardware, a small positioning error appears.

From practical experience we know how powerful machine learning (ML) in general, especially reinforcement learning (RL), can be for building, optimizing, and improving a systems controller and passivity. We also know that while ML show good results for controlling, no guarantee for stability can be made.

Therefore, the combination of RL with a more traditional controller like the ESπ is proposed. The RL agent will be therefore responsible for reshaping the desired trajectory given to the ESπ controller resulting, hopefully, in a more accurate trajectory tracking ability compared to an EPπ only approach.

To evaluate the feasibility of this approach and with the motivation of improving the transient performance of the manipulator's accuracy, a master-thesis with the topic "Control of elastic joint robots by combining the ESP concept with reinforcement learning" will be carried out. The goal will be to reshape and optimize trajectories to reduce the position and the total control error. The focus of the thesis will be evaluating the current tracking error; implementing a RL pipeline allowing the training of such RL agents, both in simulation and on real hardware; running training experiments with various parameters; understanding the effects of these parameters on the tracking error; and finally evaluating the achieved tracking error and comparing it to the previous method.

2 Definitions and Fundamentals

In the next section we will explain some physics background for the elastic link as well as the friction model used. The we will explain the control model behind both the ESP and ESπ controllers. Finally, we will have a brief overview about reinforcement learning, explain the difference between exploration and exploitation, and explain the workings of the RL-algorithm chosen and why it was the most fitting option for our use case.

2.1 Physics Model

We will show and explain both the single- and multi-DoF model of the FSJ elastic link we will be using. We will then introduce the friction model we will be using in the simulation.

2.1.1 Elastic Link

To simulate the elastic links of David's arm, we first must understand the dynamics of the FSJ. As described in the original FSJ paper [3], the joint consists of a main motor coupled to a harmonic drive gear box. A variable stiffness mechanism is used between the gear box and the link. The workings of the FSJ are shown in the Figure 1 from the paper.

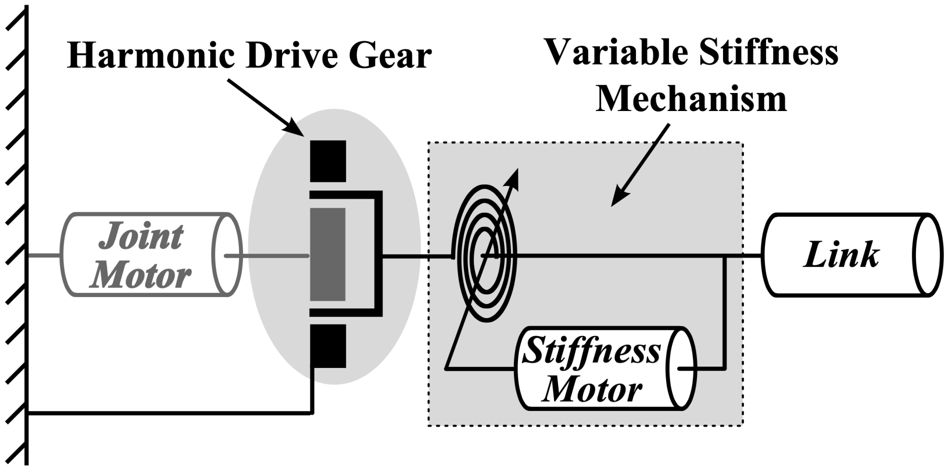

Figure 1: The spring mechanism, gear box and main actuator of the FSJ. [3]

The variable stiffness mechanism is achieved through an additional motor used to control the relative rotation of two cam disks. One of the cam disks is connected to the secondary motor, the other to the harmonic drive gear box. Between the two cam disks, two cam rollers, connected to the link, are used. Due to the progressive shape of the disks, the rollers would allow a progressive deflection between the link and main motor side. This deflection could be considered as the joint stiffness. By changing the relative rotation of the two cam disks using the secondary motor, we could achieve a different desired stiffness and therefore making a VSA. We should note here that only the equilibrium position can be set by the secondary motor and any additional torque or deflection would change the actuator stiffness. In the FSJ paper [3] the following Figure 2 was used to show the working of this mechanism.

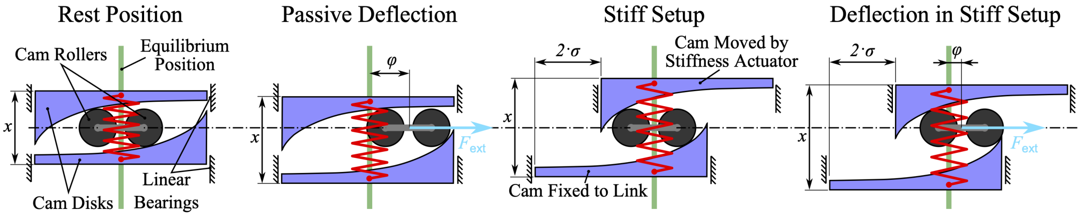

Figure 2: The variable stiffness mechanism of the FSJ. [3]

Now that we understand the workings of the FSJ, we can start to describe the dynamic model we will be using to describe it. As shown in the ESP paper [4], we could model the joint stiffness as a spring. The variable stiffness value will be dependent of the joint deflection. In the Figure 3 we see the dynamic model of the FSJ.

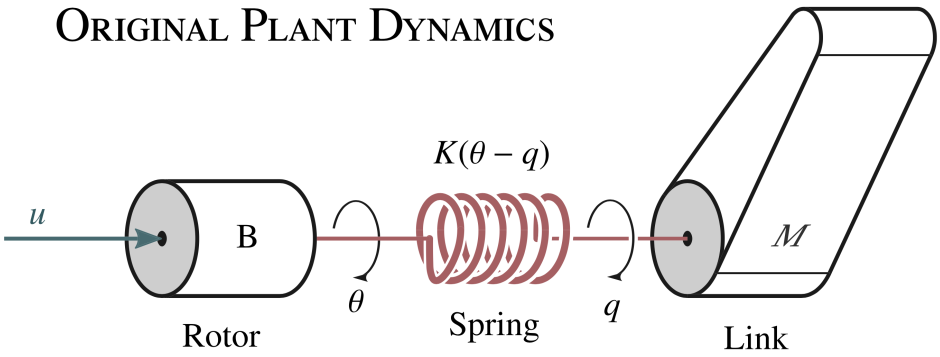

Figure 3: Dynamic model of the FSJ. [4]

This dynamic model could be describe using the following equations (1), (2) from the ESP paper [4].

$$M\ddot{q} = K(\theta - q) \tag{1}$$

$$B\ddot{\theta} + K(\theta - q) = u \tag{2}$$

With u representing the control input, B and M the motors and the link's inertia, θ and q the motor and link position and $\ddot{\theta}$ and $\ddot{q}$ the motor and link acceleration respectively.

Now that we have defined a single elastic link we can describe an n-link robot with compliant joints that we will be using. Under the assumption that the angular kinetic energy of each rotor is only from its own rotation, a simplified model (3), (4) for that kind of robot, first introduced in [6], was considered in the ESP paper [4].

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} = -g(q) + \psi(\theta - q) + \tau_{\text{ext}} \tag{3}$$

$$B\ddot{\theta} + \psi(\theta - q) = u \tag{4}$$

With M and $B \in \mathbb{R}^{n\times n}$ being respectively the inertia matrix of the rigid links and the diagonal matrix of the actuator inertias reflected through the respective gearboxes. The vectors $C(q,\dot{q})\dot{q}$ and $g(q)$ represent the Coriolis and centrifugal forces and gravitational forces respectively. ψ represents the generalized elastic forces derived by the spring potential function.

2.1.2 Friction Model

In addition to the elastic link model, a friction model will be introduced. In general, friction can be defined as force acting against the movement of the object and depends on the velocity of that object. Having a rotating system, the friction in our case will be a counter-torque on the link.

In its simplest form, we could only consider the viscous friction. This type of friction is linearly proportional to the velocity of the considered link and could be described by using equation (5), with k_d being the linearity constant.

$$M_{fric-viscous}(\dot{q}) = k_d \cdot \dot{q} \tag{5}$$

This viscous friction model while being very simple and easy to understand does not deliver the accuracy we need for our experiment. Therefore, additional types of friction must be considered. We know that for very low velocities we see a spike in the amount of friction measured. This will fade until we reach a linear curve. Compared to the viscous friction model, the final linear curve should be vertically offset in the same direction of the velocity: positive velocities moving the friction curve up, and negative velocities down resulting in more absolute friction. This is called Coulomb friction model (6).

$$M_{fric-Coulomb}(\dot{q}) = \text{sign}(\dot{q}) \cdot k_c \tag{6}$$

To model the initial fade, the Stribeck friction model will be considered (7).

$$M_{fric-Stribeck}(\dot{q}) = \text{sign}(\dot{q}) \cdot k_s \cdot e^{-|\dot{q}|} \tag{7}$$

Adding all three models (8) will result in a more accurate friction model.

$$M_{fric}(\dot{q}) = \text{sign}(\dot{q}) \cdot k_c + \text{sign}(\dot{q}) \cdot k_s \cdot e^{-|\dot{q}|} + k_d \cdot \dot{q} \tag{8}$$

The resulting friction-velocity curve can be seen in Figure 4. Here we can clearly see the jump and sudden change in direction of the friction when the velocity goes through zero.

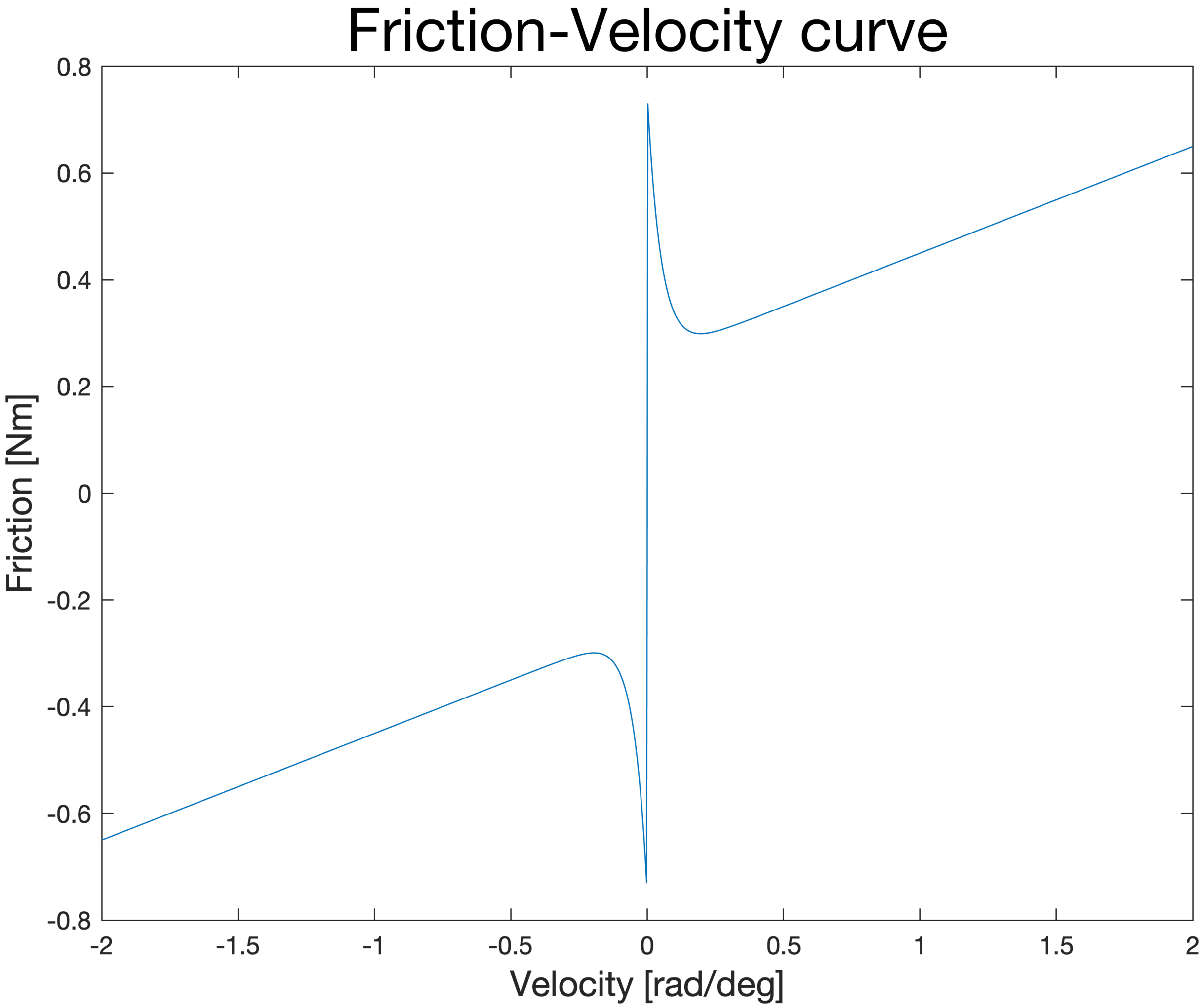

Figure 4: Friction-velocity curve.

This model while being fairly accurate for our use-case, can't be used due to the jump described above. From a mathematical point of view, this is due to the sign function. Therefore, we will consider replacing it with a tanh function. This would allow for a smoother transition between -1 and 1 compared to a sign function. With this change the friction model can be described with equation (9).

$$M_{fric}(\dot{q}) = k_c \cdot \tanh(\dot{q}) + \tanh(\dot{q}) \cdot k_s \cdot e^{-|\dot{q}|} + k_d \cdot \dot{q} \tag{9}$$

The resulting friction-velocity curve in Figure 5 does not show any discontinuities while remaining accurate enough for our use-case.

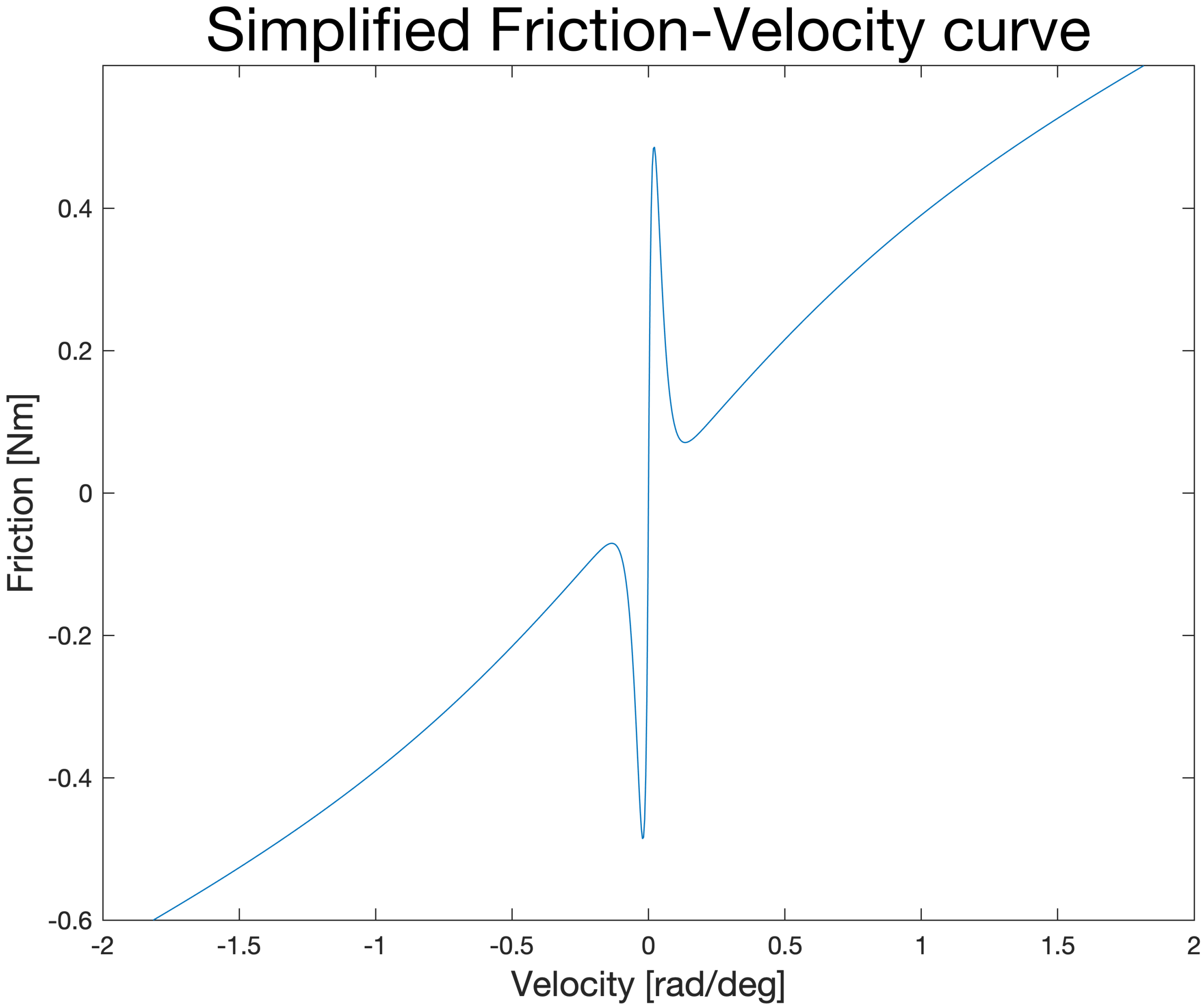

Figure 5: Friction-velocity curve for the simplified model.

Finally, one remaining thing should be considered. Based on pervious experiences, the directive that the friction at the peak near zero should roughly be double the friction at the valley was given. With that in mind, two constant, c_1 and c_2, were introduced with a slight change in the current model (10). This would result in very similar curve as above while satisfying that constraint.

$$M_{fric}(\dot{q}) = k_c \cdot \text{tanh}(\dot{q}) + k_s\,\text{tanh}(\dot{q}) \cdot c_1 \cdot e^{-c_2 \cdot |\dot{q}|} + k_d \cdot \dot{q} \tag{10}$$

With a little bit of experimentation, values for c_1 and c_2 that satisfies this requirement were found, being 130 and 50 respectively.

2.2 Controller

Here, we will explain the dynamics and control laws of the ESP and ESπ controllers.

2.2.1 ESP

As explained in [4],[7],[8] and [9], the ESP controller was developed with the purpose of achieving motion tracking and assignable damping simultaneously for the link of a compliant robot with nonlinear elastic transmission while remaining asymptoticly stable, practically feasible and achieving good performance.

To first explain the workings of the ESP controller, [4] considers the simple single DoF model described before with equations (1) and (2). The goal of this controller is to achieve a model dynamic with link side damping. Therefore, an intermediary closed loop dynamics model with a link side velocity-proportional damper D, shown in Figure 6, was considered for the regulation case.

$$M\ddot{q} = K(\eta - q) - D\dot{q} \tag{11}$$

$$B\ddot{\eta} + K(\eta - q) = \bar{u} \tag{12}$$

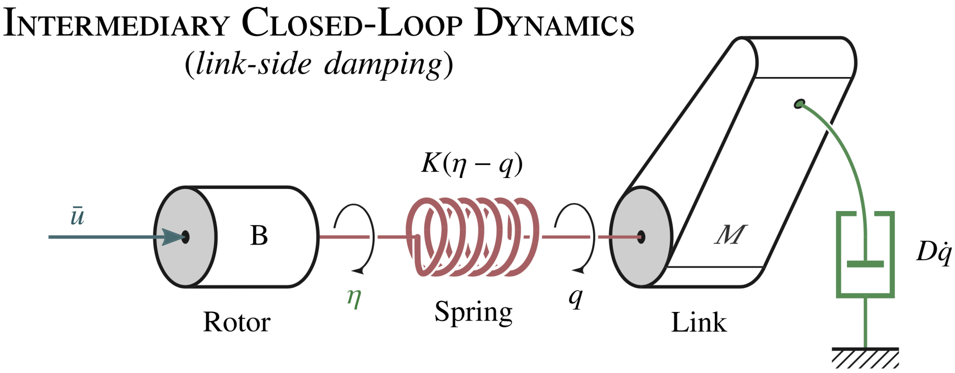

Figure 6: Intermediary closed loop dynamics of a single DoF elastic joint. [4]

With that a new variable η was introduced as the new transformed motor position. Comparing the new dynamics (11) and the original dynamics (1) the transformation (13) between the two different motor coordinates θ und η could be found.

$$\theta = \eta - K^{-1}D\dot{q} \tag{13}$$

To achieve equivalency between the two motor dynamics (12) and (2), [4] considers the following intermediary control law (14).

$$u = \bar{u} - BK^{-1}Dq^{(3)} - D\dot{q} \tag{14}$$

Furthermore, to add link-side regulation, a motor PD law (15) for $\bar{u}$ was considered.

$$\bar{u} = -K_D\dot{\eta} - K_p(\eta - \eta_d) \tag{15}$$

With K_P and K_D being the achieved spring, damper coefficient on the motor side from the closed loop dynamics shown in Figure 7, and η_d being the desired transformed motor position. In the regulation case $\eta_d := q_d$ remains constant.

Bringing everything together will result in the following closed loop dynamics (16) (17) for the regulation case.

$$M\ddot{q} = K(\eta - q) - D\dot{q} \tag{16}$$

$$B\ddot{\eta} + K(\eta - q) = -K_D\dot{\eta} - K_P(\eta - \eta_d) \tag{17}$$

Similarly, [4] shows how to design an ESP controller for the tracking case. For that the link-side position q was replaced by the link-side error $\tilde{q} = q - q_d$, with q_d being the desired link-side position. Additionally, a tracking term was added for the intermediary control law (18).

$$u = \bar{u} \underbrace{-BK^{-1}D\tilde{q}^{(3)} - D\dot{\tilde{q}}}_{\text{damping terms}} \underbrace{+BK^{-1}Mq_d^{(4)} + (B + M)\ddot{q}_d}_{\text{tracking terms}} \tag{18}$$

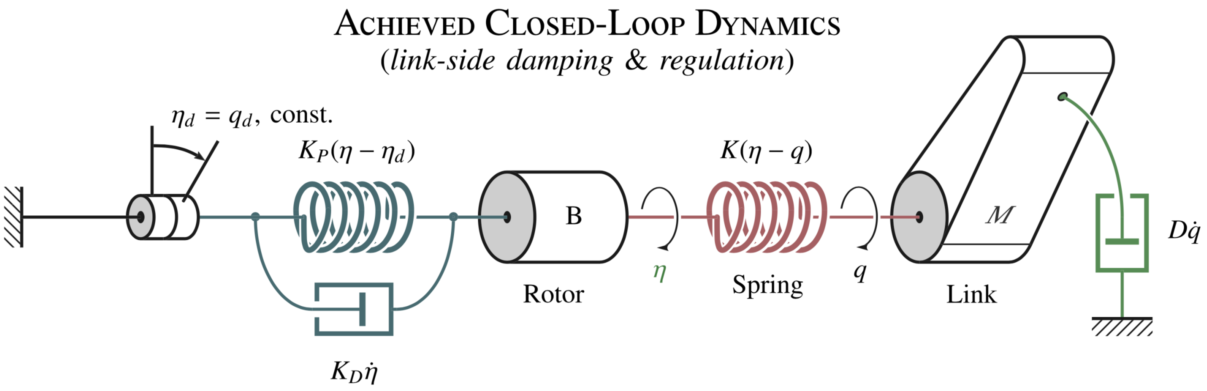

Figure 7: Closed loop dynamics: link-side damping and regulation. [4]

Following similar steps to the regulation case, the closed loop dynamics for the tracking case can be defined as follow (19), (20).

$$M\ddot{\tilde{q}} = K(\eta - \tilde{q}) - D\dot{\tilde{q}} \tag{19}$$

$$B\ddot{\eta} + K(\eta - \tilde{q}) = -K_D\dot{\eta} - K_P\eta \tag{20}$$

One stability condition imposed by the controller on the desired trajectory is $q_d(t) \in \mathbb{C}^4$.
The experimental results in [4] show the excellent damping performance of the ESP controller as well as it is good regulation and tracking abilities and resilience to external disturbances compared to a more traditional motor PD controller.

2.2.2 ESπ

The results for the ESP while good for rather stiff joints, were unsatisfactory for highly elastic robots. The vibration damping performance in those cases was still laking, therefore a new concept had to be developed. The Elastic structure preserving impedance (ESπ) controller while like the ESP controller, moves the spring from the motor to the link side.
To show the working of this controller [5] takes similar approach to [4]. For the same original dynamics shown in Figure 3 the ESπ adds a spring on the link side. Additionally, the control law (21) was chosen, achieving the following closed loop dynamics for the single-DoF case in Figure 8. Described by equations (24) and (25).

$$u = \bar{u} + BK^{-1}\ddot{n} + n \tag{21}$$

$$\bar{u} = -D_\eta \dot{\eta} \tag{22}$$

$$n = -D_q \dot{q} - K_q(q - q_d) \tag{23}$$

$$M\ddot{q} = K(\eta - q) - D\dot{q} - K_q(q - q_d) \tag{24}$$

$$B\ddot{\eta} + K(\eta - q) = -D_\eta \dot{\eta} \tag{25}$$

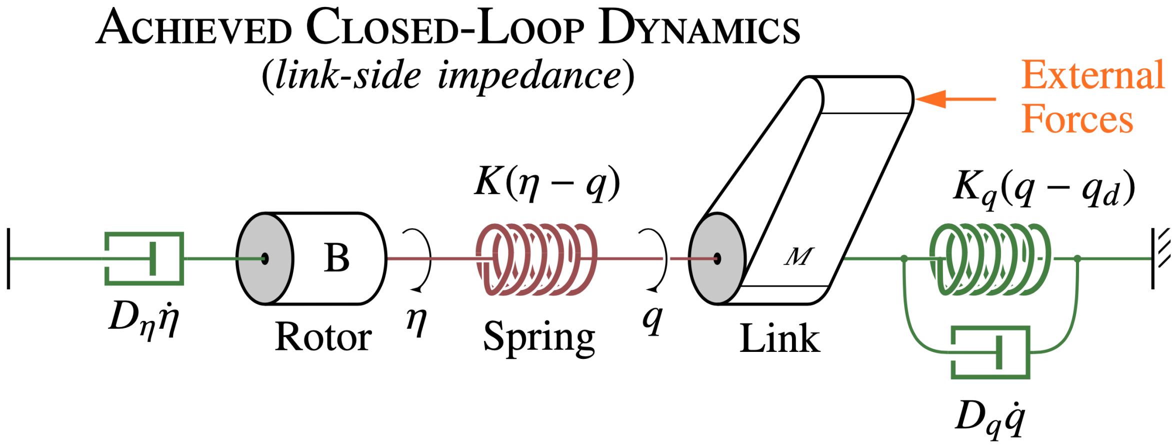

Figure 8: Closed loop dynamics for the ESπ controller. [5]

Similarly, [5] shows how to describes the closed loop dynamics of a multi-link robot with cancelled gravity using equations (26) and (27). With τ_x being the impedance term.

$$M\ddot{q} + C(q, \dot{q})\dot{q} = \psi(\eta - q) + \tau_x(\tilde{x}, \dot{x}) + \tau_{\text{ext}} \tag{26}$$

$$B\ddot{\eta} + \psi(\eta - q) = -D_\eta \dot{\eta} \tag{27}$$

The experimental results in [5], [7] and [9] show excellent damping performance and very precise stiffness control.
Also, for the ESπ we need to impose the stability condition $q_d(t) \in \mathbb{C}^4$ on the desired trajectory.

2.3 Reinforcement Learning

In this section we will discuss the basics of reinforcement learning, we will show the basic training loop model of RL and then explain the difference between deterministic and stochastic policies a how they make the difference between exploration and exploitation behaviours of the RL-agent. Finally, we will present the algorithm requirements for our robotic application and then explain why we chose the Soft Actor-Critic (SAC) algorithm and explain how it works.

2.3.1 Basics

Reinforcement Learning (RL) is relatively a new subsection off the Machine Learning (ML) paradigm that focuses on teaching an agent to do some defined tasks through trial and error. This is achieved by letting the agent interact repeatedly at each time-step t, with an environment, receiving an observation o_t and giving an action a_t in return, while also getting back a reward r_t based on his performance accomplishing the desired task. The learning is not limited to only learning one main task, additional subtasks and side conditions could be considered in the reward. This cyclic interaction between agent and environment is best shown by the model in Figure 9.

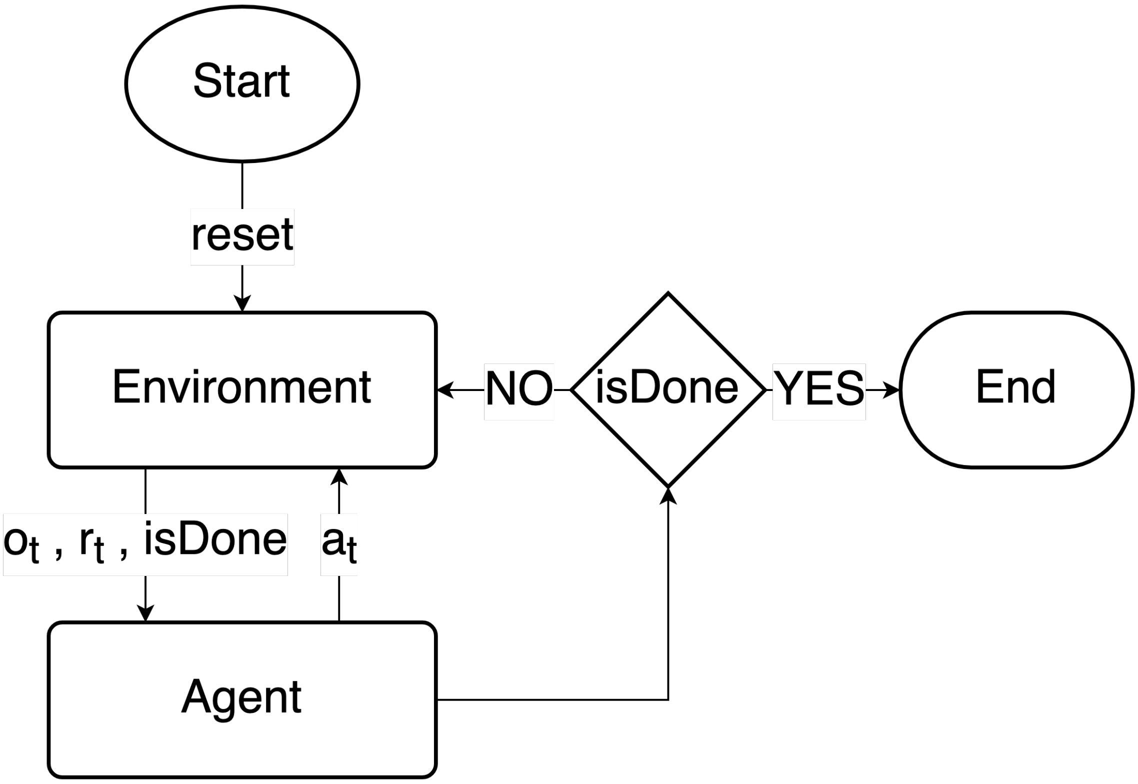

Figure 9: Agent-environment interaction loop example.

In the field of robotics, RL is used to teach robots to do new things or optimize existing tasks. This is especially interesting for tasks that are very complex to accomplish using traditional programming and control methods.

First, we will define the RL environment. This could be thought of as a separate world where the agent lives, acts, and learns, both as a simulation or real hardware. As OpenAI explains in [10], each environment has an internal state s_t and an observation state o_t. The first describes the entire state of the world and the latter is a subsection of the world state that is given to the agent to act on. We can differentiate between partially and fully observed environments which indicates the amount of information given by the observation state. For fully observed environments the observation state is equal to the internal state. Furthermore, they explain [10] that the commands sent by the agent to control the environment must follow a predefined action space a_t. For classic Atari games, a discrete action space is usually used allowing actions like go Up, Down, Left or Right to be taken. For most robotic applications like ours a continues action space would be more suitable since it allows a more precise and continues control of physical vectors like the motor position and velocity.
The agent is responsible for taking actions and learning. To do so the agent learns a state dependent policy that decides what action should be taken to maximizes the reward r_t. OpenAI explains [10] the difference between two types of policies: deterministic μ

$$a_t = \mu(s_t) \tag{28}$$

and stochastic π.

$$a_t \sim \pi(\cdot|s_t) \tag{29}$$

The agent interacts with the environments through a series of states and actions forming a trajectory τ also called an episode [10].
To steer the learning in the right direction a well thought out reward function R

$$r_t = R(s_t, a_t, r_{t+1}) \tag{30}$$

must be defined. The goal of the agent will be to maximize the cumulative reward also known as return over the entire length of an episode [10].
On a real and practical systems like ours, the system must obey Markov property indicating that the transition only depends only on the current state and actions. A formal mathematical definition of RL would be through the Markov Decision Processes (MDPs), a 5-tuple $\langle S, A, R, P, \rho_0 \rangle$

[10]. Where S and A being the set of valid states and actions respectively; R and P the reward and transition functions and finally ρ_0 the initial state.

2.3.2 Exploration vs. Exploitation

To be able, learn effectively the agent must explore its environment efficiently. The goal would be to try-out as much state-action combination as possible. Ideally all combinations would be tried, but this would be very difficult to do the larger and more complex the environment is, especially with continues action and state spaces.

During training, the agent must seek to obtain the highest reward possible. The agent would therefor try to take the action leading to the highest expected reward. This is usually seen as exploitative behaviour, since the RL-agent would only consider known state-action combinations that leads to the highest rewards and neglect unknowns state-action combinations.

Ideally the agent should find a balance between exploring the environment and exploiting the reward function. This balance should move during training from leaning more to the exploration side at first to leaning towards the exploitation side. For real hardware training, in the field of robotics too much exploratory freedom could lead to expensive accidence and hardware failure. During deployment the explorative side of the agent is kept to the minimum or even completely shut off to avoid unexpected behaviours and preserve the stability of the system.

2.3.3 Algorithms

To choose an adequate RL-algorithm for our application we must first consider the systems requirements. For most robotic applications a continues action space is used and therefore a compatible algorithm must be chosen. In addition, we must consider the sample efficiency and the parallelization properties of the algorithm. For our case, we are considering on hardware training which would not allow parallelized training and therefore higher sample efficiency is needed for faster training. Additionally, we must consider the exploration aspect of the training algorithm and whether it is imbedded in the objective function or must be added externally which adds additional complexity to the training pipeline. Considering all these requirements, we decided to use the Soft Actor-Critic algorithm.

The Soft Actor-Critic (SAC)[11] algorithm falls under the model-free RL paradigm, meaning that the agent doesn't need the knowledge on how the environment model works. This would simplify the usage of the algorithm and increase its versatility. While regular model-free RL algorithms aren't usually sample efficient, SAC overcomes this issue by using an off-policy algorithms allowing the reuse of experience during training. Additionally, SAC increase its resiliency regarding the hyper-parameters used by being more stable compared to other off-policy learning algorithms.

SAC uses a more general maximum entropy $\mathcal{H}$ objective favouring stochastic π policies by augmenting the standard RL objective function r with the expected entropy $\mathbb{E}$ of the policy over $\rho_\pi(s_t)$ for the length of learning interval T resulting in following objective function J (31).

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{31}$$

To adjust the weight of the entropy term compared to the reward, the temperature parameter α is used. This would also control the stochasticity of the used policy. During training the goal would be to minimize the objective error. This stochastic term results in a built-in exploratory behaviour during training. With a well-chosen α parameter the exploratory behaviour of the agent would start big and decrease during training giving a better robustness against local maxima traps. To evaluate or use SAC with a deterministic policy α could be set to 0, and therefore removing the stochastic term.

The most important hyper parameter of SAC according to its original paper [11] are namely reward scaling and target value update smoothing constant.

3 State of Science and Technology

In most control applications, the controller cannot follow the trajectory perfectly resulting in a tracking error. This is usually caused by unaccounted disturbances or overlooked external forces as well as the inability for the controller to compensate for them on its own. Many different approaches were taken in different fields and process applications to compensate for these forces and therefore improving the tracking ability of the controller. In this section we will be discussing a few of these approaches, their strengths, and weaknesses and how they relate or differ of each other.

The first method we will be discussing is trajectory shaping. The idea behind this is to deform the trajectory given to the controller in such a way that the controller error produce exactly the original desired trajectory. As discussed in [12], such an approach for robots in milling applications. They show how the process force could be estimated and accounted for in the kinematic model and the trajectory planning. Through an off-line calculation for a compositional milling path correction, they managed to achieve a higher accuracy for their milled product. This could be both seen as good, in a way that they managed a higher accuracy, or bad because they had to calculate the path correction off-line in advance leading in less flexibility in the trajectory planning. For them to calculate the offset they both needed a high accuracy model of the robot kinematics and dynamics as well as a highly accurate model of the milling process.

Another possible method is to change the control input directly. In simple linear control applications, the controller, for example a PD controller, is used as a basic error compensator. To better control the system an additional compensation term could be added to move the equilibrium state of the open loop system. In [13] they use this idea to control a non-linear system by using a feedforward compensator. They build the feedforward term by inverting the non-linear model of the controlled system. Similarly, [14] uses a feedforward compensator to control a flexible single-DoF robot. Both approaches [13] and [14] have shown good results, however they both need a good knowledge of the system to invert the system for the feedforward term. Any inaccuracy in that model would need to be compensated by the controller, limiting the system dynamics.

To solve the modelling problem the feedforward term could be learned. This concept is called iterative learning control theory (ILC). ILC is an old concept, as shown in [15] a linear feedforward term could be learned which will simplify the computation required to control the robot. While ILC is usually best suited for repeated fixed point to point applications, requiring a new feedforward function to be learned for each new point pair, [16] tries to overcome this by varying the initial conditions used for learning leading to increased robustness and increasing the performance compared to traditional ILC for a wider range of trajectories. While that approach did show great results, it still has its limitations since the trajectory range allowed was still limited.

A more recent approach [17] tries to incorporate neural network with ILC. The feedforward term is split into two, a main generalized feedforward function and a state dependent neural network term. Both the function and the neural network can be learned at the same time. While this expands the application range it's still limited to the range used for training, since anything outside of it could lead to unpredictable results from the neural network.

To conclude, the approaches we presented all show improvements in their respective fields and application, however they all have their shortcomings. While some need intimate knowledge of the process, robot dynamics or both, other have a limitation on the application range since for each new trajectory a new set of parameters must be learned or have a fixed predefined limited application range.

4 Analysis and Evaluation of Requirements

Before being able to improve the current system's ability to accurately follow a given trajectory, we need to evaluate the current system and define some metrics to quantify the tracking error.
First, we must define some reference trajectories that will demonstrate the tracking ability of the controller. These should be able to reflect the tracking ability on simple and standard cases as well as on those edge cases where a tracking error is expected and cannot be avoided, such as cases with a sudden change in direction requiring an infinite amount of acceleration.

To evaluate the improvements, we need to first define the error metrics. We need to check for the deviation of the link pose, velocity and acceleration compared to the desired trajectory. The goal would be to have a tracking error as close to zero as possible. Further metrics, such as mean and maximal error of the error, could be used to facilitate the understanding and the visualization of the tracking error of the trajectory.

We know that the ESπ controller can achieve perfect tracking under ideal theoretical conditions, such as in the frictionless simulation environments. However, its performance would decrease when friction is introduced. Therefore, we would also be evaluating the effect of friction on the tracking ability of the different controllers. For this a more accurate friction model will be needed. To ensure that friction can be compensated for, the friction model must remain continuous and not show any unrealistically high gradients. Therefore, we need to find a compromise between having an accurate friction model and being able to compensate for it.

In addition to friction, we would assume that the speed at which the trajectory is being followed may also influence the tracking ability of the controller. This should also be considered and evaluated.

In addition to knowing the ability of our current system, an environment should be built to run and test all the experiments we intend to perform. The environment should conform to the Gym [18] interface, since it is widely adopted in the RL world. The environment would build a defined layer between the RL agent and the simulation or the real robot. The environment would have to connect to the simulation or robot using Links and Nodes (LN) [19]. LN was developed at the DLR to allow real-time communication between multiple nodes. In our case we would use the publisher/subscriber communication model offered by LN to connect the different components we will be needing.

In addition to the environment, the complete RL training pipeline should be built. The pipeline should encompass a trajectory generator, simulation or real robot connected to the environment and of course an agent and RL training loop. We must also ensure that all parts can communicate between each other as needed and stay synchronized throughout the training process. For the agent and RL training algorithm the Stable-Baselines3 (SB3) [20] library, a well know RL open-source library with a PyTorch [21] backend, can be used.

We also need to make sure that our approach does overcome to the same shortcomings of the previous works discussed in the literature review 3. The approach should not be dependent on the process or the robot dynamics and should not present any range limitations or have the need to retrain for each new trajectory needed. To better understand the problem, validate the pipeline and iterate fast over different training parameters, a single DoF model of the elastic joint should be created.

After having validated the training pipeline, the training setup should be evaluated and tested on the complete multi-DoF simulation before running the training experiments on the real hardware.

At the end the tracking error of the ESπ + RL controller should be evaluated and compared to the initial ESπ only controller. Finally, the improvements made by our approach on the tracking ability as well as potential limitations can be shown.

5 Concept and Methodology

In this section we will discuss the concepts and the methodology used to build the RL-pipeline we will be proposing. First, we will show how we will create a baseline trajectory to compare our results. Then we will explain the concept behind our RL-pipeline. Finally, we will show how we could modify the pipeline to further improve our results.

5.1 Baseline and reference

To be able to properly evaluate the improvements made by our methods regarding tracking abilities of the ESπ controller, we must first evaluate the current tracking performance of the controller. To do so, we will need to define a set of reference trajectories that will be representative of real task and possible edge cases where the performance is expected to be worse. We also need to make sure to follow the $q_d(t) \in \mathbb{C}^4$ stability condition mentioned before, as well as to keep all physical limits of the robot.

We will start by defining a trajectory for the single-DoF model. Possible function to use were a high order polynomial with at least four degrees, a sin function, a chirp, or any combination between them. We chose to compose a trajectory out of two intervals, the first being a sin and the second a chirp function with a frequency starting the same as the sin function and linearly increasing over the interval, ensuring continuity between the segments. The resulting function could be generally described by equation (32).

$$q(t) = \begin{cases} A \cdot \sin(2\pi\omega_1 \cdot t + \varphi) & \text{if } 0 \leq t \leq t_1 \\ A \cdot \sin(2\pi\frac{\omega_2(t-t_1)+\omega_1(t_2-t)}{t_2-t_1} \cdot t + \varphi) & \text{if } t_1 \leq t \leq t_2 \end{cases} \tag{32}$$

In practice this trajectory can never be perfectly tracked. As we will later notice in the implementation, the starting velocity, acceleration, jerk, and snap are not all equal to zero for $t = 0$. In a simulated environment we will solve this issue by using initial conditions for the integrators as described in the implementation section 6.1. This, however, cannot be done on the real hardware and therefore an additional trajectory section using a higher order polynomial must be added to ensure a smooth transition from and to a complete stop. To choose the parameters of reference trajectory we need to consider the hardware limits of the system as well as the maximal frequency the controller can follow. To evaluate the worst-case scenario, we will need to push the system to its limits while keeping a little bit of freedom for the agent to modify the trajectory.

5.2 Direct trajectory modification

In contrast to most attempts, we discussed in the literature review section, we will not be offsetting the control input directly, but rather we will be modifying the desired trajectory. This will preserve the stiffness of the system and with the control law of our controller changing the control input and modifying the desired trajectory are equivalent without the risk of instability or the necessity of proving the system would remain stable with the control offset. Now that we know what we want to modify we can define the entire learning loop. As the last approach we showed [17], we will also be using a neural network to calculate a state dependent offset, however we will be using a reinforcement learning approach for the training. As shown in Figure 10, the RL-agent would get the next n normalized steps of the desired trajectory, as well as the current normalized state of the system from the RL-environment and output an offset for the trajectory. The offset will then be scaled by the environment and passed back to the robot or simulation through LN. The robot will then use it as input for the controller and publish the resulting current state. The loop would then continue until the last position of the original desired trajectory is used.

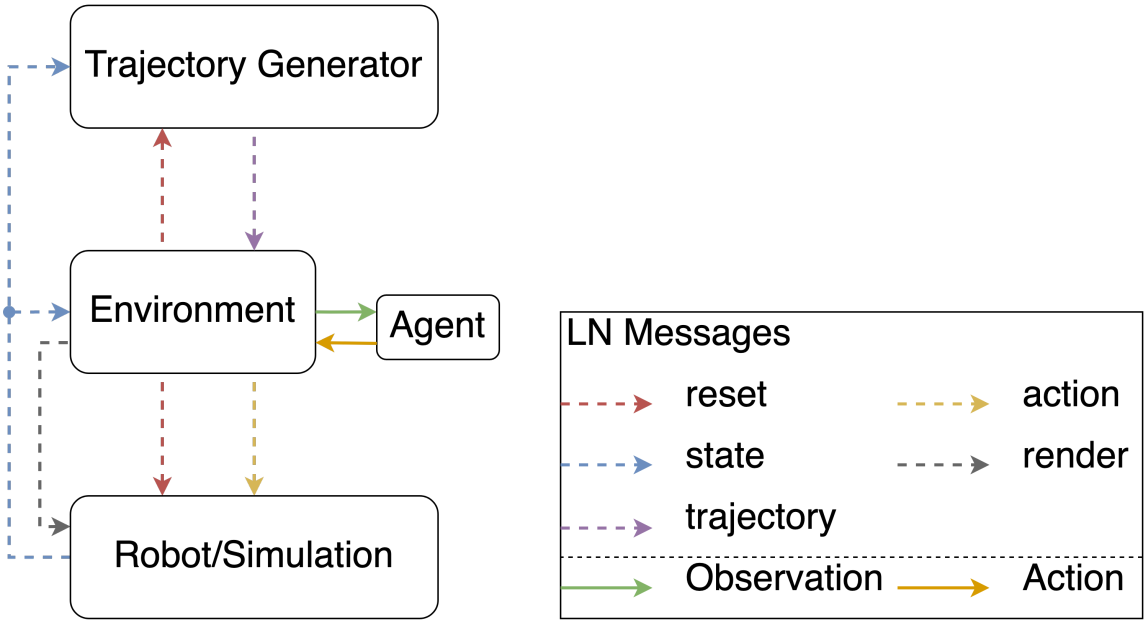

Figure 10: Agent trajectory modification concept pipeline.

The agent will have an input vector composed from the flattened n next desired trajectory steps vector for the link position q_d and its first four derivatives denoted here respectively by $\dot{q}_d$, $\ddot{q}_d$, $q_d^{(3)}$, $q_d^{(4)}$; and the last state vector composed from the current link position q, velocity $\dot{q}$ and acceleration $\ddot{q}$ and the motor position θ and its velocity $\dot{\theta}$ as well as the elastic torque ψ in that order. All these values would be collected from either the trajectory generator or the simulation by the environment which would in turn normalize them and pass them as an observation to the agent. On the other end the agent would then output a normed offset for the desired trajectory. The environment would receive the agent output as a new action, scale it and added it to the original desired trajectory creating a modified trajectory denoted by q_{mod}, $\dot{q}_{mod}$, $\ddot{q}_{mod}$, $q^{(3)}_{mod}$, $q^{(4)}_{mod}$. To properly normalize and scale the observation and action of the environment a range for these values must be found. While for some of them, q, θ, $\dot{\theta}$, u and spring deflection $|\theta - q|$ the robot datasheet could be checked, the rest need to be either calculated or estimated through experimentation. Any attempt to mathematically solve for the unknown variables proved to be difficult. Therefore, we set up an experiment to find the highest values reached by each of them. For the single-DoF model, we ran a simulation giving a sin or sin-chirp trajectory scaled between the maximal and minimal positions of the joints. We kept altering the frequency of the trajectory until we exceeded the joint limits and physical constraints of the robot. The limits that we were most interested in were the maximal control input $u_{max} = 80\text{Nm}$, the maximal motor velocity $\dot{\theta}_{max} = 720°/s$ and the motor deflection $|\theta - q|_{max} = 10°$. After we found a trajectory that managed to exceed all three of these constraints, we looked for the maximal value reached for $\dot{q}$, $\ddot{q}$, $q^{(3)}$, $q^{(4)}$ and ψ, added 5% and rounded the results to the nearest 50, and considered it as the range limit in both the positive and negative side. The exact range for these values is not critical. If we end up defining them smaller or bigger than needed, we would only be limiting the working range of the robot or losing some performance, neither of which are critical since we can always go back and adjust them.

Now that we defined the action- and observation space, we need to define the terminating state of the environment as well as the reward. The agent should terminate either if the end of the trajectory is reached or if the robot limits were exceeded. We will only be monitoring the values we know from the datasheet like joint pose q, motor pose θ and velocity $\dot{\theta}$, control input u and spring deflection $|\theta - q|$. For the reward function, we will be using the error between the original desired trajectory q_d and the actual trajectory q we have. For different trials we will also be normalizing the error. We will also try to incorporate the error from the link velocity and acceleration. Additionally, we could be also punishing any exceeding of the robot limits by adding a penalty to the reward. We will be varying the exact reward function and parameters across the different experiments, as we will later define in the implementation Section 6.2.

Now that we have defined the agent input and output and environment, we can start the learning process. As mentioned in the Definitions and Fundamentals section, Section 2.3.3, the SAC algorithm is, because of its performance, sample efficiency and build in exploration, best suited for our application and will be therefore used here for training.

Several training experiments will be conducted. After each one, the results will be evaluated, and the parameters will be adjusted for the next experiment. The results and exact parameter used will be discussed in the next sections.

5.3 Pose modification and interpolation

As we will latter discuss in the results section, the first concept we proposed did show some improvements, however it still had its short comings. For the next proposed method, we will try to remove the noise in trajectory, ensure $\mathbb{C}^4$ continuity and correlation between the pose q and its derivatives $\dot{q}$, $\ddot{q}$, $q^{(3)}$ and $q^{(4)}$ by inserting an interpolator between the modified trajectory points.

First, we need to do a few modifications to the old pipeline. We will redefine the agents output and environment action space, instead of using an offset vector for the pose and its derivatives we will only use the pose offset. The environment will scale it as needed and add it to the desired pose q_d to make the new modified desired pose q_{mod} and publish it through LN.

The pose q_{mod} will be used by the interpolator as well as the current robot state q, $\dot{q}$, $\ddot{q}$, $q^{(3)}$ and $q^{(4)}$, to generate a continues trajectory.

The remaining of the pipeline and the reward function remain unchanged. The Figure 11 shows how the training pipeline changed compared to the one in Figure 10 and how the interpolator integrates with the rest of the pipeline.

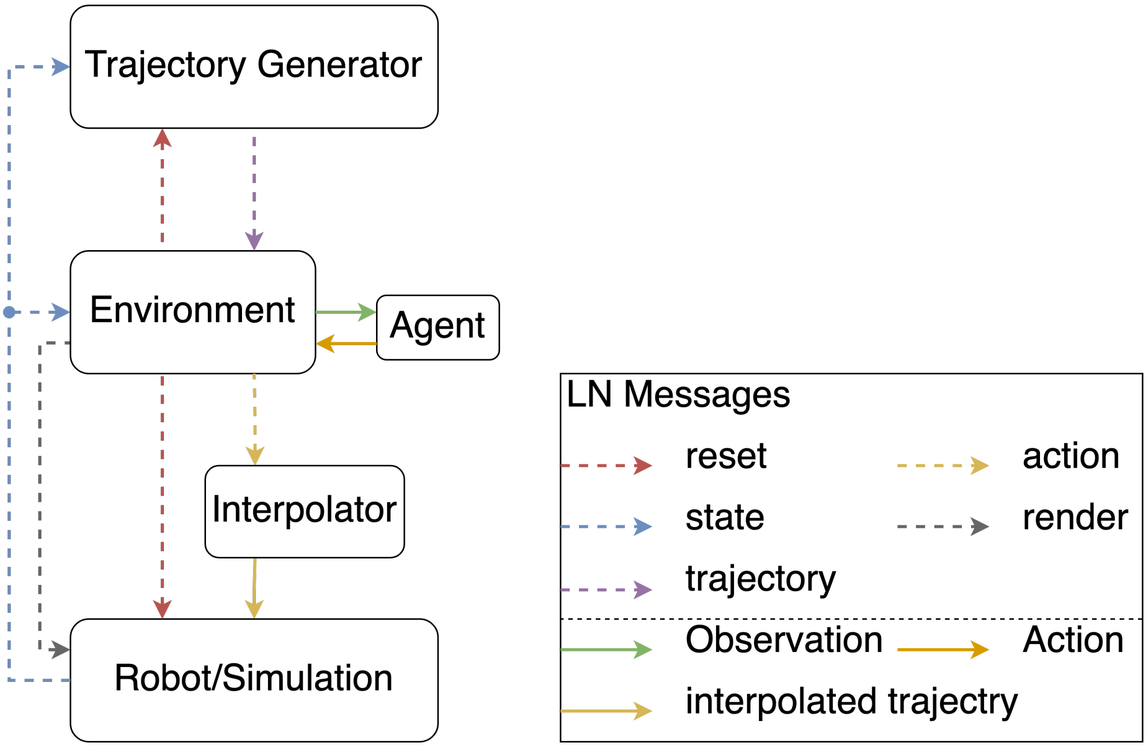

Figure 11: Agent trajectory modification and interpolation concept pipeline.

6 Implementation

In this section we show how we implemented the concepts and pipelines defined in the last section. We also define the reference and training experiments we ran.

6.1 Baseline and reference

We first start by building the base components needed for our application, then we show how they work together building a complete loop to test the baseline performance of the current controller concept. For this, two main components are needed a trajectory generator and the robot.

The trajectory generator is the component that defines the desired trajectory that the controller must track. In a production environment this would be used for example to dictate the trajectory of a pick and place task or to grasp an object seen by an advanced vision tracking system. For our testing purposes we will be generating a trajectory from a given time dependent function in the joint space. For ease of implementation, we chose to build this component using Python. To allow testing of multiple trajectories during our experiments, a fully modular script was developed that can generate the desired trajectory for q, $\dot{q}$, $\ddot{q}$, $q^{(3)}$ and $q^{(4)}$ automatically from multiple segments defined in a configuration file. The script allows to generate a single- or multi-DoF trajectory. The user should however still manually check the feasibility and the continuity over multiple segments of the trajectory.

Next, we need a robot for testing. We will first start by a using a simulated model and latter move to the real hardware when and if we were confident enough with our results. To simplify the problem and prove our optimization concept we start by using a single-DoF model. To implement the simulation, we used the model described in Section 2.1.1. We did also use the friction model from Section 2.1.2 both on the link and motor side of the elastic joint to better understand the impact of the different kinds of friction on the tracking ability of the ESP/ESπ controller. An existing simulation for the multi-DoF model did already exist, we did however change the used friction model to also consider the Coulomb and Stribeck effects thus making it more accurate. The single-DoF model had to be created from scratch. Both simulations were implemented in Simulink®.

To connect the trajectory generator with the simulation we will be using Links and Nodes. The trajectory generator will publish at each time-step the next desired trajectory poses, composed of q and the first four derivatives as well as the current timestamp. The simulation would then listen to the publish trajectory and run the simulation. The Simulink® model would also be publishing the current state of the robot as well as the current simulation time. The generator would then listen to the published state and synchronize itself with the simulation time creating the closed loop shown in Figure 12.

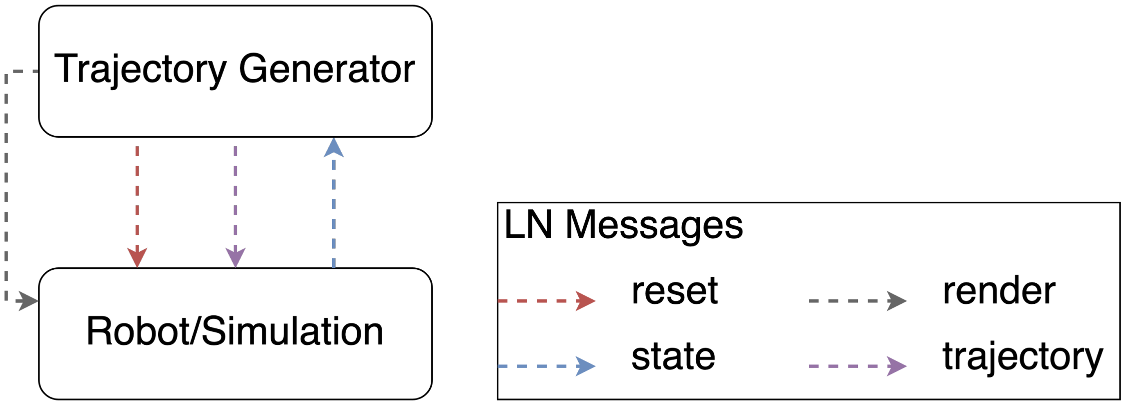

Figure 12: Traditional trajectory tracking loop.

Lastly, we need a way to visualize the movements of the robot. We will be saving the current position of the links at each time-step, allowing us to visualize the movements as well as to evaluate the system. For the single-DoF model, we will be using a 2D time plot. For the multi-DoF case, an existing URDF based visualizer will be used.
Now that we have setup everything, we can start our first experiments. To run a baseline and check for the effect of the different kinds of friction on the tracking error we will run multiple experiments varying the friction parameters. We start by only considering the dominant link-side friction and neglect the motor side friction. We varied the amount and type of friction from no friction to an unreasonably high amount and check its effect on the tracking error. We will have three parameters: the link side viscous friction kd_{fric}, Stribeck friction ks_{fric} and Coulomb friction ks_{fric} defined respectively through the slope, the peak, and the valley of the friction-velocity curve as marked in Figure 13.

We will choose kd_{fric} from the interval $[0.1, 1]$Nm.$s/°$, ks_{fric} from $[0.1, 10]$Nm and we will set $kc_{fric} = ks_{fric}/2$. We defined the experiment Group 1 with 15 experiments with different friction parameters. The experiments had to be repeated due a bug in the environment step function where the value for q_{mod} was accidentally written to all the derivatives. The bug was fixed, and the experiments were rerun as Group 4. The parameters for the experiments are defined in Table 1.

6.2 Direct trajectory modification

To modify the desired trajectory, we would need to add a few more components compared to the traditional tracking loop and solve some few extra challenges.

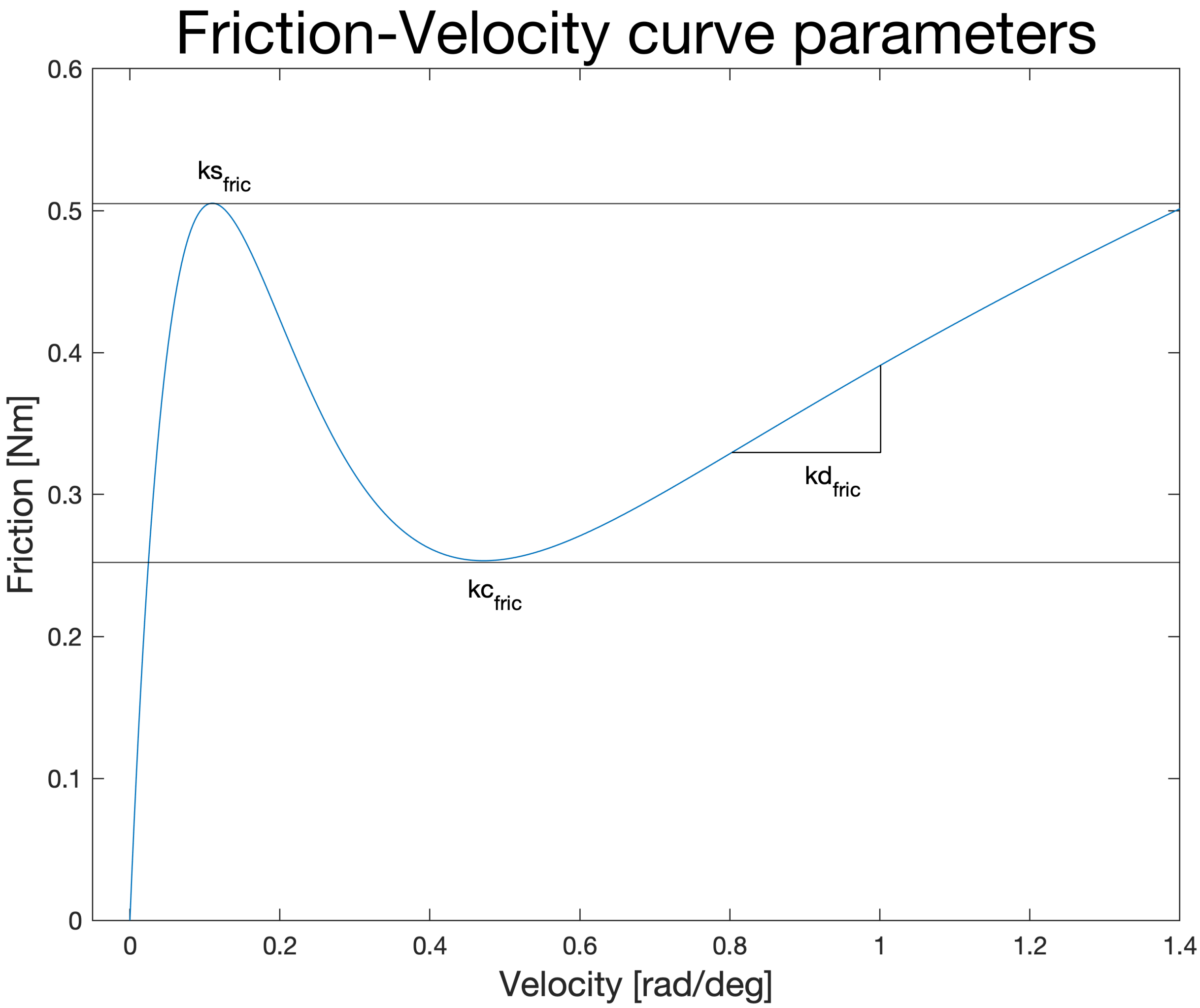

Figure 13: Friction-velocity curve parameters.

First, we will need the RL-Agent and the training loop. Since we will be using the SB3 [20] library for our agent, this part is made simple, we will use the build in agents and the predefined learning loop.

Next, we need to wrap the simulation or later the real robot hardware interface in a Gym [18] environment. The environment will be responsible to modify the desired trajectory base of the agent action and then publish it through LN. It will also be responsible to read the published robot state and creation an observation vector as well as calculating the current reward and deciding if a terminating state is reached. Since the length of the trajectory is variable and since we do not give the agent any information about the remaining time, we need to always return `info["TimeLimit.truncated"] = True` to indicate having an infinite loop problem and therefore not breaking Markov's property. This flag will be used by the agent to not limit the expected reward.

To keep the agent, environment and trajectory generator synchronized we needed a synchronization concept. For that we need to differentiate between simulation and real hardware. Due

Experiment-Id	kd_{fric}	ks_{fric}	kc_{fric}
0002 | 0020	0	0	0
0003 | 0021	0.1	0.1	0.05
0004 | 0022	0.2	0.1	0.05
0005 | 0023	0.1	0.2	0.1
0006 | 0024	0.2	0.2	0.1
0007 | 0025	0.2	0.5	0.25
0008 | 0026	0.5	0.5	0.25
0009 | 0027	0.2	1	0.5
0010 | 0028	0.5	1	0.5
0011 | 0029	1	1	0.5
0012 | 0030	0.5	2	1
0013 | 0031	1	2	1
0014 | 0032	0.5	5	2.5
0015 | 0033	1	5	2.5
0016 | 0034	1	10	5

Table 1: Experiment parameters for Group 1 and Group 4.

to lack of real-time constraints in the simulation and the difference in the stepping frequency between simulation and trajectory and agent, we need to stop the simulation, wait for the action given by the agent before continuing. Since there is no default way to pause the simulation in Simulink® we had to embed the simulation inside an enabled subsystem with an external trigger. For the trigger signal we first tried using a Stateflow® model that was driving the next simulation step. This proved to be unreliable in praxis leading in various stalls caused by numerical and rounding errors and difference in frequencies between the simulation and agent. Therefore, a more robust and reliable concept had to be found. For our second attempt we had the trajectory generator drive the system. After giving the first unmodified trajectory step, the agent reacts to the change of the published timestamp, modifying the trajectory and republishing it. The simulation would also detect the change in the timestamp and run for a predefined time interval before stopping and triggering the next step. This concept while being more complex at first did simplify the overall model and made it much more reliable for our training loop. On the other hand, we have the advantage of running on a real-time PC for real hardware testing. Due to time limitations, we did not manage to do any hardware testing and therefore we do not yet have any hardware synchronization concept.

Before starting with the experiments, we need to find the working range and joint limits of the robot. For that we defined the experiment Group 0 and latter repeat it as Group 3. Here we turned off all friction and varied the trajectory until the range and limits were found. We took the known limit parameters for the single-DoF joint to be the same as the first joint of the real robot.

After finding the range and limits of the robot we can chose the agent freedom we will be using. For q we will allow an offset of $\pm 5°$ and for the remaining derivatives we will start by allowing an offset off $\pm 10\%$ of the total range for those derivatives. In latter experiments these freedoms will be seen as training parameters and adjusted to improve the tracking error. We would still define them as a relative percentage of the total range.

Now that we have built the pipeline, we can validate it by running our first training experiment. We first started by using the parameters from Experiment 0007. We defined our first reward function with (33), with ω as a weight parameter and $q_{error} = q_d - q$ the tracking error. Here we take $\omega = -1$.

$$\omega \cdot \sum q_{error}^2 \tag{33}$$

Now we could start to make our first learning steps. In the next experiment group (Group 2) we ran two experiments. We did also normed the error and added a few terms to the reward function to punish any violation of the robot joint limits, resulting in the following reward function (34). With ω_i as a reward term for each of the robot limits. This time we chose $\omega = 1$ for the error term. The other parameters chosen for the experiments in Group 2 can be found in Table 2.

$$\omega \cdot e^{-\sum q_{error}^2} + \sum \omega_i \tag{34}$$

Experiments-Id	max spring deflection weight	max motor angular velocity weight	min max joint limit weight	max control input weight
0017	-0.5	-0.5	-0.5	-0.5
0018	-1	-1	-1	-1

Table 2: Experiment parameters for Group 2.

While running these experiments we noticed an unexpected tracking behaviour, which lead to the finding of the bug in the environment step function we mentioned before. After fixing the issue the baseline and range finding experiments had to be repeated.

Now that we had the major issues fixed, we could continue with our training. For the next experiment group (Group 5) we try to find good reward parameters and see how to improve and optimize learning. Building on experiment Group 2, we try changing the reward terms for exceeding the robot limits. We also try changing the leaning rate from the fixed default value to a time-based decay value. At the end we try to improve the training speed by switching from SB3 to Stable-Baselines Jax (SBX)[22], a Stable-Baselines3[20] implementation with a faster Jax backend and see if the learning speed improves. This results in a total of 9 experiments described in Table 3. To simplify the learning, the agent freedom was limited to only q starting with experiment 0038.

Experiments-Id	max spring deflection weight	max motor angular velocity weight	min max joint limit weight	max control input weight	Learning rate	Implementation
0035	-0.5	-0.5	-0.5	-0.5	default	SB3
0036	-1	-1	-1	-1	default	SB3
0037	-2	-2	-2	-2	default	SB3
0038	-0.5	-0.5	-0.5	-0.5	time based decay $lr0 = 0.001$ $decay = 0.1$	SB3
0039	-1	-1	-1	-1	time based decay $lr0 = 0.001$ $decay = 0.1$	SB3
0040	-1	-1	-1	-1	time based decay $lr0 = 0.001$ $decay = 0.1$	SB3
0041	-0.5	-0.5	-0.5	-0.5	default	SBX
0042	-1	-1	-1	-1	default	SBX
0043	-2	-2	-2	-2	default	SBX

Table 3: Experiment parameters for Group 5.

Next to these experiments we also worked on resolving some remaining issues and optimizing the training pipeline. One major problem we had was a memory leak in the simulation that led to crashes or forced early terminations of the experiments. The issue was caused by some default Simulink® configuration that logged some additional data for debugging such as the time vector, which lead to the consumption of more and more memory over time. Additionally,

some extra scripts were built to automatically lunch multiple training instances simultaneously. This was especially challenging because we needed to generate new configuration files for LN and Matlab®/Simulink® and run multiple LN and Matlab® instances simultaneously without any conflicts.
Because of the slow learning we saw in Group 5 we decided to better normalize the error term in the reward function (35) for the experiments in Group 6. We can think of the σ parameters as the standard deviation off the expected error during training. We defined 7 experiments in this group with parameters in Table 4.

$$\omega \cdot e^{-\frac{\sum q_{error}^2}{2\cdot\sigma^2}} + \sum \omega_i \tag{35}$$

Experiments-Id	sigma	Remark
0044	0.025	$\times 0.1$ from bevor sigma was introduced
0045	0.0087	$0.5^{\circ/s} \rightarrow \text{rad}/s$
0046	0.0029	$0.5^{\circ/s}/3 \rightarrow \text{rad}/s$
0047	0.00056	$0.1^{\circ/s}/3 \rightarrow \text{rad}/s$
0048	0.0003	$0.05^{\circ/s}/3 \rightarrow \text{rad}/s$
0049	0.002	Small deviation from 0046
0050	0.001	Small deviation from 0046

Table 4: Experiment parameters for Group 6.

While barely noticeable in the trajectory curve, we can clearly see an oscillation problem in the velocity curve in Figure 14. This was due to a faulty calculation of the initial conditions of the link in the simulation and was fixed for experiments after 0046 the oscillations problem was fixed.

After limiting the agent's freedom in Experiment 0038 we want to check its effect on the training's performance. Therefore, in the experiment Group 7 we run 6 experiments giving the agents different amount of freedom as defined in Table 5. The reward function was unchanged compared to the previous experiment group, and the same σ value as Experiment 0046 was used.

As we will latter discuss in the results section the experiments in Group 7 showed improvements in the q tracking error by heavily modifying $\dot{q}$ and $\ddot{q}$. To improve that, we incorporated the error of $\dot{q}$ and $\ddot{q}$ in the reward function (36). Similarly, to previous experiments, for the experiment Group 8 we try changing the σ values for $\dot{q}$ and $\ddot{q}$ as well as the agent freedom across 6 different

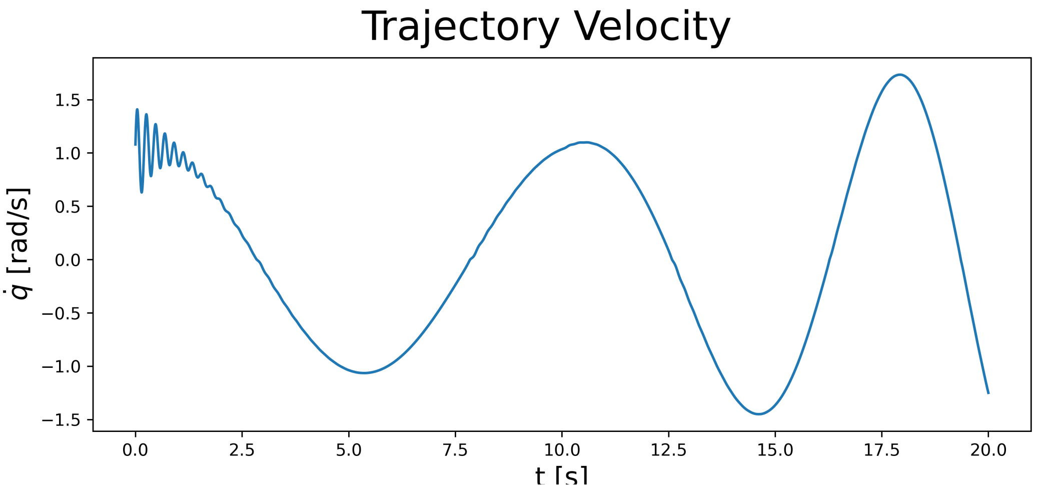

Figure 14: Trajectory oscillations as seen in the velocity curve.

Experiments-Id	Agent $\dot{q}$ freedom	Agent $\ddot{q}$ freedom	Agent $q^{(3)}$ freedom	Agent $q^{(4)}$ freedom	Remark
0051	± 12	± 97	0	0	1% of max range
0052	± 60	± 485	0	0	5% of max range
0053	± 120	± 970	0	0	10% of max range
0054	± 12	± 97	± 12.5	± 6	1% of max range
0055	± 60	± 485	± 62.5	± 30	5% of max range
0056	± 120	± 970	± 125	± 60	10% of max range

Table 5: Experiment parameters for Group 7.

experiments defined in 6. Additionally, the weight for each of the error terms was defined in such a way that the sum of all three of them is 1. This should allow to keep the reward value from the errors normed between 0 and 1 as it was before. Based on the relative importance of each error term on the performance we want to achieve, we choose a weight of 0.8, 0.15 and 0.05 for ω_{q_d}, $\omega_{\dot{q}_d}$ and $\omega_{\ddot{q}_d}$ respectively. We will also be limiting the agent freedom to only modifying q_d, $\dot{q}_d$ and $\ddot{q}_d$.

$$\omega_q \cdot e^{-\frac{\sum q_{error}^2}{2\cdot\sigma_q^2}} + \omega_{\dot{q}} \cdot e^{-\frac{\sum \dot{q}_{error}^2}{2\cdot\sigma_{\dot{q}}^2}} + \omega_{\ddot{q}} \cdot e^{-\frac{\sum \ddot{q}_{error}^2}{2\cdot\sigma_{\ddot{q}}^2}} + \sum \omega_i \tag{36}$$

So far, we only gave our agent the desired trajectory poses of the next step. In the experiment Group 9 we will try giving the agent the pose of several upcoming time steps. We will also check if allowing the agent to also modify $q^{(3)}$ and $q^{(4)}$ would improve the tracking performance. Overall, we defined 6 additional experiments in Table 7. The aim here would be to further

Experiments-Id	Relative agent freedom	$\sigma_{\dot{q}}$	Remark	$\sigma_{\ddot{q}}$	Remark
0057	$\pm 1\%$	0.0058	$1^{\circ/s}/3 \rightarrow \text{rad}/s$	0.029	$5^{\circ/s^2}/3 \rightarrow \text{rad}/s^2$
0058	$\pm 5\%$	0.0058	$1^{\circ/s}/3 \rightarrow \text{rad}/s$	0.029	$5^{\circ/s^2}/3 \rightarrow \text{rad}/s^2$
0059	$\pm 10\%$	0.0058	$1^{\circ/s}/3 \rightarrow \text{rad}/s$	0.029	$5^{\circ/s^2}/3 \rightarrow \text{rad}/s^2$
0060	$\pm 1\%$	0.0029	$0.5^{\circ/s}/3 \rightarrow \text{rad}/s$	0.0029	$0.5^{\circ/s^2}/3 \rightarrow \text{rad}/s^2$
0061	$\pm 5\%$	0.0029	$0.5^{\circ/s}/3 \rightarrow \text{rad}/s$	0.0029	$0.5^{\circ/s^2}/3 \rightarrow \text{rad}/s^2$
0062	$\pm 10\%$	0.0029	$0.5^{\circ/s}/3 \rightarrow \text{rad}/s$	0.0029	$0.5^{\circ/s^2}/3 \rightarrow \text{rad}/s^2$

Table 6: Experiment parameters for Group 8.

decrease the error of q, $\dot{q}$ and $\ddot{q}$, while also reducing the noise around $\ddot{q}$.

Experiments-Id	Relative agent freedom $\dot{q}$ and $\ddot{q}$	Relative agent freedom $q^{(3)}$ and $q^{(4)}$	Desired trajectory sequence length
0063	$\pm 1\%$	0	3
0064	$\pm 5\%$	0	3
0065	$\pm 1\%$	0	5
0066	$\pm 5\%$	0	5
0067	$\pm 5\%$	$\pm 1\%$	5
0068	$\pm 5\%$	$\pm 1\%$	3

Table 7: Experiment parameters for Group 9.

Building on the results from the last group, we settle on giving the agent the next 3 steps. To further reduce the pose error, we decide on changing the weight distribution of the reward function to 0.9, 0.075 and 0.025 for ω_q, $\omega_{\dot{q}}$ and $\omega_{\ddot{q}}$ respectively. For the 6 experiments of Group 10 we also will be changing the parameters for the agent's freedom as defined in Table 8.

Experiments-Id	Relative agent freedom $\dot{q}$ and $\ddot{q}$	Relative agent freedom $q^{(3)}$ and $q^{(4)}$
0069	$\pm 0.5\%$	$\pm 0.5\%$
0070	$\pm 1\%$	$\pm 0.5\%$
0071	$\pm 1\%$	$\pm 1\%$
0072	$\pm 1\%$	$\pm 5\%$
0073	$\pm 1\%$	$\pm 10\%$

Table 8: Experiment parameters for Group 10.

6.3 Pose modification and Interpolation

The base implementation remains like before. We first change the action given by the agent and environment to only modifying the pose q of the trajectory. Second, we add the interpolator to the pipeline inside the Simulink® component in front of the controller. After the interpolation block, we would have the remaining derivatives of the trajectory as well as the smoothed correlated transition to the next pose, which we will pass to the controller.
The goal of the interpolator is to bridge the gap between two time-steps q_s and q_e with a smooth continues trajectory as shown in Figure 15. To implement the interpolator, we need to use an interpolation function. The obvious solution is to use a polynomial function. The goal here would be to ensure the continuity between the interpolated steps, the correlation between the pose and derivatives offsets and the reach of the given desired pose. Therefore, we need to impose a few side conditions. For an interpolation step between time-steps t_s and t_e we need to unsure the following:

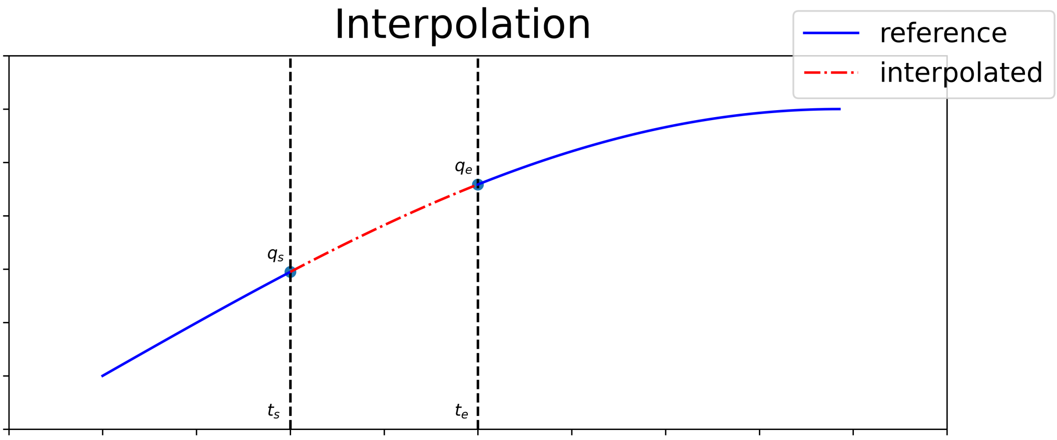

Figure 15: Interpolation problem.

$$q(t = t_s) = q_s \tag{37}$$

$$\dot{q}(t = t_s) = \dot{q}_s \tag{38}$$

$$\ddot{q}(t = t_s) = \ddot{q}_s \tag{39}$$

$$q^{(3)}(t = t_s) = q_s^{(3)} \tag{40}$$

$$q^{(4)}(t = t_s) = q_s^{(4)} \tag{41}$$

$$q(t = t_e) = q_e \tag{42}$$

$$\tag{43}$$

Where the constants being the respective desired trajectory poses at the start or end of the interpolation interval. Having six conditions we would need at least six parameters in the interpolation function. We first try to use a simple 5-degree polynomial (44).

$$q(t) = a + b \cdot t + c \cdot t^2 + d \cdot t^3 + e \cdot t^4 + f \cdot t^5 \tag{44}$$

To keep the output values in check and simplify the problem we norm the interpolation interval between 0 and $t_e =$ interval length. We then use the conditions to solve for the coefficients of (44) and create the interpolation function. We also need to build a mechanism to hold the constants for the duration of the interval to feed that function.

Running a few tests, we realize that the interpolated trajectory does deviate a lot in the mild from the expected trajectory. This results in a very high acceleration of the motor as well a high control input.

In our next attempt we try to use different polynomial function that would minimize that problem. As we see it, we could circumvent this problem by ensuring that the trajectory does not poses any turning points inside the interpolation interval. We will be using a polynomial function with only odd exponent terms. If all coefficients had the same sign, we would have only one point where the derivatives are zero and therefore, we would only have one turning point at $t = 0$. This would however result in an unsolvable system. Therefore, we decide to move this turning point to the other end of the interval t_e, ensuring it remains at one edge of the interval minimizing the deviation. Now that we are using only odd terms, we need to use a 9th-degree polynomial (45), with a translation along the X and Y-axis.

$$q(t) = b + c \cdot (t - a) + d \cdot (t - a)^2 + e \cdot (t - a)^3 + f \cdot (t - a)^4 + g \cdot (t - a)^5 \tag{45}$$

With $a = t_e$ set the position of the turning point. As is, solving for the remaining coefficients will not ensure that they will all have the same sign. Nevertheless, running a few tests show that the interpolator shows an almost perfect trajectory for q. However, the velocity curve shows some spikes which results in high values for the acceleration, higher derivatives, and control input. While the results show lower values, they still exceed the robot limits.

For a final attempt we chose to interpolate the offset instead of interpolating the trajectory. We would interpolate the q offset Δq given by the agent to between the single steps and use the interpolated values to calculate the required offset of the derivatives. We would then add the offsets to the desired trajectory (46), (47), (48), (49) and (50) and the give it to the controller.

$$q_{mod}(t) = q(t) + \Delta q \tag{46}$$

$$\dot{q}_{mod}(t) = \dot{q}(t) + \Delta \dot{q} \tag{47}$$

$$\ddot{q}_{mod}(t) = \ddot{q}(t) + \Delta \ddot{q} \tag{48}$$

$$q^{(3)}_{mod}(t) = q^{(3)}(t) + \Delta q^{(3)} \tag{49}$$

$$q^{(4)}_{mod}(t) = q^{(4)}(t) + \Delta q^{(4)} \tag{50}$$

To calculate the offsets of the higher derivatives we integrate (50) four times, compare the results with equations (46), (47), (48) and (49) and solve for the offsets as follow:

$$\Delta \dot{q} = 4 \cdot \frac{\Delta q}{dt} \tag{51}$$

$$\Delta \ddot{q} = 12 \cdot \frac{\Delta q}{dt^2} \tag{52}$$

$$\Delta q^{(3)} = 24 \cdot \frac{\Delta q}{dt^3} \tag{53}$$

$$\Delta q^{(4)} = 24 \cdot \frac{\Delta q}{dt^4} \tag{54}$$

This change in the approach should keep the deviation from the expected trajectory given by the interpolator minimal and controllable by the given agent freedom Δq_{max}. While the first implementation tests seem to support this hypothesis, we did not have enough time to run any training experiments.

7 Evaluation of the Results

In this section we show the results of the baseline and training experiments we ran.

7.1 Baseline and reference

We started by running the friction experiments defined in Table 1. The results of the first experiments group (Group 1) will be dismissed because of the bug. The results from experiment Group 4 would now be evaluated. Analysing the recorded trajectory data in a Jupyter notebook we calculate the mean and maximal square error of the link position across the trajectory. As seen in Figure 16 both the root mean and maximal squared root error increase with increasing friction. We can also infer that ks_{fric} and kc_{fric} have a slightly higher effect than kd_{fric} on the tracking error.

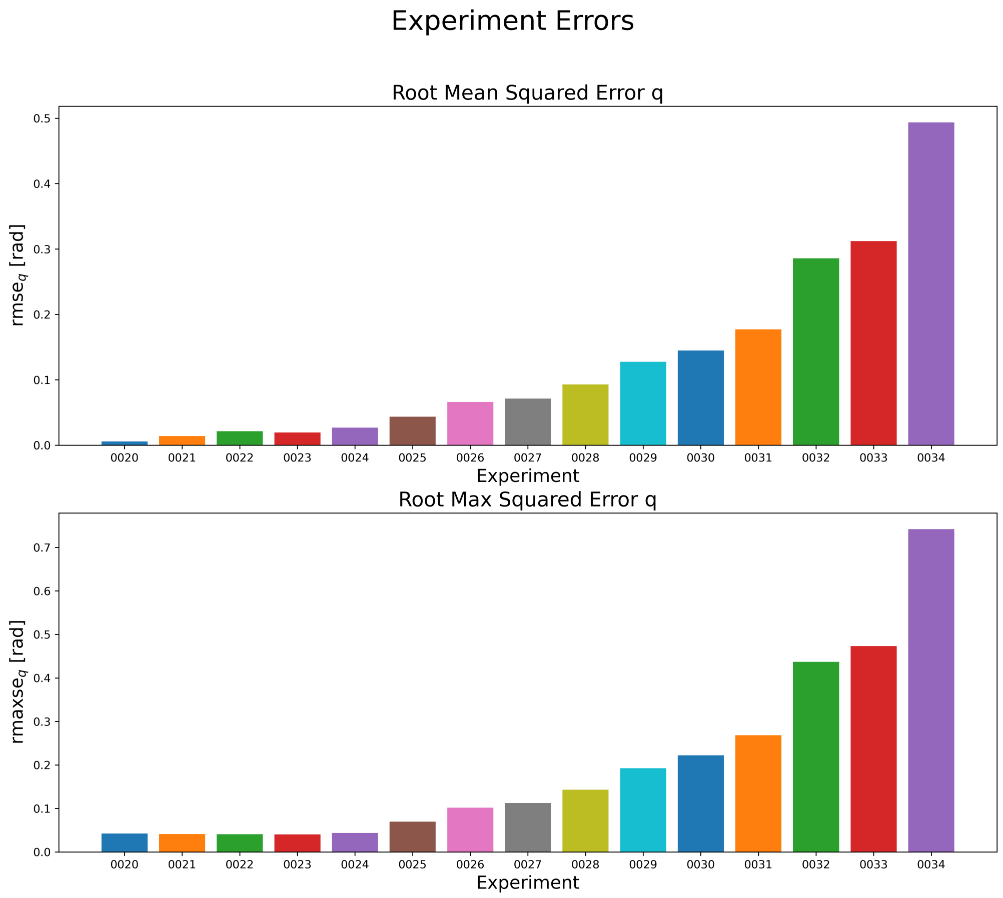

Figure 16: Root mean and maximal squared tracking error for experiments Group 4.

The root mean squared error over the trajectory of for the experiments from Group 4 ranged between $0.0059[\mathrm{rad}]$ and $0.4936[\mathrm{rad}]$ and $0.0436[\mathrm{rad}]$ for experiment 0025.

The friction parameter from experiment 0025 are the most realistic for our application and will be latter used as a reference and baseline trajectory to evaluate the improvements made in the next experiments. The baseline run will be latter repeated as part of experiment 0048 to remove the error caused by the faulty initialization and used to evaluate the experiments starting 0046. Due to time constraints, we did not repeat the experiments with the other friction parameters. The results should however show the same effect of friction with minimal difference in the mean squared error values.

7.2 Direct trajectory modification

We will start with the results from the range finding experiment 0019 from Group 3. After several adjustments to the trajectory, we found one that would exceed all the known limits. As seen in Figure 17, some of the limits are reached at the start of the trajectory and others at the end. We mark each time-steps where the maximal deflection, control input and motor velocity were exceeded. We also show the maximal value reached in the interval until all three reference limits were reached. Due to the error in the initialization of the initial condition we can see a slight oscillation in trajectory at the beginning of the trajectory. Since, in none of the cases, the final range was set or exceeded due to these oscillations, the results we have remain valid and the experiment was not repeated.

Now that we have the robot model limits defined, we can start discussing the training experiments. Each training experiment was run twice to ensure that all the results were reproducible. The first three experiments groups were run while the environment step function was still faulty and therefore their results will be dismissed. The episodes during the training experiments from these groups were very short and stopped after just a few time-steps.

After fixing the bug, we ran the remainder of the experiment groups. We start with the results of Group 5. We will group the results in 3 sections. First the experiments 0035 to 0037 showed a slight decrease in the error rate compared to the baseline. Only one of the two experiments 0036 and 0037 was run, this was due to some faulty configuration in the automated launch script. Therefore, we could not consider the results of the single experiments as credible. To increase the learning ability of the agent, we try using a variable learning rate with a time-based decay. The results of the experiments 0038 to 0040 showed however worse results than the group before. Even with a longer training time, the learning rate did not decrease fast enough for the agent to learn. The results of the experiment 0040 were accidentally deleted and therefore not presented in the Figure 18. For the last three experiments, we switched to the SBX implementation and repeated the training with a fixed learning rate. As expected, the results were like before, the error was comparable with the baseline. As seen in Figure 18 the root mean squared

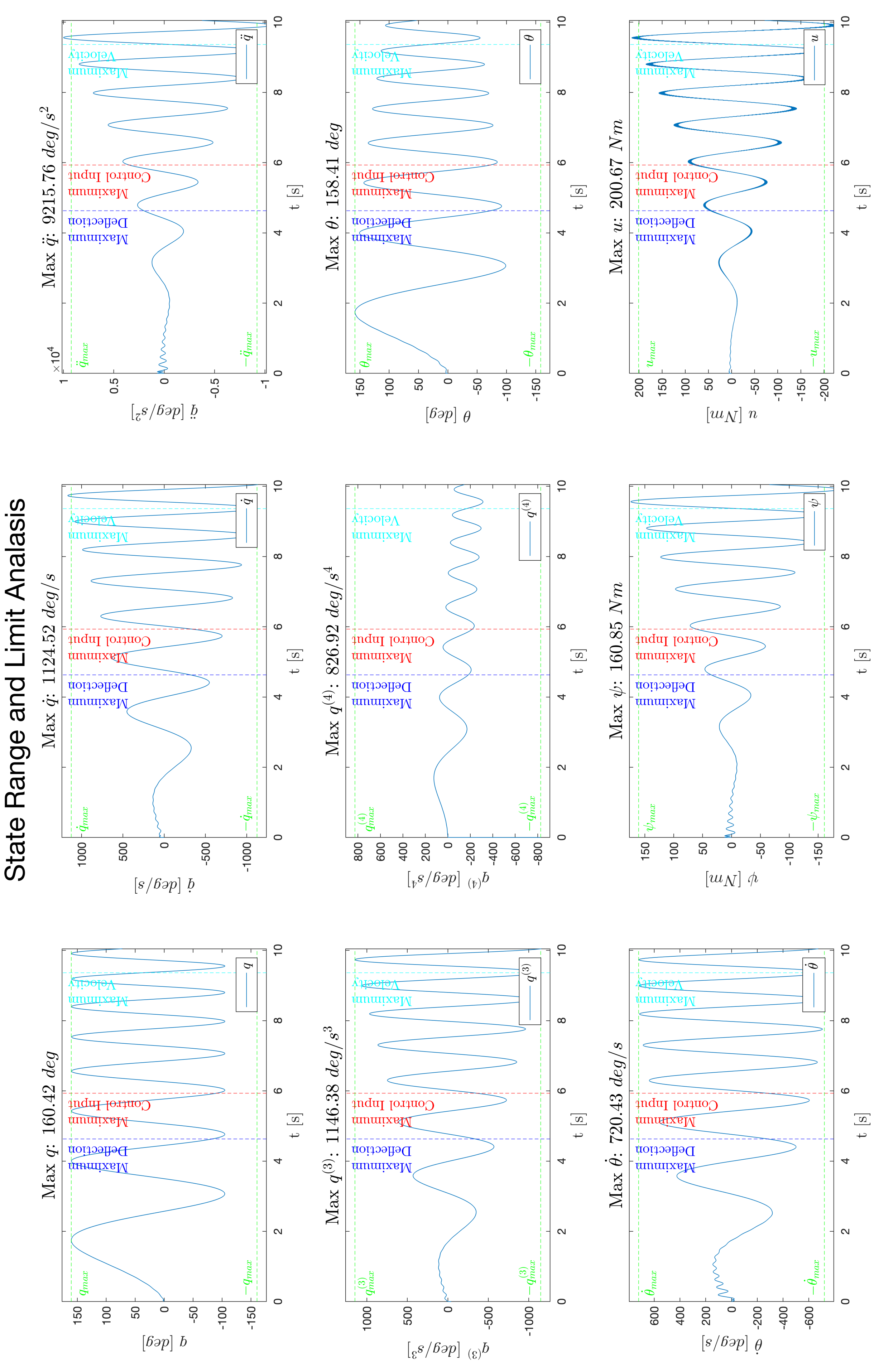

Figure 17: Range and limit analysis of single DoF model.

error of the pose q was very chaotic during training and ranged mostly between -150% and $+15\%$ at the end of the training compared to the baseline. This shows that the agent did have trouble learning and understanding in which direction to learn.

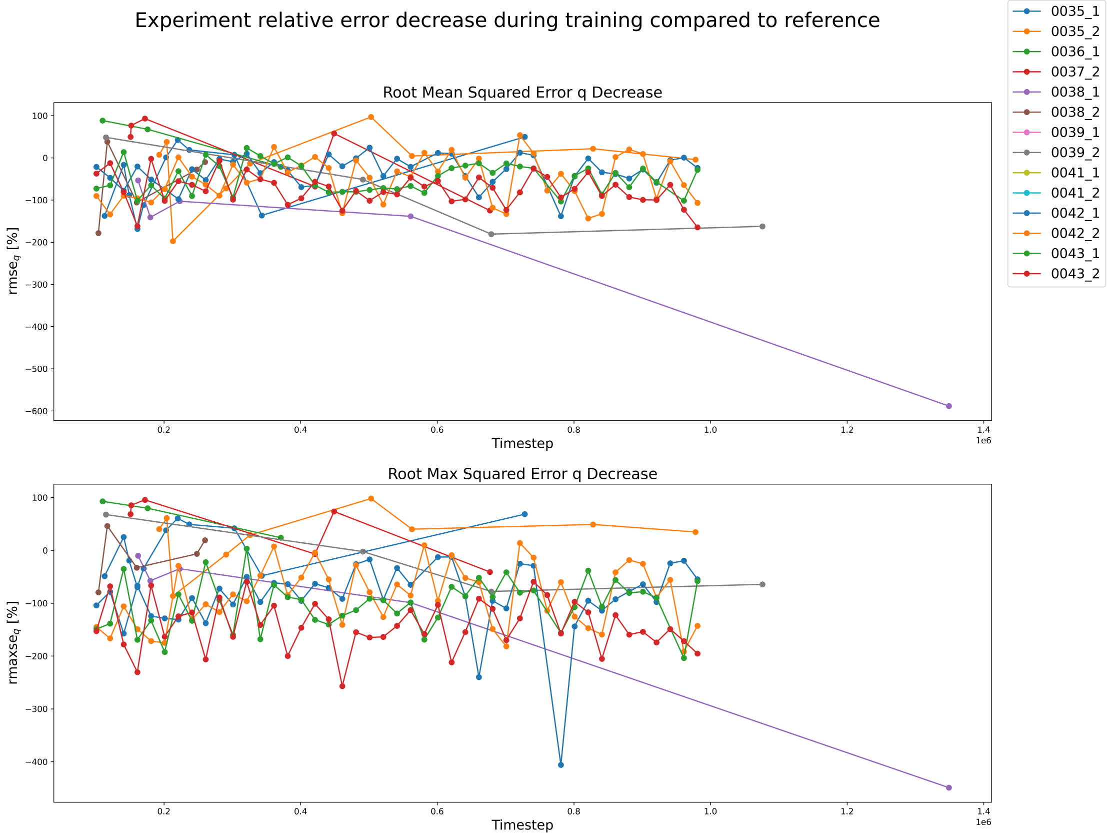

Figure 18: Group 5 experiment relative error decrease during training.

To help the agent with its learning, we decide to norm the error in the reward function. Running all the experiments with the different σ values we learn that setting the σ value too high or too low affects the learning performance will decrease. The experiments 0044 and 0045 had the highest σ value and showed almost no improvements. The two experiments still had the old reference trajectory from experiment 0025 with the oscillations at the start of the trajectory, for the other experiments a new baseline trajectory with the same friction parameters was run with the configuration file of experiment 0048. On the other extreme using a low σ value, like in experiments 0047 and 0048 showed a decrease in tracking performance. The remaining experiments showed an improvement of up-to 75% for the root mean squared error of q and 60% for the maximal root squared error of q. In Figure 19 we can clearly see how the tracking performance improved compared to the reference during training of Group 6. All experiments terminated early because the tracking performance stopped improving.

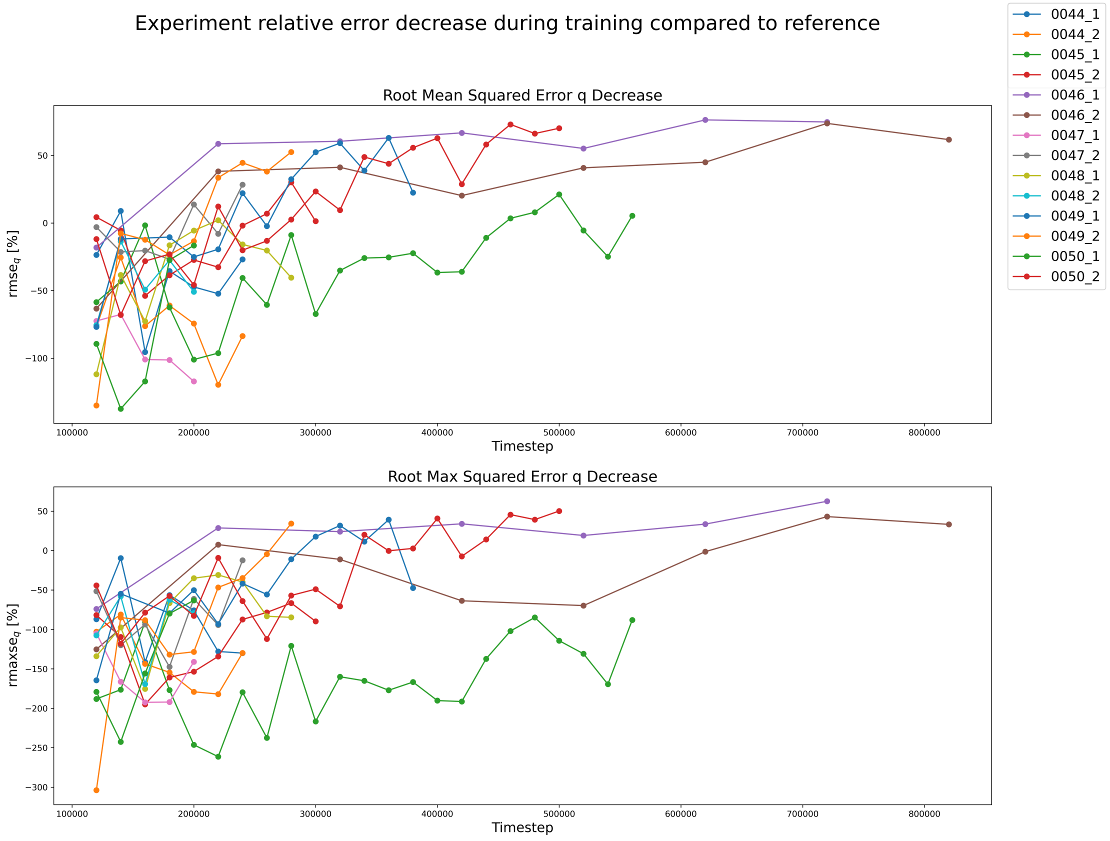

Figure 19: Group 6 experiment relative error decrease during training.

Both runs for experiment 0046, trained for the longest and showed the best results. In Figure 20 we see how the trajectory from both experiments 0046 overlaps almost perfectly with the reference, while the baseline trajectory deviates from it at the sin picks.

Now that we showed that we can achieve some improvements we need to optimize the training parameter. As mentioned in the implementation Section 6.2, we had the agent freedom limited to only changing the pose q. For the next experiment group (Group 7) we would like to explore the impact it has on the learning performance. Looking at the trajectory taken by the last agent of each experiment in Figure 21 we can denote two important things. First, the tracking error decreased below the baseline error and stayed minimal for the first half of the trajectory, but for the second trajectory half with the chirp signal the error kept increasing alongside the frequency and surpassing the reference error. Second, while the pose trajectory appears to be very smooth, the derivatives and control input appear to be very noisy. This is due to the uncorrelated offsets of the trajectory derivatives and pose given by the agent. The problem appears to be especially problematic when we increase the agent's freedom. For the experiments 0054 to 0056, with freedom for the higher derivatives, the deviation from the desired velocity and acceleration is

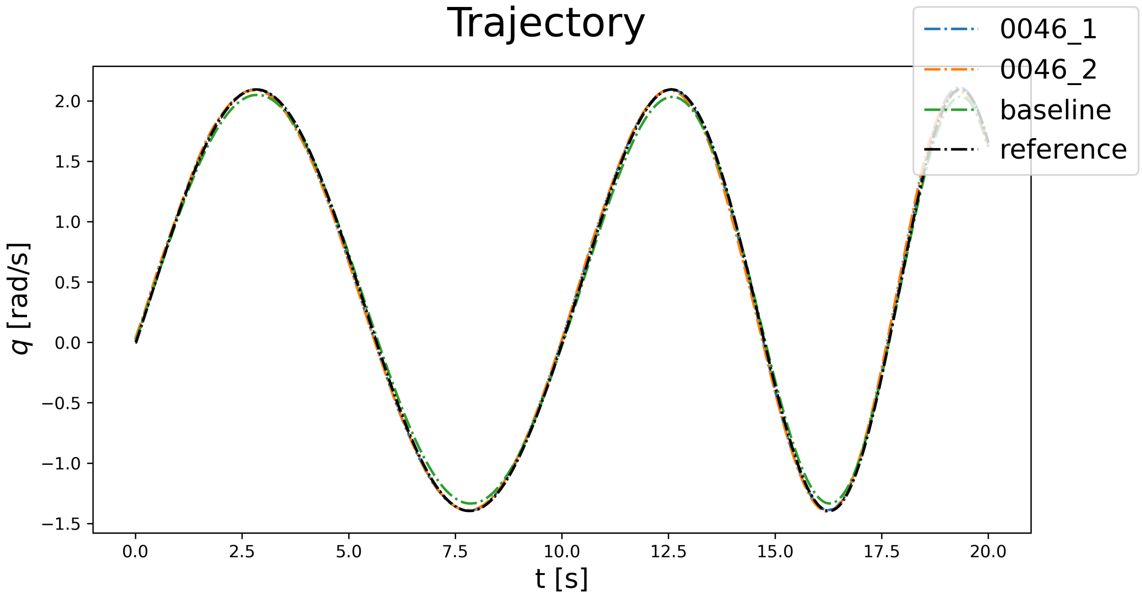

Figure 20: Trajectory of experiments 0046.

worse. One additional thing we need to point out, the trajectory of experiment 0056_1 is offset by $2.84s$. The reason of why this might be the case is unknown. However, this was the only experiment where we saw this behaviour. Overall, the results of this experiment group showed slight to no improvement in the mean error of q compared to experiment 0046.

We will try to solve those limitations one by one. We will first try to reduce the noise around the control input and the derivatives. In experiment Group 8 we incorporate the velocity and acceleration error in the reward function. The idea here would be to penalize big deviations from the desired values. The results show that the noise around the velocity and acceleration curves was reduced. The maximal error along the trajectory also decreased compared to experiment 0046. This however comes at the cost of the mean pose error, which increased. Figure 22 still shows the same frequency dependent error.

Second, we will try to reduce the frequency dependent error. To do so, we will try to give the agent more information about the upcoming trajectory. We will be giving the agent the next n upcoming steps of the desired trajectory, allowing the agent to see further in the future. Furthermore, we will change the weights of the single reward terms for the errors, giving back the pose error more weight. The hope here is to compensate for the worsening of the mean squared error of the pose we saw in Group 8. Running the experiments, we see that the mean squared error improved compared to the baseline, however the performance is worse compared to experiments 0046. Additionally, we see little to no improvements regarding the frequency dependent error. Looking at the relative mean squared error improvements compared to the

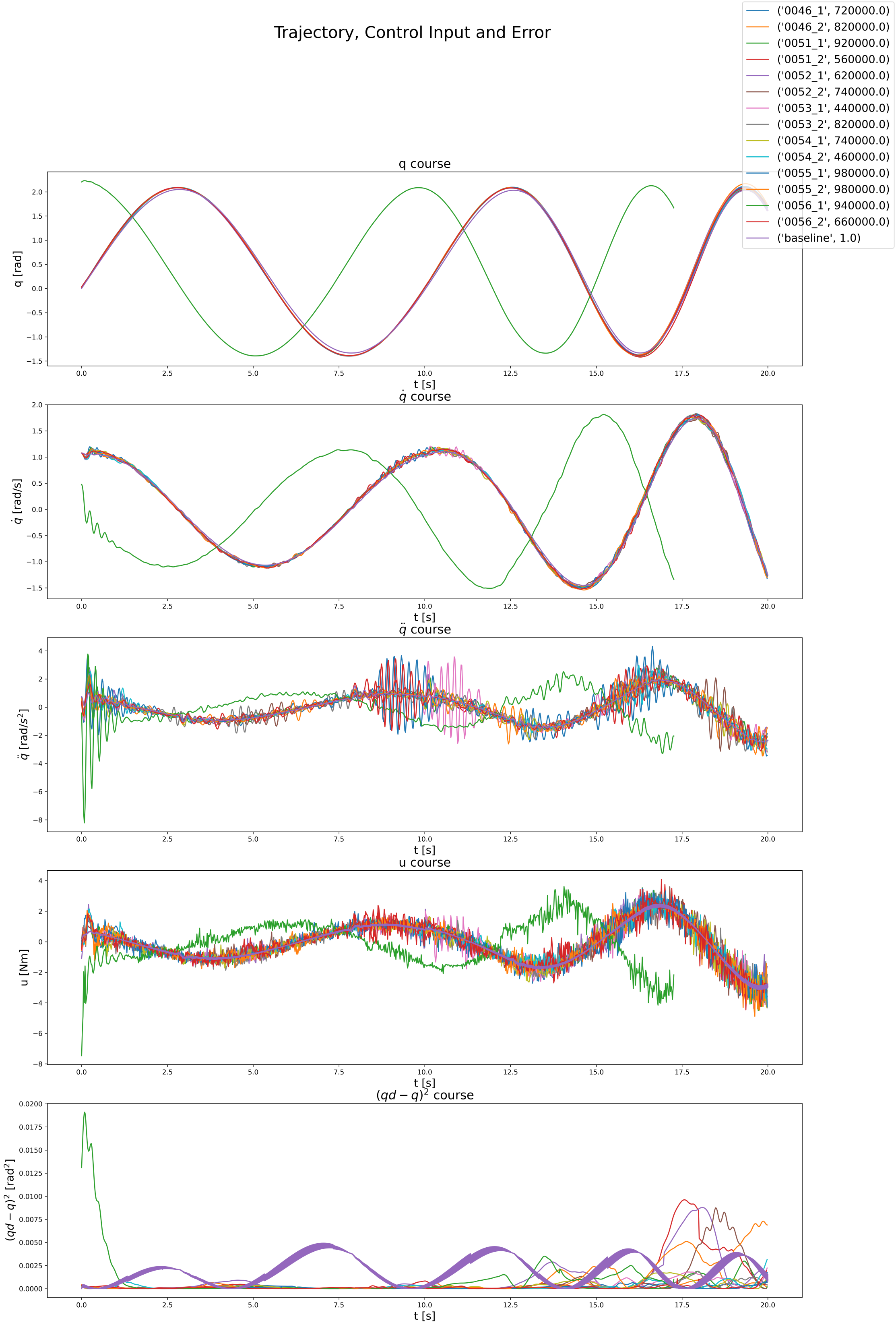

Figure 21: Trajectory, control input and error of the last episode in Group 7.

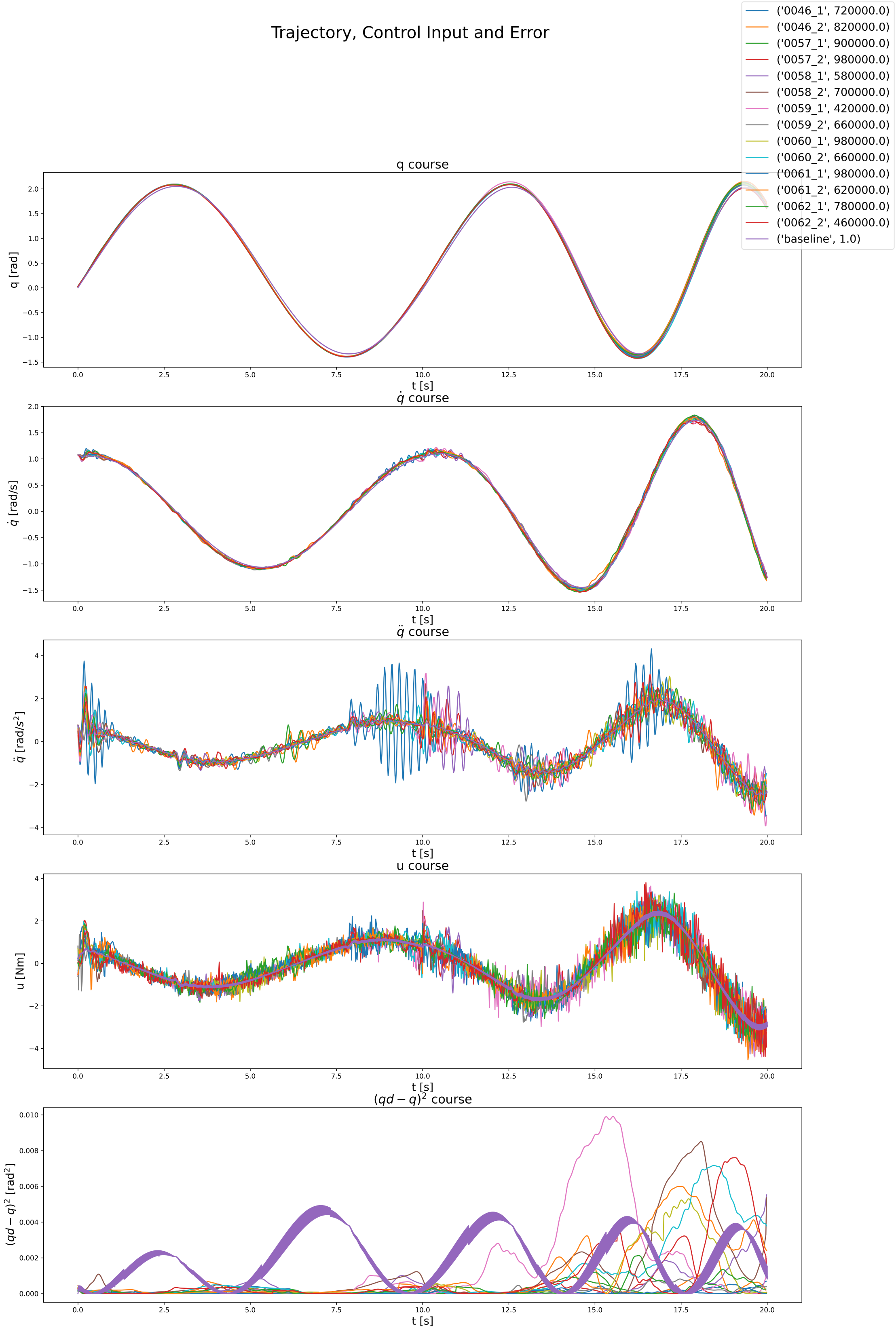

Figure 22: Trajectory, control input and error of the last episode in Group 8.

baseline in Figure 23, we see that for most experiments the training stopped very early suggesting that there might be room for improvements. The experiments 0067 and 0068 did train the longest and showed comparable results to 0046. This indicates a relationship with the agent freedom of the higher derivatives $q^{(3)}$ and $q^{(4)}$. The experiments with the smaller window 0063, 0064 and 0068 did show slightly better performance than their counterpart experiment with the bigger window size. This could be the indication that giving the agent too much information of the future increases the overall complexity of the system, slowing the learning down.

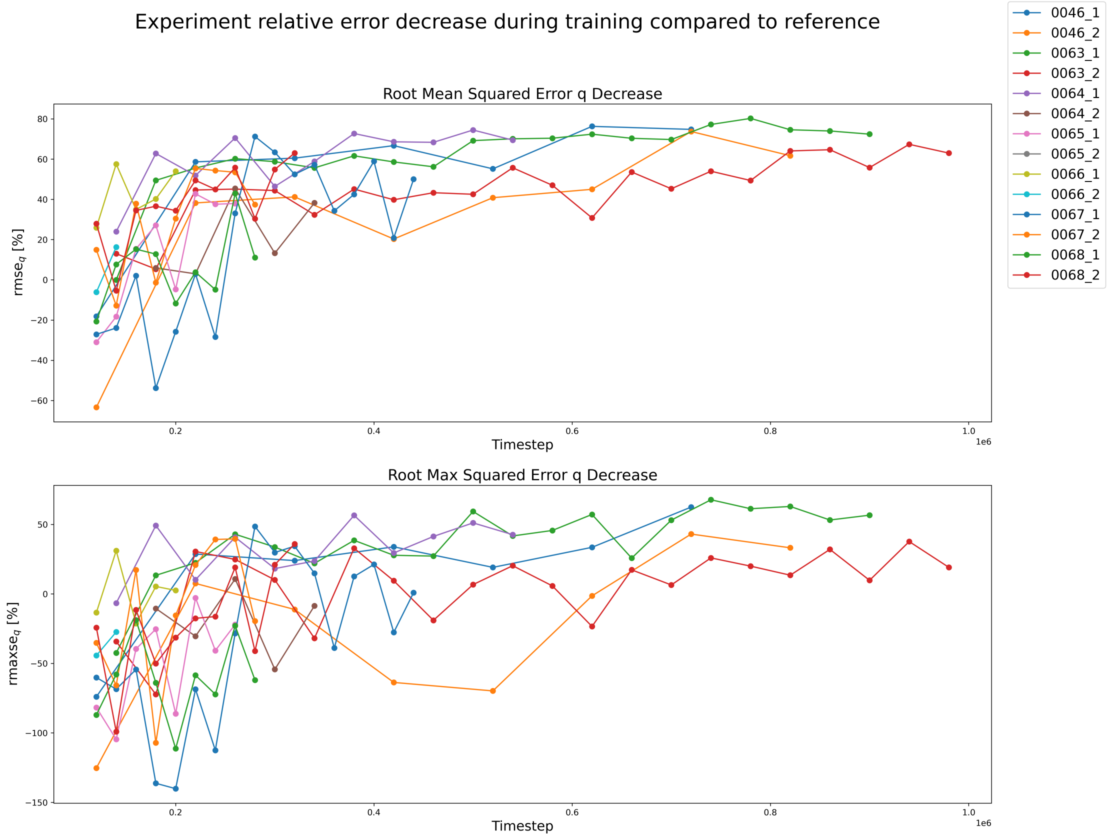

Figure 23: Group 9 experiment relative error decrease during training.

Building on these results, we would like to further inspect the effect of the agent freedom, especially of the higher derivatives $q^{(3)}$ and $q^{(4)}$, in experiment Group 10. The results were slightly inconstant, one run of each experiment terminated early while the other one trained until reaching or almost reaching the maximal episode count given. Experiments 0069 to 0072 had a training budget of 500000 steps while experiment 0073 did get double the budget. As a hole the experiments did show comparable results to experiment 0046. Looking at the trajectories of Group 10 in Figure 24 we see that the noise around the control input and especially around the acceleration is lower than experiment 0046. The frequency dependent error we see in the second half of the trajectory did get lower for some of the experiment trials, but the results are not conclusive.

The results seem promising however some further investigation is required to improve the performance even more. We also need to solve the issue with the high noise around trajectory, since this breaks the stability requirements of the controller. The experiment given the best overall consistent results remains 0046 with a root mean squared error of 0.01099[rad], a decrease of 74.78% compared to the baseline. Finding a balance between agent freedom and the number of future steps given seems promising. We would need to run more experiments while giving the agent more the ability to learn complex task and systems. One idea would be to adjust the complexity and structure of the agent's network and increase the number of learning parameters of the network.

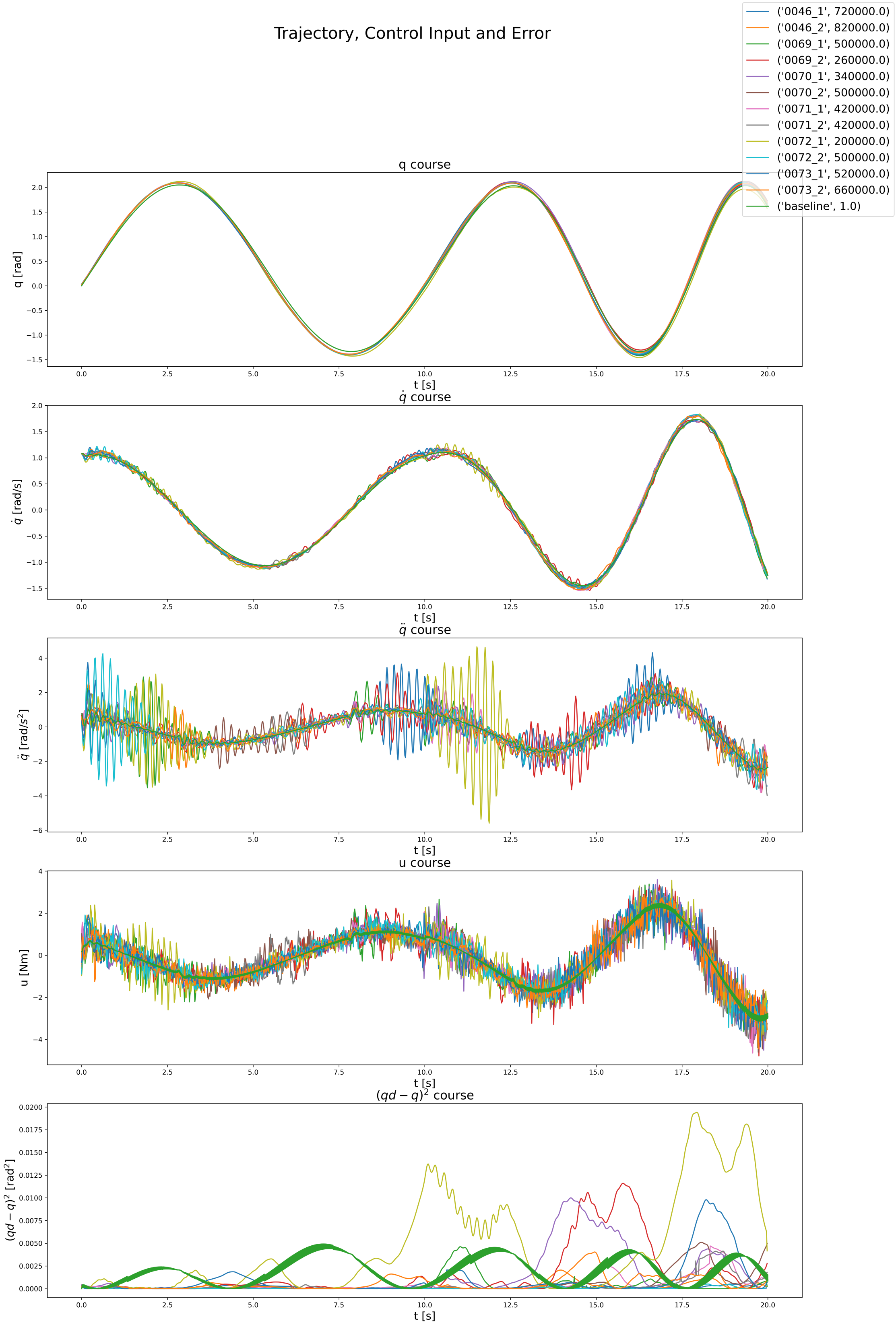

Figure 24: Trajectory, control input and error of the last episode in Group 10.

8 Summary and Outlook

For this thesis we planned on examining a RL approach to better increase the tracking ability of the current ESP/ESπ controller for David's elastic arm system. After building a simplified single-DoF simulation model of the system, we showed the impact friction has on the tracking performance. We did not however have the time to study the impact of the trajectory velocity on the tracking error. After building a working RL-pipeline, we managed to run several experiments that did show some improvements compared to the baseline trajectory. With our first concept we did manage to overcome the main limitations we discussed in the literature Section 3, like the limited range, the need to retrain for each new trajectory or trajectory range and especially the need to have intimate knowledge of the robotic system and or the process. The concept used a neural network-based agent to predict the necessary offsets for the trajectory pose and derivatives to minimize the error of the resulting trajectory compared to the desired one. This approach showed a lot of improvements and some promising results. However, the learning stalled, and we saw a lot of noise around the trajectory especially for the higher derivatives which broke the continuity assumption we need for the controller and the correlation between the pose and the derivatives. While we did not see any signs of instability, this limitation means that we cannot guaranty a stable system during operation especially not during stochastic training.

We did drop this approach in favour of the newer concept with an interpolator. This newer approach should guaranty stability, while still showing most if not all the advantages we got with the first concept. The difficulty in this approach is in building a stable interpolator that does not deviate a lot from the expected trajectory between the interpolation steps. We first did consider interpolating the entire trajectory, then we decided on interpolating the offset and then adding it the desired trajectory, therefore limiting the deviation to a minimal. Sadly, due to time limitation we did not get the chance to fully test the interpolator approach.

We hope that in future steps further experiments for the second concept can be run. We also would like to finish the multi-DoF pipeline we were building in parallel to the single-DoF one and fully validate the concept in simulation. We also hope to run training experiments on real hardware, both on a single-DoF mock-up of the elastic system and on the full David arm.

9 Bibliography

1. DLR. *DLR Institute of Robotics and Mechatronics* [online]. 2022 [visited on 2022-12-27]. Available from: `https://www.dlr.de/rm/en/desktopdefault.aspx/tabid-8017/`.
2. DLR. *DLR David* [online]. 2022 [visited on 2022-12-27]. Available from: `https://www.dlr.de/rm/en/desktopdefault.aspx/tabid-11666/#gallery/27642`.
3. WOLF, Sebastian; EIBERGER, Oliver; HIRZINGER, Gerd. The DLR FSJ: Energy based design of a variable stiffness joint. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 5082–5089. Available from doi: `10.1109/ICRA.2011.5980303`.
4. KEPPLER, Manuel; LAKATOS, Dominic; OTT, Christian; ALBU-SCHÄFFER, Alin. Elastic Structure Preserving (ESP) Control for Compliantly Actuated Robots. *IEEE Transactions on Robotics*. 2018, vol. 34, no. 2, pp. 317–335. Available from doi: `10.1109/TRO.2017.2776314`.
5. KEPPLER, Manuel; LAKATOS, Dominic; OTT, Christian; ALBU-SCHAFFER, Alin. Elastic Structure Preserving Impedance (ESπ)Control for Compliantly Actuated Robots. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 5861–5868. Available from doi: `10.1109/IROS.2018.8593415`.
6. SPONG, M. W. Modeling and Control of Elastic Joint Robots. *Journal of Dynamic Systems, Measurement, and Control*. 1987, vol. 109, no. 4, pp. 310–318. issn 0022-0434. Available from doi: `10.1115/1.3143860`.
7. KEPPLER, Manuel; OTT, Christian; ALBU-SCHÄFFER, Alin. From underactuation to quasi-full actuation: Aiming at a unifying control framework for articulated soft robots. *International Journal of Robust and Nonlinear Control*. 2022, vol. 32, no. 9, pp. 5453–5484. Available from doi: `10.1002/rnc.6102`.
8. HARDER, Marie; KEPPLER, Manuel; MENG, Xuming; OTT, Christian; HÖPPNER, Hannes; DIETRICH, Alexander. Simultaneous Motion Tracking and Joint Stiffness Control of Bidirectional Antagonistic Variable-Stiffness Actuators. *IEEE Robotics and Automation Letters*. 2022, vol. 7, no. 3, pp. 6614–6621. Available from doi: `10.1109/LRA.2022.3176094`.
9. KEPPLER, Manuel; RASCHEL, Clara; WANDINGER, David; STEMMER, Andreas; OTT, Christian. Robust Stabilization of Elastic Joint Robots by ESP and PID Control: Theory and Experiments. *IEEE Robotics and Automation Letters*. 2022, vol. 7, no. 3, pp. 8283–8290. Available from doi: `10.1109/LRA.2022.3187277`.

10. ACHIAM, Joshua. Spinning Up in Deep Reinforcement Learning. 2018.
11. HAARNOJA, Tuomas; ZHOU, Aurick; ABBEEL, Pieter; LEVINE, Sergey. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. arXiv, 2018. Available from doi: `10.48550/ARXIV.1801.01290`.
12. REINL, C.; FRIEDMANN, M.; BAUER, J.; PISCHAN, M.; ABELE, E.; STRYK, O. von. Model-based off-line compensation of path deviation for industrial robots in milling applications. In: *2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. 2011, pp. 367–372. Available from doi: `10.1109/AIM.2011.6027113`.
13. YU, Wen; DE LA SEN, M. Adaptive control of arbitrary nonlinear inputs with feedforward compensation. In: *1999 European Control Conference (ECC)*. 1999, pp. 735–740. Available from doi: `10.23919/ECC.1999.7099393`.
14. WELLS, R.L.; SCHUELLER, J.K.; TLUSTY, J. Feedforward and feedback control of a flexible robotic arm. *IEEE Control Systems Magazine*. 1990, vol. 10, no. 1, pp. 9–15. Available from doi: `10.1109/37.50663`.
15. BONDI, P.; CASALINO, G.; GAMBARDELLA, L. On the iterative learning control theory for robotic manipulators. *IEEE Journal on Robotics and Automation*. 1988, vol. 4, no. 1, pp. 14–22. Available from doi: `10.1109/56.767`.
16. TAO, Hongfeng; LI, Jian; CHEN, Yiyang; STOJANOVIC, Vladimir; YANG, Huizhong. Robust point-to-point iterative learning control with trial-varying initial conditions. *IET Control Theory &amp+ Applications*. 2020, vol. 14, no. 19, pp. 3344–3350. Available from doi: `10.1049/iet-cta.2020.0557`.
17. ZHANG, Dailin; WANG, Zining; MASAYOSHI, Tomizuka. Neural-Network-Based Iterative Learning Control for Multiple Tasks. *IEEE Transactions on Neural Networks and Learning Systems*. 2021, vol. 32, no. 9, pp. 4178–4190. Available from doi: `10.1109/TNNLS.2020.3017158`.
18. BROCKMAN, Greg; CHEUNG, Vicki; PETTERSSON, Ludwig; SCHNEIDER, Jonas; SCHULMAN, John; TANG, Jie; ZAREMBA, Wojciech. *OpenAI Gym*. 2016. Available from eprint: `arXiv:1606.01540`.
19. SCHMIDT, Florian. *Links and Nodes*. [N.d.].
20. RAFFIN, Antonin; HILL, Ashley; GLEAVE, Adam; KANERVISTO, Anssi; ERNESTUS, Maximilian; DORMANN, Noah. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*. 2021, vol. 22, no. 268, pp. 1–8. Available also from: `http://jmlr.org/papers/v22/20-1364.html`.
21. PASZKE, Adam; GROSS, Sam; MASSA, Francisco; LERER, Adam; BRADBURY, James; CHANAN, Gregory; KILLEEN, Trevor; LIN, Zeming; GIMELSHEIN, Natalia; ANTIGA, Luca; DESMAISON, Alban; KOPF, Andreas; YANG, Edward; DEVITO, Zachary; RAI-

SON, Martin; TEJANI, Alykhan; CHILAMKURTHY, Sasank; STEINER, Benoit; FANG, Lu; BAI, Junjie; CHINTALA, Soumith. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. Available also from: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

22. RAFFIN, Antonin; HILL, Ashley; GLEAVE, Adam; KANERVISTO, Anssi; ERNESTUS, Maximilian; DORMANN, Noah. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*. 2021, vol. 22, no. 268, pp. 1–8. Available also from: `http://jmlr.org/papers/v22/20-1364.html`; `https://github.com/araffin/sbx`.

Erklärung / Declaration

Ich versichere, dass ich diese Masterarbeit selbstständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben, sowie wörtliche und sinngemäße Zitate gekennzeichnet habe.

I certify that I have completed this master's thesis independently, have not submitted it elsewhere for examination purposes, have indicated all sources and aids used, and have marked literal and analogous quotations.

Wolfertschwenden, den 28.02.2023 Christoph Sood

Unterschrift / Signature

Ermächtigung / Authorization

Hiermit ermächtige ich die Hochschule Kempten zur Veröffentlichung der Kurzzusammenfassung (Abstract) meiner Arbeit, z. Bsp. auf gedruckten Medien oder auf einer Internetseite.

I hereby authorize the Kempten University of Applied Sciences to publish the short summary (abstract) of my work, e.g. on printed media or on a website.

Wolfertschwenden, den 28.02.2023 Christoph Sood

Unterschrift / Signature