Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Herman Kolstad Jakobsen

Reinforcement learning for robotic soft-body interaction

Master's thesis in Cybernetics and Robotics
Supervisor: Jan Tommy Gravdahl
Co-supervisor: Andreas Østvik and Akhil S. Anand
June 2021

NTNU
Norwegian University of
Science and Technology

Herman Kolstad Jakobsen

Reinforcement learning for robotic soft-body interaction

Master's thesis in Cybernetics and Robotics
Supervisor: Jan Tommy Gravdahl
Co-supervisor: Andreas Østvik and Akhil S. Anand
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

It is deemed challenging to integrate robotic systems into medical procedures due to the complex scene involved. The variations between patients, combined with the handling of moving objects and soft materials, has proven to be an obstacle for robot-assisted procedures. Research within reinforcement learning has facilitated the design of new robot controllers, making it possible for robot manipulators to learn from experience. Data is limited, but realistic simulators have shown to be a viable source for acquiring the necessary training material and experience for robot manipulators to adapt to a real-life scenario. Employing sophisticated robotic systems could significantly benefit medical procedures and possibly help reduce the increasing workload in the health sector.

In this thesis, a simulation framework was used to investigate how deep reinforcement learning can be used as robot control for soft body interaction tasks. Three different models were trained to complete an interaction task of sweeping a probe across the surface of a soft body, while exerting a desired contact force and keeping a desired velocity. The baseline model outputs a desired wrench and was used as a reference to quantify the sampling efficiency and performance increase of using an appropriate low-level controller. The two other models were used to examine how a variable impedance control law would perform at the interaction task, where the manipulator has to compensate for the motion and deformation of the body dynamically. The action space of the models consisted of proportional gains, making it possible to vary the impedance along the execution of the task. The action space of one model was extended with an additional parameter, allowing control of the end-effector movement in the z-direction directly. In the test episode, the baseline model showed promising results before yielding unstable behavior. The variable impedance models tracked the pose and velocity in a satisfactorily manner. However, the models struggled to track the desired force, especially where frequent spike noise characterized the applied force in the simulator measurements. Suggestions on how to overcome this limitation is presented.

Combining reinforcement learning with impedance control looks promising for solving complex interaction tasks with high uncertainty. However, a vast amount of work and further development are needed to fully exploit the potential and transfer the technology into clinical applications.

Sammendrag

Innen klinisk medisin ansees det som utfordrende å benytte autonome robotsystemer. Håndtering av bevegelige objekter og myke materialer, samt variasjon mellom pasienter, har vist seg å være en hindring for robotassisterte inngrep. Forskning innen *reinforcement learning* har lagt til rette for utformingen av nye robotkontrollere, noe som gjør det mulig for robotmanipulatorer å lære av erfaring. Datatilgangen er begrenset, men realistiske simulatorer har vist seg som en mulig kilde for nødvendig opplæringsmateriell og erfaring slik at robotmanipulatorer kan tilpasse seg virkelige scenario. Å benytte sofistikerte robotsystemer kan gi en betydelig fordel innen klinisk medisin og muligens bidra til å redusere den økende arbeidsmengden i helsesektoren.

I denne masteroppgaven ble et simuleringsrammeverk brukt for å undersøke hvordan *deep reinforcement learning* kan brukes som robotkontroll for myke kroppsinteraksjonsoppgaver. Tre forskjellige modeller ble trent for å fullføre interaksjonsoppgaven. Referansemodellen gir en ønsket kraft og dreiemoment, og ble brukt for å kvantifisere prøvetakingseffektiviteten og ytelsesøkningen ved å bruke en passende lavnivåkontroller. De to andre modellene ble brukt til å undersøke hvordan variabel impedanskontroll ville håndtere interaksjonsoppgaven, der manipulatoren må kunne dynamisk kompensere for kroppens bevegelse og deformasjon. Handlingsområdet til modellene besto av justerbare forsterkningsgrader, noe som gjorde det mulig å variere impedansen under utførelsen av oppgaven. Handlingsområdet til en modell ble utvidet med en tilleggsparameter som tillot direkte kontroll av bevegelsen i z-retningen. Iløpet av testepisoden viste referansemodellen lovende resultater før den ga ustabil oppførsel. De variable impedansmodellene fulgte ønsket positur og hastighet på en tilfredsstillende måte. Generelt slet modellene med å utøve ønsket kontaktkraft, hvor den påførte kraften var svært støyete. Forslag for å overkomme denne begrensningen ble presentert.

Å kombinere *reinforcement learning* med impedanskontroll ser lovende ut for å løse komplekse interaksjonsoppgaver med høy usikkerhet. Imidlertid er det behov for mye arbeid og videreutvikling for å utnytte potensialet fullt ut og overføre teknologien til kliniske applikasjoner.

Preface

This paper is submitted as a master's thesis in Robotic Systems and concludes my master's degree within the Cybernetics and Robotics programme at the Norwegian University of Technology and Science.

Even though I am listed as the only author, this work is not a result of my contributions alone. Therefore, I would like to say thank you to my supervisors for helping and guiding me throughout this semester. Thanks to Jan Tommy Gravdahl for your administrative contributions, and for sharing your knowledge and thoughts during our meetings. A tremendous thank you to Andreas Øsvtik and Akhil S. Anand for your invaluable guidance and expertise. Your passion for this project is contagious, and it would not have been possible to finish this thesis without your help. I am grateful for having had the opportunity to learn from such dedicated and skilled scientists.

My five years at NTNU have been both challenging and rewarding. Hence, I would like to express my sincere gratitude to all my friends and family for making these five years into an experience I would not wish to be without. Thank you mom, dad, Fredrik and Jesper for always supporting me. I hope you all know how much you mean to me.

Trondheim, June 6, 2021

Herman Kolstad Jakobsen

Contents

Chapter 1

Introduction

Robotic and automated systems are rapidly incorporated into society, where they can perform time-consuming and repetitive tasks previously performed by humans. Robotic assistants were first used in medicine in the mid-1980s [1], and they have since developed into becoming a well-established part of clinical procedures. Robotic systems have enormous potential in the healthcare sector, where they can perform precision-demanding tasks with high repeatability. Further, robots have greater durability compared to humans, allowing them to meet the ever-increasing demand for treatments and medical procedures. Generally, the cost of industry-standard robot manipulators is decreasing, ultimately making them more accessible. In terms of cost, robotic systems can prove economically beneficial and a viable alternative to current solutions.

That being said, automation of tasks using robot manipulators, such as ultrasound imaging, has not been widely implemented in the healthcare sector. One reason is that modern robot systems are still incapable of proficiently handling moving objects and soft materials [2]. This makes interaction with moving and shifting body parts and organs difficult, not only because the patient does not always lie entirely still, but also because of breathing motion and pulsation. In addition, as part of the medical procedure, the body will be manipulated, introducing further interaction uncertainties.

While conventional control methods, such as PID regulators, are proficient at following references and trajectories in free space [3], they offer restricted adaptive behavior. Mainly, these methods show limited performance in applications that require contact between the robot and its environment. Identifying and modeling contact interaction is complex, making it demanding to achieve adaptable yet robust robot control with the use of conventional methods [4]. Conventional control methods also lack the ability to generalize their behavior, meaning that all possible system behavior must be considered during their design phase.

Reinforcement learning methods could potentially solve these challenges, where they are capable of yielding robot controllers that both generalize well and can han-

dle uncertain environments [5]. By learning from experience, reinforcement learning methods unlock a whole new range of possibilities within robot control. These methods, however, often require large quantities of data to produce adequate results, and how to effectively collect such large amounts of data remains an unresolved problem. Using simulators to replicate the physical world has proved to be a viable approach, where the robot learns from simulated experiences.

1.1 Goal of the thesis

The work done in this master's thesis utilizes the simulation framework created and designed by the author in autumn 2020. The overarching goal is to investigate the use of deep reinforcement learning and robot control for robotic soft body interactions. This includes examining how the combination of deep reinforcement learning and different low-level controllers affects the performance, as the robot arm dynamically compensates for uncertainties such as object's motion and deformation. More precisely, the goal is to make a robot manipulator, with an ultrasound probe attached as its end-effector, learn how to perform a sweeping motion across the surface of a soft body, while both exerting a desired contact force and keeping a desired velocity. Lastly, the work aspires to facilitate further research and development on the use of reinforcement learning for robotic soft body interaction tasks.

1.2 Contributions

This thesis examines how reinforcement learning can be used for soft body interaction robot control, where the robot manipulator has to dynamically compensate for the deformation and motion of a soft body. The contributions of this thesis are

- Quantification of how reinforcement learning, combined with variable impedance control, perform at a soft body interaction task.
- A tunable and modular reward function for learning pose, force and velocity tracking.
- Investigation of how the learning efficiency is affected by the choice of observation space.
- Sample-efficiency and performance comparisons of combining a reinforcement learning algorithm with different low-level controllers and action spaces.
- A simulation framework facilitating training of reinforcement learning models for soft body interaction tasks.

1.3 Outline

This master's thesis is organized into seven chapters, excluding appendices. The following outlines the remaining chapters.

Chapter 2 - Background. Gives an overview of the theoretical prerequisites. This chapter should give the reader a fundamental understanding of the theory needed to apprehend the task at hand, and also the ability to comprehend the work presented in the following chapters.

Chapter 3 - Methodology. Describes the methods used to make the robot manipulator able to learn a sweeping motion across a soft body. This includes details regarding the choice of reinforcement learning algorithm, the design of the reward function and observation space, and the implementation of low-level controllers. Modifications done to the simulation framework are also mentioned.

Chapter 4 - Experimental Setup. Explains the setup used for conducting the experiments. The chapter summarizes the parameters chosen for configuring the algorithm and the controller. The organization of the reinforcement learning training process is also presented.

Chapter 5 - Results. Presents the results from the reinforcement learning models.

Chapter 6 - Discussion. Discusses the results presented in the previous section, and how the models performed. Considering the overall framework, suggestions for future work will be presented.

Chapter 7 - Conclusion. Concludes the thesis with closing remarks.

Chapter 2

Background

In the beginning of this chapter, a brief overview of robot dynamics in operational space is presented . This section is followed by the introduction of three compliant control methods. Fundamental reinforcement learning theory is then presented, together with an overview of a popular reinforcement learning algorithm. The chapter is concluded with a an overview over some related work.

2.1 Robot Dynamics

2.1.1 Jacobian

For a manipulator with n-joints in three dimensions, the Jacobian $\mathbf{J} \in \mathbb{R}^{6\times n}$ is a mapping between the spatial velocity $\boldsymbol{\nu}$ and the joint velocities $\dot{\mathbf{q}}$. The relationship is given by

$$\boldsymbol{\nu} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}, \tag{2.1}$$

where $\boldsymbol{\nu}$ consists of the linear velocity $\dot{\mathbf{p}}$ and the angular velocity $\boldsymbol{\omega}$, i.e. $\boldsymbol{\nu} = \begin{bmatrix}\dot{\mathbf{p}} & \boldsymbol{\omega}\end{bmatrix}^T$. It is worth noting that the Jacobian is dependent on the joint configuration $\mathbf{q}$. Generally, the Jacobian is computed using the forward kinematics of the manipulator.

2.1.2 Operational Space Dynamics

When planning and executing robotic manipulation tasks, it is often advantageous to define the dynamics of the manipulator in the operational space. Not only is this formulation more intuitive for the user, but it also simplifies the handling of end-effector constraints.

From [6], the dynamic model in operational space for a rigid manipulator with 6 degrees of freedom can be stated as

$$\boldsymbol{\Lambda}(\boldsymbol{\xi})\ddot{\boldsymbol{\xi}} + \boldsymbol{\Gamma}(\boldsymbol{\xi},\dot{\boldsymbol{\xi}})\dot{\boldsymbol{\xi}} + \boldsymbol{\eta}(\boldsymbol{\xi}) = \mathbf{h}_c - \mathbf{h}_e, \tag{2.2}$$

where ξ is a six-dimensional vector representing the position and orientation of

the end-effector. Specifically, $\boldsymbol{\xi} = \begin{bmatrix} \mathbf{p} & \boldsymbol{\phi} \end{bmatrix}^T$, where $\mathbf{p}$ is the position and $\boldsymbol{\phi}$ is a set of Euler angles describing the orientation. Hence, the relationship $\dot{\boldsymbol{\xi}} = \boldsymbol{\nu}$ can be established. Further, $\mathbf{h}_c$ is the controller output and $\mathbf{h}_e$ is the external wrench. The cartesian inertia matrix is denoted $\boldsymbol{\Lambda}(\boldsymbol{\xi}) \in \mathbb{R}^{6\times6}$, while $\boldsymbol{\Gamma}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) \in \mathbb{R}^{6\times6}$ is the wrench caused by centrifugal and Coriolis effects. The wrench due to gravitational effects is represented by $\boldsymbol{\eta}(\boldsymbol{\xi}) \in \mathbb{R}^{6\times1}$.

The inertia matrix is computed as

$$\boldsymbol{\Lambda}(\boldsymbol{\xi}) = (\mathbf{J}\mathbf{H}(\mathbf{q})^{-1}\mathbf{J}^T)^{-1}. \tag{2.3}$$

Here, $\mathbf{H}(\mathbf{q}) \in \mathbb{R}^{n\times n}$ is the symmetric and positive-definite joint space inertia matrix which represents the mass distribution of the manipulator in joint space. Naturally, this matrix is highly dependent on the joint configuration. Further, the wrench caused by centrifugal and Coriolis effects is calculated as

$$\boldsymbol{\Gamma}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) = \mathbf{J}^{-T}(\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{H}(\mathbf{q})\mathbf{J}^{-1}\dot{\mathbf{J}})\mathbf{J}^{-1}, \tag{2.4}$$

where $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the centrifugal and Coriolis effects given in the joint space. The wrench due to gravitational effects is given by a simple mapping from joint space to operational space:

$$\boldsymbol{\eta}(\boldsymbol{\xi}) = \mathbf{J}^{-T}\mathbf{g}(\mathbf{q}), \tag{2.5}$$

where $\mathbf{g}(\mathbf{q})$ are gravitational effects given in the joint space.

2.2 Compliant Robot Control

2.2.1 Stiffness Control

Stiffness control can be regarded as the corner stone of indirect force control [7]. Indirect force control methods achieves force control through control of the manipulator's motion. That is, the manipulator will change its reference position in order to achieve a compliant interaction with the environment.

Consider the following motion control law, consisting of a PD controller and gravity compensation, given in the operational space

$$\mathbf{h}_c = \mathbf{A}^{-T}(\boldsymbol{\phi})\mathbf{K}_P\Delta\boldsymbol{\xi} - \mathbf{K}_D\boldsymbol{\nu} + \boldsymbol{\eta}(\boldsymbol{\xi}). \tag{2.6}$$

Here, the pose error between the current end-effector pose and a reference pose is denoted $\Delta\boldsymbol{\xi}$. The matrices $\mathbf{K}_P$ and $\mathbf{K}_D$ are symmetric and positive 6×6 gain matrices. The $\mathbf{A}$ matrix maps velocity from the joint space to the operational space, and is defined as

$$\mathbf{A}(\boldsymbol{\phi}) = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}(\boldsymbol{\phi}) \end{bmatrix}, \tag{2.7}$$

where $\mathbf{I}$ is the 3×3 identity matrix, $\mathbf{0}$ is a 3×3 null matrix, and $\mathbf{T}$ is the 3×3 matrix of the mapping $\boldsymbol{\omega} = \mathbf{T}(\boldsymbol{\phi})\dot{\boldsymbol{\phi}}$. The $\mathbf{T}$ matrix is dependent on the particular choice of Euler angles.

Utilizing Lyapunov stability theory [8] in the absence of interaction with the environment (i.e. $\mathbf{h}_e = \mathbf{0}$), it can be shown that the asymptotically stable equilibrium for the closed-loop system is given by $\Delta\boldsymbol{\xi} = \mathbf{0}$ and $\boldsymbol{\nu} = \mathbf{0}$. In the presence of a constant wrench $\mathbf{h}_e$, Lyapunov stability theory yields the following asymptotically stable equilibrium

$$\mathbf{h}_e = \mathbf{A}^{-T}(\boldsymbol{\phi})\mathbf{K}_P\Delta\boldsymbol{\xi}. \tag{2.8}$$

According to (2.8), the end-effector will in the steady-state behave as a six-degree-of-freedom spring in respect of the external wrench $\mathbf{h}_e$. The $\mathbf{K}_P$ matrix will therefore act as an active stiffness, where the elements of the matrix specify the elastic behavior of the end-effector during interaction with the environment. A higher active stiffness value will yield higher accuracy of the position control at the cost of higher interaction forces, while lower values will allow discrepancies in position to ensure low interaction forces. Designing this static relationship between the deviation in the end-effector pose and the force exerted on the environment, is known as stiffness control.

To summarize, by designing an appropriate stiffness matrix, stiffness control allows to limit the interaction force at the expense of end-effector pose deviations. However, choosing the stiffness parameters is difficult and the appropriate values is highly dependent of the task at hand. In addition, the control method struggles to handle uncertain environments and disturbances. In such cases, low active stiffness values may produce large and unwanted deviations in the end-effector pose [9].

2.2.2 Impedance Control

Another type of indirect force control is impedance control. Impedance control is a unified control scheme used to simultaneously control both the motion of a manipulator and the contact force, by adjusting the impedance [10]. Here, impedance refers to the dynamic relationship between the motion variables of the manipulator and the contact force, as opposed to the static relationship utilized in stiffness control. Impedance control is suitable for dealing with mechanical interaction tasks where contacts between objects are present, and is often used in manipulation tasks where it is desired to control the position of an end-effector while simultaneously maintaining the contact force in a preset safety range. The use of impedance control makes it possible to overcome position uncertainties and subsequently large impact forces, since manipulators are controlled to modulate their motion or compliance according to force perceptions.

When impedance first was introduced in mechanical manipulation, it referred to the ratio between an output effort and an input flow [11]. The input flow represents

the velocity of the manipulator, while the output effort is the contact force resulted from the interaction motion between the manipulator and the environment. In the frequency domain, the impedance $Z(s)$ can be defined as the ratio of the Laplace transformed effort $F(s)$ to the Laplace transformed flow $\dot{X}(s)$,

$$Z(s) = \frac{F(s)}{\dot{X}(s)}. \tag{2.9}$$

Substituting $sX(s)$ for $\dot{X}(s)$, (2.9) can be rewritten into

$$F(s) = sZ(s) \cdot X(s) \tag{2.10}$$

For manipulation tasks it is desirable to control both the contact force $F(s)$ and the motion of the manipulator $X(s)$ accurately. Due to the coupling $sZ(s)$ between the states, it is, however, not possible to control $F(s)$ and $X(s)$ independently. As a compromise, impedance control works by directly controlling $X(s)$. Then, by designing a desired impedance $Z(s)$, the contact force $F(s)$ can be indirectly regulated by (2.10). The impedance controller commonly resembles a virtual spring-damper system between the environment and robot end-effector

$$\mathbf{Z}(s) = \boldsymbol{\Lambda} s + \mathbf{B} + \frac{\mathbf{K}}{s}, \tag{2.11}$$

where the coefficient matrices $\boldsymbol{\Lambda}$, $\mathbf{B}$ and $\mathbf{K}$ represent the desired inertia, damping and stiffness of the system, respectively. Choosing the impedance controller to resemble a virtual spring allows for the manipulator to interact with the environment in a safe and energy-efficient way.

Correspondingly, in the time domain, the desired impedance can be expressed by a differential equation

$$\boldsymbol{\Lambda}_d(\ddot{\boldsymbol{\xi}} - \ddot{\boldsymbol{\xi}}_d) + \mathbf{B}_d(\dot{\boldsymbol{\xi}} - \dot{\boldsymbol{\xi}}_d) + \mathbf{K}_d(\boldsymbol{\xi} - \boldsymbol{\xi}_d) = \mathbf{F}(t), \tag{2.12}$$

where the matrices $\boldsymbol{\Lambda}_d$, $\mathbf{B}_d$ and $\mathbf{K}_d$ represent the desired inertia, damping and stiffness, respectively. The desired damping and stiffness are usually referred to as k_v gains and k_p gains, respectively. The actual contact force is denoted $\mathbf{F}(t)$, which corresponds to a controller's calculated input u. The actual pose of the end-effector is denoted $\boldsymbol{\xi}$ and $\boldsymbol{\xi}_d$ is the desired trajectory pose of the end-effector, in the operational space.

The conventional procedure of implementing an impedance control law is to substitute the designed target impedance into the actual dynamics of the manipulator, where the desired impedance is usually defined as a second-order dynamic equation

$$\boldsymbol{\Lambda}_d(\ddot{\boldsymbol{\xi}} - \ddot{\boldsymbol{\xi}}_d) + \mathbf{B}_d(\dot{\boldsymbol{\xi}} - \dot{\boldsymbol{\xi}}_d) + \mathbf{K}_d(\boldsymbol{\xi} - \boldsymbol{\xi}_d) = -\mathbf{h}_e. \tag{2.13}$$

The controller is then able to impose impedance-defined dynamics on the original, and more complicated, end-effector dynamics.

Without loss of generality, assume that we have a six degrees of freedom robotic manipulator, meaning the dynamics of the manipulator in the operational space is given by (2.2). The control law is then obtained by inserting (2.13) into (2.2):

$$\boldsymbol{\tau} = \mathbf{J}^T \mathbf{F} \tag{2.14a}$$

$$\mathbf{F} = \mathbf{h}_e + \boldsymbol{\Gamma}\dot{\boldsymbol{\xi}} + \boldsymbol{\eta} + \boldsymbol{\Lambda}\{\ddot{\boldsymbol{\xi}}_d - \boldsymbol{\Lambda}_d^{-1}[\mathbf{B}_d(\dot{\boldsymbol{\xi}} - \dot{\boldsymbol{\xi}}_d) + \mathbf{K}_d(\boldsymbol{\xi} - \boldsymbol{\xi}_d) + \mathbf{h}_e]\} \tag{2.14b}$$

The manipulator driven by the control torque $\boldsymbol{\tau}$ obtained from (2.14), will follow the dynamics defined by the impedance in (2.13).

2.2.3 Variable Impedance Control

Robots need to vary their impedance along the execution of certain tasks [12]. For instance, robots in unstructured and uncertain environments may need to interact with objects of varying physical properties. Such tasks demand the use of different control forces according to the different mass, friction forces and other physical traits of the objects. In such scenarios, measured forces give valuable information regarding the control forces needed to perform the tasks, which again can be governed through stiffness and damping variations.

In order to state the Variable Impedance Controller (VIC), it is first needed to slightly modify (2.13) into

$$\boldsymbol{\Lambda}_d(\ddot{\boldsymbol{\xi}} - \ddot{\boldsymbol{\xi}}_d) + \mathbf{B}_d(t)(\dot{\boldsymbol{\xi}} - \dot{\boldsymbol{\xi}}_d) + \mathbf{K}_d(t)(\boldsymbol{\xi} - \boldsymbol{\xi}_d) = -\mathbf{h}_e, \tag{2.15}$$

where $\mathbf{B}(t)$ and $\mathbf{K}(t)$ are same the quantities as defined in (2.12). The only difference is that the quantities are now time-varying. For a six degrees of freedom manipulator, the VIC can be obtained by inserting (2.15) into (2.2), essentially yielding the same control law as (2.14), only now with time-varying damping and stiffness matrices.

There exists numerous methods to update the stiffness and damping matrices of the variable impedance controller. One of the simplest methods is to update the joint stiffness online using a primitive adaption rule

$$\mathbf{k}_t = \mathbf{k}_0 + \alpha \mathbf{e}_t^2 \tag{2.16}$$

where $t = 1, 2, \cdots, T$ are time-steps, $\mathbf{k}_t = \text{diag}(\mathbf{K}_t) = \begin{bmatrix} k_{1,t} & \cdots & k_{J,t} \end{bmatrix}^T$ are the joint stiffness and J is the number of joints. The vector $\mathbf{k}_0 = \begin{bmatrix} k_{1,0} & \cdots & k_{J,0} \end{bmatrix}^T$ is a small stiffness used to prevent unsafe interaction, $\mathbf{e}_t$ is the joint trajectory error and α is a positive gain. In practice, the update rule (2.16) increases the joint stiffness when the trajectory error is high, making the robot able to track the desired trajectory more accurately.

Recently, robot learning algorithms have gained great interest for learning, reproducing and adapting variable impedance parameters. These types of algorithms are collectively called for Variable Impedance Learning (VIL), and they utilize methods such as imitation learning, iterative learning and reinforcement learning. Generally, VIL methods learn a nonlinear mapping $\boldsymbol{\phi}(\cdot)$ in the form

$$\mathbf{K}_{d,t} = \boldsymbol{\phi}^{\mathcal{K}}(\mathbf{x}_t, \dot{\mathbf{x}}_t, \mathbf{f}_t^e, \boldsymbol{\theta}^{\mathcal{K}}) \tag{2.17a}$$

$$\mathbf{B}_{d,t} = \boldsymbol{\phi}^{\mathcal{B}}(\mathbf{x}_t, \dot{\mathbf{x}}_t, \mathbf{f}_t^e, \boldsymbol{\theta}^{\mathcal{B}}), \tag{2.17b}$$

where the learning algorithm uses N demonstrations in the form $\{\{\mathbf{x}_{t,n}.\dot{\mathbf{x}}_{t,n}, \mathbf{f}_t^e\}_{t=1}^T\}_{n=1}^N$ and a set of parameters $\boldsymbol{\theta}$ to learn parameterized impedance gains. At run time, the learned model in (2.17) is used to retrieve the desired impedance gains from the measurements. The techniques used to approximate the nonlinear mappings $\boldsymbol{\phi}^{\mathcal{K}}$ and $\boldsymbol{\phi}^{\mathcal{B}}$ distinguish the different VIL approaches. Some VIL methods also performs inertia shaping where a desired inertia matrix $\boldsymbol{\Lambda}_{d,t}$ is estimated.

Variable Impedance Learning Control (VILC) is categorized as methods that erase the boundary between the learning algorithm and the controller design. As with VIL, VILC uses training data to learn both parameterized impedance gains The key difference between VIL and VILC is that, in VILC, the data collection process itself depends on the underlying control structure. Hence, VILC methods also learn a parameterized reference trajectory. Compared to VIL, VILC approaches adopt more complex impedance learning strategies requiring iterative updates and robot self-exploration. Utilizing reinforcement learning, variable impedance can be adopted as a parameterized policy

$$\pi_{\theta,t} = \mathbf{K}_{\theta,t}(\boldsymbol{\xi}_{d,\theta,t} - \boldsymbol{\xi}_t) + \mathbf{B}_{\theta,t}(\dot{\boldsymbol{\xi}}_{d,\theta,t} - \dot{\boldsymbol{\xi}}) + \mathbf{f}_t^e \tag{2.18}$$

where $\pi_{\theta,t}$ is a control policy depending on a set of learnable parameters θ. The parameters define the desired trajectory, $\boldsymbol{\xi}_{d,\theta,t}$ and $\dot{\boldsymbol{\xi}}_{d,\theta,t}$, as well as the desired impedance behaviour, $\mathbf{K}_{\theta,t}$ and $\mathbf{B}_{\theta,t}$. As stated, methods from reinforcement learning can be used to find good values for these parameters.

2.3 Reinforcement Learning

2.3.1 Overview

Reinforcement learning is a type of machine learning, and it aims at achieving a goal by learning from interaction and experience [13]. The learner, also known as the decision-maker, is called the agent. The agent interacts with its surroundings, which is called the environment. The agent and environment interact continually with each other, where the agent selects actions and the environment responds to these actions by presenting new situations, or observations, to the agent. The interaction between the agent and the environment is illustrated in Fig. 2.1. In addition, the environment

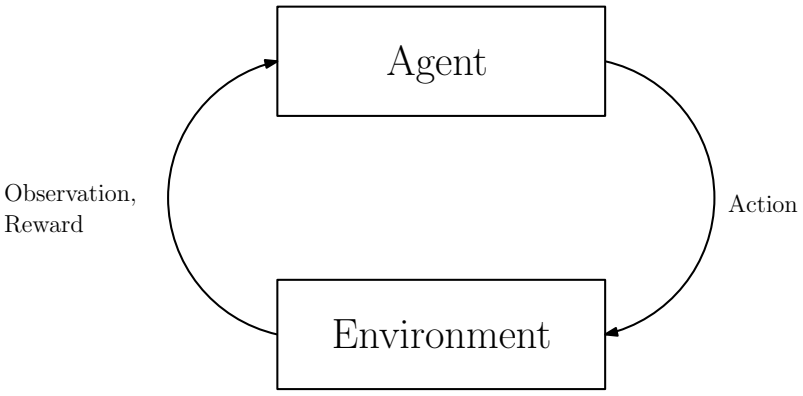


Figure 2.1: Interaction between the agent and the environment in reinforcement learning. The agent selects an action, and the environment responds by presenting new observations to the agent. The agent also receives a reward depending on how good the selected action was for achieving the overarching goal.

outputs reward signals represented as scalar values that the agent tries to maximize over time. To summarize, the goal of the agent is to maximize the total amount of reward it achieves over the long run by choosing actions given the situation the agent is in.

More precisely, assuming a discrete scenario, the agent and the environment interact at each discrete time step $t = 0, 1, 2, 3, ...$ At each time step t, the agent receives a representation of the environment's state $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states. A state can simply be interpreted as environmental information available to the agent, such as sensory data. Based on the given state, the agent then selects an action $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(S_t)$ is the set of actions available in state S_t. One time step later, the agent will find itself in a new state S_{t+1} and receive a numerical reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, both based on its chosen action. Hence, the reward R_{t+1} and the new state S_{t+1} are jointly determined by the previous state S_t and the chosen action A_t.

In order to determine an action at each time step t, the agent needs to implement a mapping from states to probabilities of selecting each possible action in that state. This mapping is called the agent's policy and is denoted π_t. The probability of choosing action $A_t = a$ when the agent is in state $S_t = s$ is denoted $\pi(a|s)$. In essence, reinforcement learning methods specify how the agent changes its policy as a result of its experience.

In many cases, the interaction between the agent and the environment can be naturally broken into subsequences, which are called episodes. Examples of this are games like chess and checkers or manipulation tasks for robots. Each episode have a final time step T and ends in a distinct state called the terminal state, followed by a reset to a starting state. The starting state can either be chosen as a standard or sampled from a standard distribution of starting states. Tasks with episodes of this kind are called episodic tasks.

2.3.2 Returns

As stated, the goal of the agent is to maximize the cumulative reward it receives in the long run. More generally speaking, the agent aims to maximize the expected return, where the return G_t is defined as some specific function of the reward sequence. Denoting the sequence of received rewards after time step t as $R_{t+1}, R_{t+2}, R_{t+3}, ...$, the return can in the simplest case be defined as the sum of the rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \tag{2.19}$$

where T is the final time step.

The return formulation (2.19) is problematic for non-episodic tasks. That is, tasks that go on continually without a limit and have a final time step of $T = \infty$. Such tasks could for instance be applications for robots with long life spans or continual control tasks. In these cases, the return that the agent wishes to maximize could itself be infinite.

The problem of infinite returns can be solved by introducing the concept of discounting. Instead of maximizing the sum of all future rewards, the agent should try to select actions so that the sum of the discounted rewards it receives over the future is maximized. Specifically, the agent chooses A_t to maximize the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.20}$$

where $0 \leq \gamma \leq 1$ is the discount rate.

The discount rate decides the present value of future returns, where a return received k time steps in the future is only worth γ^{k-1} times what it would be worth if it was to be received immediately. Choosing $\gamma < 1$, the infinite sum has a finite value as long as the reward sequence R_k is bounded. Setting $\gamma = 0$ will make the agent concerned with only maximizing the immediate rewards. That is, the agent will learn how to choose A_t to maximize R_{t+1}. Hence, the discount rate γ is a measure on how farsighted the agent will be. As γ approaches 1, the agent takes future rewards more strongly into account.

2.3.3 Markov Decision Processes

A reinforcement learning task that satisfies the Markov property is called a Markov decision process (MDP). That is, the response of the task's environment at time $t+1$ depends only on the state and action representations at time t. More formally, assuming there exists a finite number of states and reward signals, the environment's dynamics of a task that satisfies the Markov property is specified by

$$p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}, \tag{2.21}$$

for all r, s', S_t and A_t. If the state and action spaces are finite, the task is called a finite Markov decision process (finite MDP).

Given the dynamics defined in (2.21), it is possible to compute the expected reward for state-action pairs,

$$r(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a), \tag{2.22}$$

the state-transition probabilities

$$p(s'|s,a) = \Pr\{S_{t+1} = s'|S_t = s, A_t = a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a), \tag{2.23}$$

and the expected rewards for state-action-next-state triples,

$$r(s,a,s') = \mathbb{E}\left[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'\right] = \frac{\sum_{r\in\mathcal{R}} rp(s', r|s, a)}{p(s'|s,a)}. \tag{2.24}$$

2.3.4 Value Functions

Value functions are functions of states (or action-state pairs) that estimate how good it is for the agent to be in the given state (or perform a given action in the given state). Usually, the expected return is used as a measurement of "goodness". Generally, reinforcement learning algorithms involve estimating value functions.

As stated, an agent's expected return is dependent on what actions the agent will take. Hence, value functions are defined with respect to certain policies. The value of a state s under a policy π, denoted $v_\pi(s)$, is the expected return when starting in s and following π thereafter. The function v_π is called the state-value for policy π. More formally, $v_\pi(s)$ can be defined for MDPs as

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right], \tag{2.25}$$

where $\mathbb{E}_\pi[\cdot]$ indicates the expected value of a random variable given that the agent follows policy π, and the value t is an arbitrary time step. It is worth noting that the value of a terminal state is always zero.

Likewise, the value of taking an action a in state s under a policy π, denoted $q_\pi(s,a)$, can be defined as the expected return starting from s, taking the action a, and following policy π thereafter

$$q_\pi = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right]. \tag{2.26}$$

The function q_π is called the action-value function for policy π.

Value functions satisfy useful recursive relationships that are commonly utilized in reinforcement learning. For any policy π and any state s, the following holds,

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|S_t = s\right] \\ &= \sum_a \pi(a|s)\sum_s^{'}\sum_r p(s',r|s,a)\left[r + \gamma\mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}|S_{t+1} = s'\right]\right]. \end{aligned} \tag{2.27}$$

Using (2.22), the recursive property of the value functions can be shown by further simplifying (2.27):

$$v_\pi = \sum_a \pi(a,s)\sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]. \tag{2.28}$$

Equation (2.28) is the Bellman equation for v_π, and it states a relationship between the value of a state and the value of its successor states.

2.3.5 Optimal Value Functions

In essence, solving a reinforcement task means finding a policy that maximizes the expected return. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. More formally, $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}$ for all $s \in \mathcal{S}$. It always exists at least one policy that is equal to or better than all other policies. This is an optimal policy and is denoted π_*. All the optimal policies share the same optimal state-value function v_* defined as

$$v_*(s) = \max_\pi v_\pi(s), \tag{2.29}$$

for all $s \in \mathcal{S}$

The optimal action-value function q_* is also shared between the optimal policies, and is defined as

$$q_*(s,a) = \max_\pi q_\pi(s,a), \tag{2.30}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. For the state-action pair (s,a), the optimal action-value function gives the expected return for performing action a in state s, and then following an optimal policy thereafter. Hence, the following relationship between q_* and v_* can be derived:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a\right]. \tag{2.31}$$

Since v_* is an optimal value function, the Bellman equation for v_* is called the Bellman optimality equation. This equation states that the value of a state under an optimal policy must equal the expected return for the best action for that state,

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}\left[G_t | S_t = s, A_t = a\right] \\ &= \max_a \mathbb{E}_{\pi_*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \\ &= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s, A_t = a\right] \\ &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right] \\ &= \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_*(s')\right]]. \end{aligned} \tag{2.32}$$

The Bellman optimality equation for q_* is

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s',a')\right]. \end{aligned} \tag{2.33}$$

For finite MDPs, if the dynamics of the environment $p(s',r|s,a)$ is known, it is in principle possible to find a unique solution of (2.32). In other scenarios, (2.32) can be approximately solved using different methods.

Once v_* is obtained, the actions that appear to be best after a one-step search will be optimal actions. In other words, any policy that is greedy with respect to the optimal value function v_* is an optimal policy. The optimal value function v_* takes into account the reward consequences of all possible future behaviour, effectively turning the optimal expected long-term return into a quantity that is locally and immediately available for each state. With q_*, the agent can simply choose the action that maximizes $q_*(s,a)$ for any state s, in order to determine an optimal policy.

2.3.6 Policy Gradient Methods

For tasks with small number of states and actions, estimates of the value functions are usually represented as a table with one entry for each state (or state-action pair). However, in many of the tasks where it is desirable to apply reinforcement learning, most of the encountered states will never have been experienced before. To learn on these tasks, it is necessary to generalize from previously experienced states to unseen ones.

Function approximation is a wide-spread generalization method which takes samples from a desired function (e.g. value function) and attempts to generalize from them to construct an approximation of the entire function. The standard way of approximating the value function and then determine a policy from it has, however, shown to be theoretically unmanageable. For instance, small changes in the value function can lead to large, and potentially breaking, changes in the resulting policy.

Instead of approximating a value function and then use that to compute a deterministic policy, policy gradient methods approximate a stochastic policy directly using an independent function approximator with its own parameters [14]. Usually, a policy is represented by a neural network whose input is a representation of the state and the output is action selection probabilities. The network weights will then work as the policy parameters. Letting θ be the the vector of the policy parameters and ρ the performance of the corresponding policy (e.g. average reward per step), the policy parameters are updated approximately proportional to the gradient

$$\Delta\theta \approx \alpha \frac{\partial \rho}{\partial \theta}, \tag{2.34}$$

where α is a positive-definite step size. As a contrast to the value-function approach, small changes in θ will cause only small changes in the policy and in the state-visitation distribution. If the update rule (2.34) is achieved, then θ can usually be assured to converge to a locally optimal policy in the performance measure ρ.

2.3.7 Actor-Critic Methods

Actor-critic methods are types of Temporal Difference (TD) methods which aim at learning a policy π by estimating a value function v [15]. The estimation is usually done through the use of policy gradient methods. Especially for TD methods, the agent learns directly from raw experience without any knowledge of the environment's dynamics - an approach which is often called for model-free learning. TD methods also utilizes bootstrapping, where the methods update estimates based on previously calculated estimates. The estimates are updated online, meaning the observed reward and value function at the next time step, R_{t+1} and $V(S_{t+1})$, are used to update the estimates. In this way, TD methods do not have to wait for an episode to end before updating its value function estimates as compared to offline methods.

Actor-critic methods are recognized by their characteristic architecture shown in Fig. 2.2. The policy structure is called for the actor, as it is deciding which action to perform. The estimated value function is known as the critic, because it judges the actions made by the actor. Usually, the critic is a state-value function. All the learning is on-policy, which means the critic learns about and critiques whatever policy currently being followed by the actor. The critique is a scalar error signal which facilitates all the learning in both the actor and the critic. After an action is

selected, the critic evaluates the new state in order to decide whether the selected action lead to a better or worse state than expected. The evaluation is defined as the error

$$\delta_t = R_{t+1} + \gamma V_t(S_{t+1}) - V(S_t). \tag{2.35}$$

Here, V_t is the value function implemented by the critic at time t. A positive error encourages to increase the chance of choosing action A_t when in state S_t, while a negative error suggests a decrease in the chance of choosing this action being in state S_t.

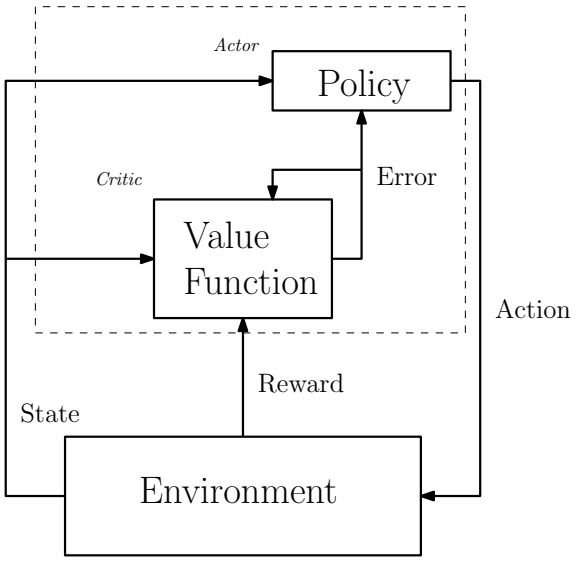


Figure 2.2: Architecture for actor-critic methods. Based on the current state, the actor chooses an action resulting in the environment presenting a new state to the actor and the critic. Using the corresponding reward for choosing this action, the critic calculates an error that is used to update both the policy and the value function estimate.

Inspired by: [13]

2.3.8 Domain Randomization

Domain randomization is a method used to improve the generalization capabilities of reinforcement learning models, where the simulation environment is randomized during training to expose the model to a wide range of environment variants. This method is proven to effectively reduce the reality gap [16]. The reality gap is a collective term used to describe discrepancies between physics simulators and the real world, such as unmodeled physics and low-fidelity simulated sensors. A large reality gap makes it difficult to transfer behaviors from simulation into the real world, ultimately forming a barrier to use simulated data on real robots. However, if the variability in the training environment is sufficiently large, models trained in simulation are capable of transferring to the real world with limited additional

training. In such cases, the real world may appear to the model as just another variation.

2.4 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [17] is a reinforcement learning algorithm that has gained great popularity in the recent years. The algorithm is based on an actor-critic structure, combined with trust region methods, to achieve performance that is comparable or better than state-of-the-art approaches, while being much simpler to implement. In short, trust region methods work by first defining a region around the current policy and then confine the updated policy to lie within this region. In this way, PPO is able to yield robust policy updates in both continuous and discrete action spaces.

In order to estimate the value function, PPO aims at maximizing the following objective function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right], \tag{2.36}$$

where the expectation $\hat{\mathbb{E}}_t[...]$ at timestep t specifies the empirical average over a finite batch of samples, as the algorithm alternates between sampling and optimization. The probability ratio between the new and the old policy is denoted $r_t(\theta)$. That is,

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, \tag{2.37}$$

where $r(\theta_{\text{old}}) = 1$. The estimator of the advantage function is denoted $\hat{A}_t$ and serves the same purpose as (2.35). The estimator is defined as

$$\hat{A}_t = \delta_t + \gamma\delta_{t+1} + \cdots + \gamma^{T-t+1}\delta_{T-1}, \tag{2.38}$$

where δ_t is given by (2.35). The second term $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)$ clips the probability ratio and is used to remove the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$ where ϵ is a hyperparameter usually chosen to $\epsilon = 2$. By taking the minimum of the clipped and the unclipped objective $r_t(\theta)\hat{A}_t$, the final objective works as a lower bound on the unclipped objective. Hence, the change in probability ratio is ignored when it would make the objective improve, and included when it makes the objective worse. It is also possible to further modify the objective function by adding a penalty on KL divergence [18]. Both the clipped objective and KL divergence contribute to PPO making robust policy updates.

An implementation of the PPO algorithm, taken from [17], is shown in Algorithm 1. It is worth noting that PPO facilitates multiprocessing, as it is possible to run

several actors at once.

Algorithm 1: PPO, Actor-Critic Style

for *iteration = 1, 2, ...* **do**
  for *actor = 1, 2, ..., N* **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
  end
  Optimize objective function L wrt. θ, with K epochs and minibatch size $M < NT$
  $\theta_{\text{old}} \leftarrow \theta$
end

2.5 Related Work

Considerable research has been done on combining reinforcement learning with contact-rich manipulation tasks in unstructured and uncertain environments. These tasks often require haptic and visual feedback to yield robust control. An approach of using self-supervision learning to combine multisensory information is proposed in [19]. To combat the sample complexity of high-dimensional continuous state-action spaces, [20] divides the task into free and contact-rich sub-tasks. To handle the contact-rich sub-tasks, the dynamic movement primitive framework [21] is extended by a coupling term that provides active compliance under contact with the environment. Another approach to increase the training efficiency is to learn the environment dynamics with Gaussian Process while training the policy, and utilize the learned dynamic model to improve target value estimation [22].

Previous work has demonstrated the importance of having force control when learning interaction tasks. Force control can be learned explicitly [23], where the policy is initialized from a user-provided kinetic demonstration and a reinforcement learning algorithm is later used to optimize the policy through trial and error. A more modern approach is to learn force control by learning a desired wrench [24]. The technique combines reinforcement learning with force and torque information by incorporating a hybrid operational space motion/force controller. The reinforcement learning algorithm is configured to learn a desired wrench on the end-effector, which is then sent to the controller. Through the choice of the desired wrench, the hybrid controller implicitly achieves adaptive impedance behavior. To process the noisy force and torque readings from the sensors, a novel neural network architecture was introduced where the force/torque information is concatenated in the second last layer of the network.

Learning impedance schemes in the task space can significantly improve the learning speed of manipulation tasks [25]. The operational space formulation used in this work shows the advantage of abstracting the robot kinematics for contact-rich

tasks. However, this may lead to a drawback of fixing the redundancy resolution scheme, limiting the range of potential behaviors. Furthermore, results show that the best choice for a task-space may vary across and within tasks [26].

A policy directly outputting joint torques should in theory be able to solve any task. In practice, however, it is seen that reinforcement learning algorithms struggle to robustly solve delicate tasks without any organization of the action space. How the choice of action space can give robust performance in tasks with contact uncertainties is investigated in [27]. The authors propose a policy giving output impedance and desired position in joint space.

Most state-of-the-art work in robotic manipulation focuses on handling rigid bodies. However, deformable object manipulation has many relevant real-world applications, such as surgery [28]. Because of the large configuration space of deformable objects, solutions using traditional modelling approaches requires extensive engineering work. Using deep reinforcement learning algorithms, [29] solves the problem of manipulating a deformable cloth. The solution utilizes RGB images to derive the position of the cloth, and by exposing the agent for domain randomization while training in simulation, the agent is successfully deployed in the real world.

Ultrasound imaging is progressively becoming more automated. Recently, a machine learning based robot-assisted system that automatically scans tissue was proposed [30]. The ultrasound image feedback system automatically adjusts the probe contact force based on the image quality. The images' correlation, compression and noise characteristics are fed into a support vector machine classifier, and the robot arm adjusts the scanning force based on the classifiers output. Another approach of automating ultrasound scans is to determine the scan range and scan path after finding a 3-D contour of the skin surface [31]. The 3-D contour is found by adopting a depth camera to capture a point cloud of the skin surface. A normal-vector-based method is then used to determine the pose of the ultrasound probe corresponding to each scan point along the scan path. For fine-tuning of the pose, contact force feedback from two force sensors was implemented.

Chapter 3

Methodology

The work done in the project thesis [32], which includes the creation of a simulation environment for robot-assisted medical procedures, was used as a foundation for further development. Utilizing the simulation framework, it was now possible to explore the potential of learning soft contact interaction tasks with the use of reinforcement learning.

3.1 Simulation Framework

As mentioned, the work in this thesis builds on the simulation framework created in [32]. An overview of the framework architecture, which uses the robosuite framework [33] as foundation, is shown in Fig. 3.1. A simulation model is instantiated by the MuJoCo engine to make a simulation runtime, referred to as an environment. The policy sends a set of actions to the low-level controller, and the controller uses these actions to compute a set of torques. Based on the torques, the MuJoCo engine performs internal calculations to determine the next state of the simulation. The sensors retrieve information from the new simulation state and generate corresponding observations, which are then sent back to the policy. The data flow during the training of a reinforcement learning model is represented by the dashed lines. Compared to the framework in [32], the environment has been extended with a trajectory generator. A trajectory is created by the environment at initialization and a desired position and orientation is sent to the controller at each timestep.

The simulation environment is defined by the simulation model which contains definitions of the robot and object models. Using robosuite's modelling APIs, it is possible to create an environment containing relevant objects and robots with customized end-effectors. The simulation environment created in [32] is shown in Fig. 3.2. The environment consists of a robot manipulator, a soft body and a table. Note that the shape of the soft body has been changed into a box shape, compared to the cylindrical shaped soft body used in [32]. An ultrasound probe is attached as the robot manipulator's end-effector. The environment provides support for both

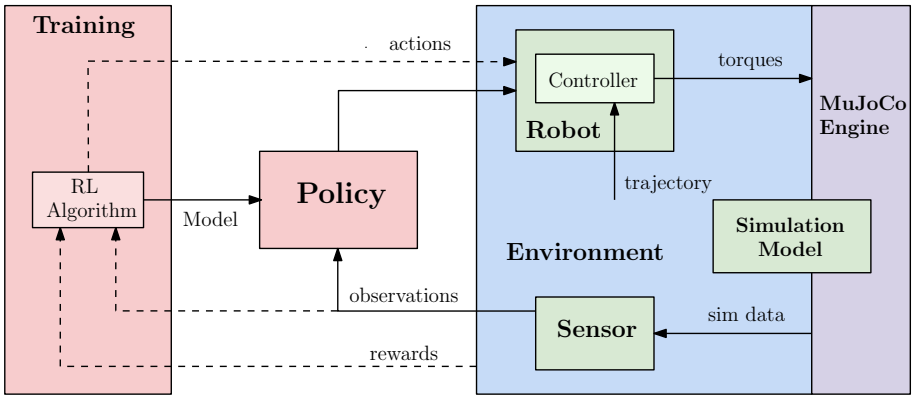


Figure 3.1: Framework architecture. Actions are sent from the policy to the robot's controller and converted into low-level torque commands. The MuJoCo engine uses the torque commands to calculate a new simulation state. The sensors interprets the new simulation state and converts them into observations, which are sent back to the policy. A desired trajectory pose is fed to the controller at each timestep. The training loop is illustrated by the dashed lines.

Inspired by: [32]

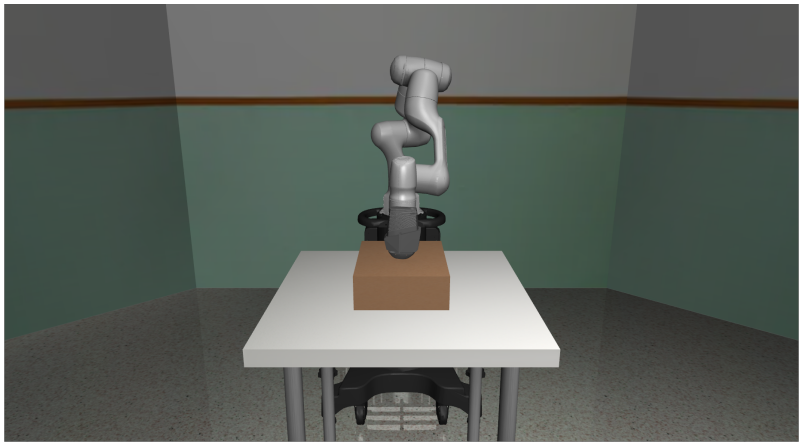
the Panda [34] and the UR5e [35] robot manipulator.

The soft body object is a central component of the simulation environment, and is shown in Fig. 3.3. The soft object consists of several, smaller rigid bodies placed in a grid. Each rigid body has a joint to the grid center, making it possible for each body to be displaced. Simultaneous displacements of the rigid bodies will make the object compress or expand, ultimately replicating the physical properties of a soft body. It is further possible to specify the weight, quantity, spacing and size of the rigid bodies, together with the stiffness and damping of the joints, allowing for full customization of the soft object's physical properties. A visualization of the soft body's physical properties is shown in Fig. 3.4.

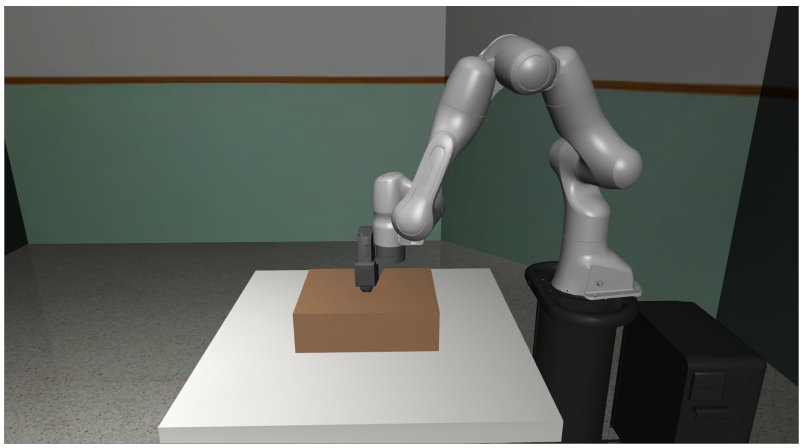
3.2 Low-level Controller

As stated, the goal of this thesis is to explore the capabilities of making a robot manipulator learn to handle soft contacts using reinforcement learning. The trained policy outputs a set of actions that are sent to a low-level controller, as shown in Fig. 3.1. Hence, a good choice of low-level controller is crucial in laying a foundation for the agent to be successful. In order to handle the uncertainties of an environment containing a soft body, the controller should be able to vary its impedance along the execution of the task.

As mentioned, [25] presents a variable impedance controller in the end-effector space (VICES). The controller has shown itself advantageous for constrained and contact-rich tasks, a trait that can prove itself beneficial when interacting with soft



(a) Front view.



(b) Side view.

Figure 3.2: Simulation environment. The environment contains a robot manipulator (i.e. Panda robot) with a probe as its end-effector, a soft body and a table.

objects. The controller builds on the operational space controller (OSC) formulation [36], where the torques applied to the robot joints are given by

$$\tau = u + \tau_{gravity} + \tau_{null}. \tag{3.1}$$

Here, $\tau_{gravity}$ are torque compensations due to gravity and Coriolis forces, and τ_{null} are the nullspace torques. The VICES control law u is defined as

$$\begin{aligned} u = & J_{pos}^T \left[\Lambda^{pos} \left[k_p^{pos}(p_{des} - p) - k_v^{pos} v \right] \right] + \\ & J_{ori}^T \left[\Lambda^{ori} \left[k_p^{ori}(R_{des} \ominus R) - k_v^{ori} \omega \right] \right], \end{aligned} \tag{3.2}$$

where $\Lambda \in \mathbb{R}^{6\times6}$ is the inertial matrix in the end-effector frame that decouples the end-effector motion. The quantities Λ^{pos} and Λ^{ori} are the parts of the inertial matrix that corresponds to position and orientation, respectively. The position and orientation parts of the end-effector Jacobian is denoted J_{pos} and J_{ori}, respectively. The symbol $\ominus$ represents subtraction in $\mathbb{SO}(3)$. The desired position p_{des} and ori-

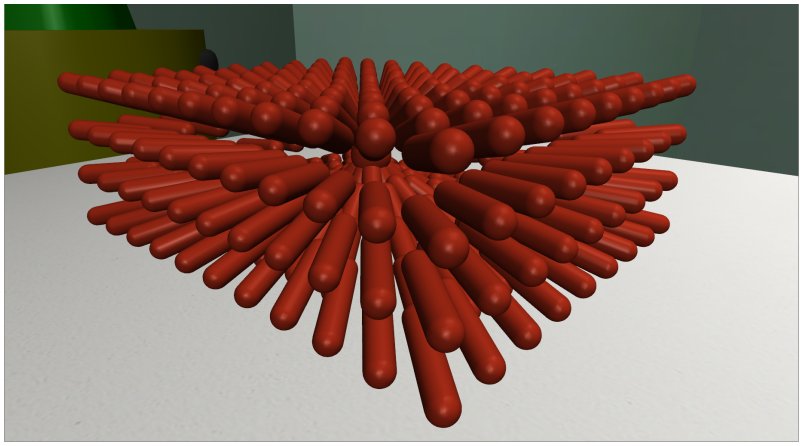


Figure 3.3: Soft body object. The soft body consists of several rigid bodies placed in a grid. Each rigid body has a joint connected to the center, allowing each body to be displaced. The displacements of the rigid bodies replicates the physical properties of a soft body. Note that the soft body is default covered by a skin mesh, which is effectively concealing the rigid bodies.

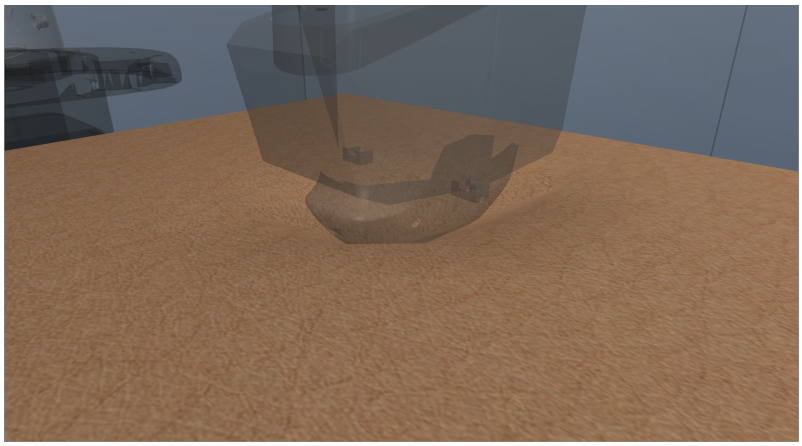


Figure 3.4: Visualization of the soft body's physical properties. The ultrasound probe applies a contact force on the soft body, making the body compress. The probe, robot manipulator and tabletop have been made transparent to enhance visibility.

entation R_{des} are given by the trajectory. In essence, (3.2) is a PD controller in end-effector space.

Choosing the controller gains, k_p^{pos}, k_v^{pos}, k_p^{ori} and k_v^{ori}, as the action space allows the agent to get full control of the probe's compliant behavior. Changing the gains while following a trajectory will allow the probe to effectively adjust to its environment, where, for instance, the z-component of k_p^{pos} and k_v^{pos} can be reduced to allow displacements from the trajectory in the z-position. This is a useful attribute for interactions with soft bodies, as the probe may need to adjust its compliance depending on the exerted force and physical properties of the body. Hence, the

policy action space $\mathcal{A}$ for the first model was chosen to be

$$\mathcal{A} = \{k_p^{pos}, k_p^{ori}\}, \tag{3.3}$$

This action space consists of a total of six values. The three first values k_p^{pos} corresponds to the proportional gain for the position tracking, while the three next values k_p^{ori} corresponds to the proportional gain for the orientation tracking. Note that in (3.2), the orientation error is described by a set of axis-angles. The velocity gains were chosen such that critical damping was obtained. That is,

$$k_v^{pos} = 2\sqrt{k_p^{pos}} \tag{3.4a}$$

$$k_v^{ori} = 2\sqrt{k_v^{ori}}. \tag{3.4b}$$

To make the probe's behavior in the z-direction fully independent, practically separating it from the desired trajectory z-position, a second policy with an extended action space was created. The extended action space $\mathcal{A}_z$ was chosen as the following:

$$\mathcal{A}_z = \{k_p^{pos}, k_p^{ori}, \Delta z\}, \tag{3.5}$$

where Δz is a relative displacement in z-position. Instead of only being able to control the compliance in the z-direction, the controller is now able to control the z-position of the end-effector directly. Ultimately, this gives more freedom in controlling the movement in z-direction. It was believed that this extended action space would make it easier for the manipulator to exert a desired contact force. As a consequence of introducing Δz, the desired position was modified to

$$p_{des} = \begin{bmatrix} x_{des} & y_{des} & z_{des} + \Delta z \end{bmatrix}, \tag{3.6}$$

where x_{des} , y_{des} and z_{des} are the desired positions given by the trajectory in the x-, y- and z-direction, respectively. The velocity gains of the controller were sat according to (3.4).

A baseline policy was also generated in order to quantify the sample efficiency and enhanced performance of using an appropriate low-level controller. Instead of having the controller gains as its action space, the baseline policy outputs a desired wrench. That is, the policy outputs the desired force f_{des} and desired torque τ_{des} that should be applied to the probe directly, effectively removing the low-level controller layer. More specifically, the control law is modified to be

$$u_{bl} = J^T \Lambda \begin{bmatrix} f_{des} \\ \tau_{des} \end{bmatrix}, \tag{3.7}$$

where J is the Jacobian. The action space $\mathcal{A}_{bl}$ of the baseline policy was then chosen as

$$\mathcal{A}_{bl} = \{f_{des}, \tau_{des}\}. \tag{3.8}$$

3.3 Reward Function

A critical part of making an agent capable of learning a task using reinforcement learning is the design of the reward function. As stated, rewards are used as measurements on the "goodness" of a selected action. This reward is then used to update the current policy, meaning the reward function is implicitly defining the learned behaviour of the agent. Therefore, the overarching goal of a task is defined by an environment's reward function. For instance, if a robot manipulator is to learn how to lift up a box, the manipulator should get rewards to incentive such a behaviour. This could include rewards for performing actions such as the manipulator moving its end-effector closer to the box, coming in contact with the box and actually lifting up the box.

The design of the reward function is inspired by [37]. As known, the goal of the ultrasound task is to make a robot manipulator, with an attached ultrasound probe as its end-effector, learn how to perform a sweeping motion on the surface of a soft body. More precisely, the manipulator's end-effector should learn to follow a reference trajectory going along the surface of a soft body, while both exerting a reference contact force and keeping a reference mean velocity throughout the episode. Since the task consists of different components, it is natural to also divide the reward function into corresponding reward components. With a modular form for the reward function, it is possible to weight the components based on their importance.

The reward function contains terms which encourage following the reference trajectory, exerting a desired force and keeping a desired mean velocity throughout the episode. The trajectory term is further divided into two parts; one part encouraging minimization of the probe's position error and one part encouraging minimization of the probe's orientation error. The same is done for the force term; one part encourages minimization of the force deviation while the other part encourages minimization of change in the exerted force. The reward function r_{tot} can be summarized as

$$r_{\text{tot}} = w_p r_p + w_o r_o + w_f r_f + w_d r_d + w_v r_v, \tag{3.9}$$

where r_p is the positional reward, r_o is the orientation reward, r_f is the force reward, r_d is the derivative force reward and r_v is the velocity reward. The weights $w_p = 5$, $w_o = 1$, $w_f = 3$, $w_d = 2$ and $w_v = 1$ are the corresponding reward weights, where w_p weights the position reward, w_o weights the orientation reward, w_f weights the force reward, w_d weights the derivative force and w_v weights the velocity reward. The weights were chosen manually based on their importance for completing the task. The most central part of the task is to learn the manipulator to follow a trajectory while exerting a desired contact force. Thus, the trajectory term and the force term were given the largest weights.

Generally, each reward term takes the following form

$$r = \exp(-e), \tag{3.10}$$

where r is a reward term and e is an error corresponding to that reward term. Due to the nature of the exponential function, all the reward terms are scaled such that they lie in the interval $[0,1]$. The magnitude of the reward term can therefore be decided by multiplying the term with an appropriate weight. It is also possible to specify the sensitivity of the reward term by scaling the error. Hence, by designing each reward term according to (3.10), it is possible to decide both the sensitivity and magnitude of each term. As an effect, the reward function in (3.9) can be shaped to fit various requirements.

The position error e_p, which denotes the norm of the squared difference between the probe position and the reference trajectory in the xy-plane, is defined as

$$e_p = \left\| \left[c_p(\mathbf{p}_t - \mathbf{p}_{\text{goal},t}) \right]^2 \right\|, \tag{3.11}$$

where $\mathbf{p}_t$ is the x and y position of the probe, and $\mathbf{p}_{\text{goal},t}$ is the x and y position of the reference trajectory at timestep t. The error coefficient was set to $c_p = 90$.

The orientation error e_o, which denotes the difference between the current orientation of the probe and the desired orientation, is defined as

$$e_o = c_o \cdot d(\mathbf{q}_t, \mathbf{q}_{\text{goal},t}). \tag{3.12}$$

Here, $c_0 = 0.2$. The quaternions $\mathbf{q}_t$ and $\mathbf{q}_{\text{goal},t}$ represent the current orientation of the probe and the goal orientation where the probe is kept at an upright position, respectively. The quaternions are defined as $\mathbf{q} = v + \mathbf{u} \in S^3$, where S^3 is a unit sphere in $\mathbb{R}^4$, $v \in \mathbb{R}$, $\mathbf{u} \in \mathbb{R}^3$. The distance metric between two quaternions $d(\cdot)$ is a scalar value, and it is given by [38]

$$d(\mathbf{q}_1, \mathbf{q}_2) = \begin{cases} 2\pi, & \mathbf{q}_1 * \bar{\mathbf{q}}_2 = -1 + [0,0,0]^T \\ 2\left\|\log(\mathbf{q}_1 * \bar{\mathbf{q}}_2)\right\|, & \text{otherwise} \end{cases}, \tag{3.13}$$

where the quaternion logarithm is given by

$$\log(\mathbf{q}) = \log(v + \mathbf{u}) = \begin{cases} \arccos(v)\frac{\mathbf{u}}{\|\mathbf{u}\|}, & \mathbf{u} \neq 0 \\ [0,0,0]^T, & \text{otherwise} \end{cases}. \tag{3.14}$$

The force error e_f, denoting the squared difference between the running mean of the probe contact force in z-direction and the goal contact force, can be expressed as

$$e_f = \left[c_f(\bar{f}_t - f_{\text{goal},t}) \right]^2, \tag{3.15}$$

where $c_f = 0.7$. Due to noisy contact force measurements, a running mean was used as a filter to smooth out the measurements. The running mean $\bar{f}_t$ was chosen as an exponential moving average [39] such that

$$\bar{f}_t = \alpha f_t + (1-\alpha)\bar{f}_{t-1}, \quad 0 < \alpha \leq 1. \tag{3.16}$$

The probe contact force measurement in z-direction at timestep t is denoted f_t, while α is a smoothing factor chosen as $\alpha = 0.1$.

The derivative force error e_d, denoting the squared difference between the derivative of the contact force in the z-direction and a goal derivative force, is defined as

$$e_d = \left[c_d(f'_t - f'_{\text{goal},t})\right]^2, \tag{3.17}$$

where $c_d = 0.01$. The derivative of the force was calculated as

$$f'_t = \frac{f_t - f_{t-1}}{T_c}. \tag{3.18}$$

Here, T_c is the timestep of the controller.

The velocity error e_v, defined as the squared difference between the running mean of the probe velocity and the goal mean velocity, can be written as

$$e_v = \left[c_v(\bar{v}_t - v_{\text{goal},t})\right]^2, \tag{3.19}$$

where $c_v = 45$. In order to ensure that the average velocity throughout the episode is kept at a desired value, a running mean was used instead of the raw velocity measurement. By choosing the running mean $\bar{v}_{\text{probe, t}}$ as a single moving average [40], all the measurements would be weighted equally throughout the episode. More specifically

$$\bar{v}_t = \bar{v}_{t-1} + \frac{\|\mathbf{v}_{t-1}\| - \bar{v}_{t-1}}{N}, \tag{3.20}$$

where $\mathbf{v}_t$ is the probe velocity at timestep t, and N is the number of timesteps taken in the episode (i.e. number of previous data points). Equation (3.20) is based on Welford's online algorithm [41].

Overall, the error multipliers were chosen empirically, such that each error showed reasonable sensitivity. That is, the error should not be too sensitive nor too unresponsive, as that would make it difficult for the agent to learn.

3.4 Observation Space

To achieve efficient training and good performance, it is important to represent the environment state in a satisfactory manner. That is, the observations should be rich and contain useful information the agent might need to learn a well performing policy. However, the observation space should also be minimalistic, as unuseful or redundant observations could potentially make the reinforcement algorithm learn slower. The trade-off between including more observations that might be helpful for the agent to learn and keeping the observation space minimalistic, is something worth taking into consideration when designing the observation space.

The chosen observation space consists of the following observations:

- End-effector contact force.

- End-effector torque.
- End-effector velocity.
- End-effector pose difference, i.e. deviation between trajectory and end-effector pose.
- Difference between the derivative of the contact force and a goal derivative contact force.
- Difference between running mean contact force and goal contact force.
- Difference between running mean velocity and goal velocity.

The first five observations were deemed necessary for the agent to learn a good performing policy, as they contain essential information. The last two observations, however, were added to investigate if they would accelerate the learning process. Specifically, the information contained in the last two observations can be deduced by combining the information given by the other observations and the reward function. Hence, it can be argued that the two last observations are redundant. It is also common to add joint measurements, like joint positions and joint velocities. These observations were judged unnecessary as the control law already contains this information through the implemented forward kinematics of the manipulator.

3.5 Algorithm

Due to its sample efficiency, stability, ease-of-use and multi-processing capabilities, the stable-baselines [42] PPO algorithm implementation was chosen as the preferred reinforcement learning algorithm. The neural network architecture for the actor and the critic in the PPO algorithm is shown in Fig. 3.5. The actor and the critic both have their own independent network mainly consisting of two fully connected layers. The first layer consists of 256 neurons, while the second layer consists 128 neurons. The activation function used after each layer is *tanh*. Both networks also have a third fully connected layer, which is used to map the output features of the previous layer into appropriate dimensions for the actor and the critic, respectively.

3.6 Reinforcement Learning Techniques

3.6.1 Domain Randomization

In order to generalize the agent's capabilities of interacting with the soft body surface, trajectory randomization was implemented. A visualization of the trajectory randomization is shown in Fig. 3.6, where a grid of data points was created in the

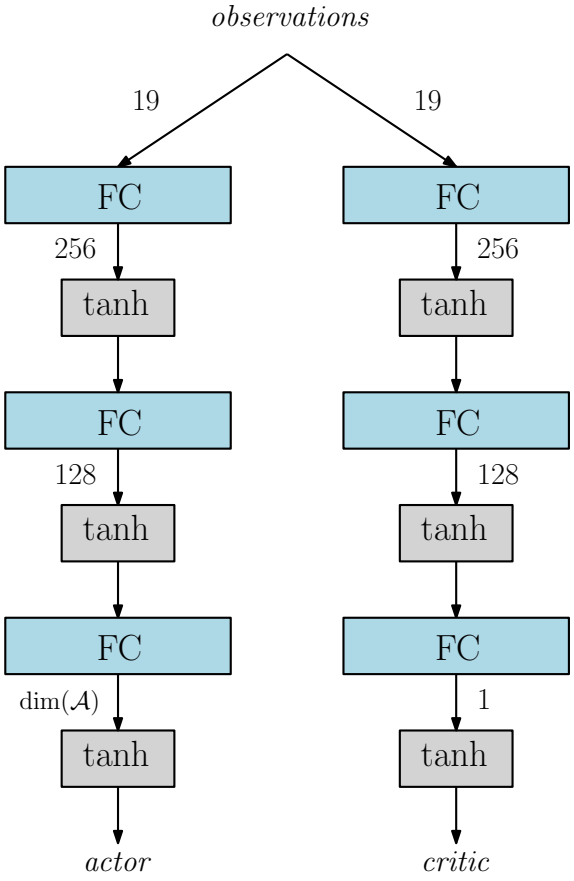


Figure 3.5: Neural network architecture for the actor and the critic in the PPO algorithm. The scalar values represent the number of input features and output features (i.e. neurons) for each fully connected layer.

xy-plane of the torso. The z-position for all the data points in the grid was fixed to the same value. At the start of each episode during training, the start point and end point of the trajectory were randomly selected from the grid. To allow the agent to fully explore each trajectory, the end-effector was initialized at a random point along the trajectory. Gaussian noise was also added to the initial position of the probe to introduce further trajectory variability.

Exposing the agent to randomized trajectories facilitates exploration of the soft body surface. Through exploration, the agent will discover new states and will be forced to act upon them. By continuously exposing the agent to new states it will learn how to generalize its behavior, as opposed to only exposing the agent to a set of deterministic trajectories. If an agent is left to just learn from deterministic trajectories, it will often converge to a local minimum where it will only learn to handle states given by the trajectories in the training set.

To introduce further environment variability during training, the physical properties of the torso were randomized between episodes. In [32], a value range for both the stiffness and damping of the torso were acquired through the execution of a calibration task. During training, the soft torso was initialized with a random

stiffness and damping chosen from their respective value ranges.

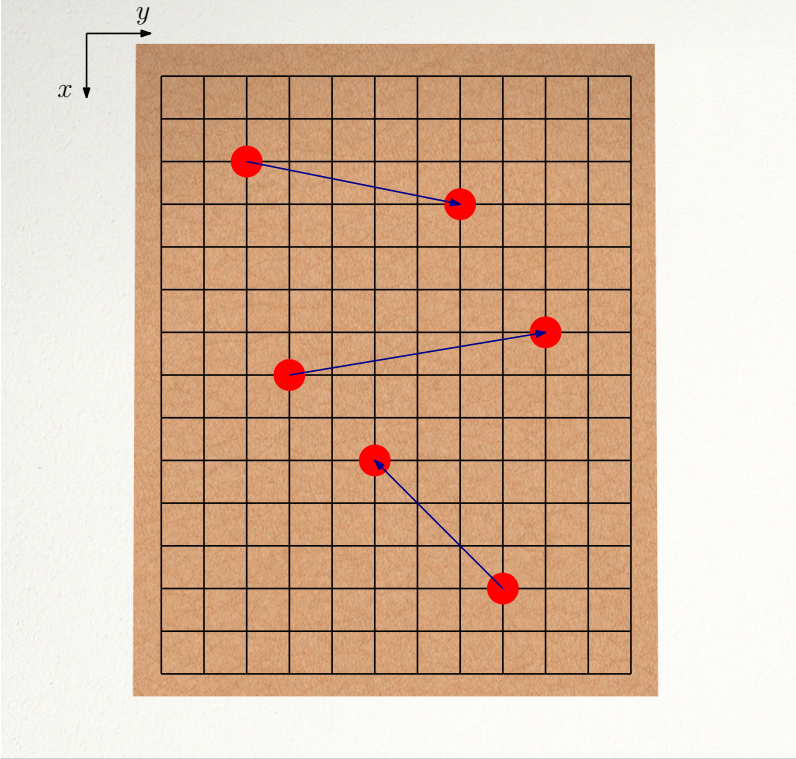


Figure 3.6: Trajectory randomization. A grid of data points was created in the xy-plane of the torso. During training, a start point and end point for the trajectory are randomly chosen from the grid at the start of each episode. Note that that the granularity of the grid in the figure does not match the granularity of the implemented grid.

3.6.2 Normalization

After weight initialization and before training, the actor neural network will generally output values in a small range around zero. In order for the low-level controller to have an acceptable performance, the k_p values should be in the range of hundreds. For the neural network to output values of this magnitude, the weights have to undergo large changes. This is time consuming as the updating process is based on computational demanding simulation data. The wide output value range also makes it hard for the algorithm to efficiently explore the whole action space. Ultimately, the training process is slowed down and the performance of the trained agent is reduced.

To combat these issues, it is important to normalize both the inputs and the outputs of the neural network. The input observations are easily normalized by wrapping the environment with a normalization wrapper provided by the stable-baselines framework. Likewise, the output values were normalized to values close to zero. This was done for all the three action spaces.

Chapter 4

Experimental Setup

This chapter presents the setup and hyperparameters used to conduct the experiments. The configurations of the reinforcement learning algorithm and the low-level controller are first described, before presenting the organization of the training process. To summarize, Section 4.4 gives an overview of the different models.

4.1 Algorithm Hyperparameters

Reinforcement learning algorithms usually have numerous tunable hyperparameters. For the PPO algorithm, the hyperparameters encompass, among others, the batch size for each update and the KL divergence limit. It is also possible to decide the learning rate of the update rule, where support for adaptive learning rates depending on the current remaining training progress is included. The hyperparameters were chosen as the default values specified in the stable-baselines3 implementation [43], since these values have empirically shown to yield good results for a variety of environments.

4.2 Controller Configuration

Like the PPO algorithm, the OSC has a range of configurations. Both the policy output actions and the controller output torques were sent at a frequency of 500 Hz. Practically, this means that a new policy output was sent to the controller at every timestep. Since the policy and controller were running at the same frequency, it was not necessary to interpolate the policy commands. The desired trajectory pose was updated at the same frequency as the controller.

The range for the k_p values was set to $[0, 500]$, with an initial value of $k_p = 300$. Since the PPO actor neural network initially outputs values in the range $[-1, 1]$, the controller's k_p input range was modified to $[0, 1]$. Here, an input of 0 corresponds to $k_p = 0$ and an input of 1 corresponds to $k_p = 500$. The same type of scaling was also done for Δz, f_{des} and τ_{des}. The input limits were set to $[-1, 1]$, while the output

range for Δz was set to $[-0.05, 0.05]$ and the output range for f_{des} and τ_{des} was set to $[-10, 10]$. The input/output mapping is summarized in Table 4.1.

Table 4.1: Input/output mapping for the controller. The actor network gives values in the range specified by the *input* values. The input values are then scaled to be in the interval specified by the *output* values, before they are used in the controller computations.

	min	max
k_p input	0	1
k_p output	0	500
Δz input	-1	1
Δz output	-0.05	0.05
f_{des} input	-1	1
f_{des} output	-10	10
τ_{des} input	-1	1
τ_{des} output	-10	10

The remainder of the control configurations were set to their default values as specified in the robosuite implementation [44].

4.3 Training Configuration

The horizon of each episode (i.e. the episode length) was set to 1000 timesteps. To improve the learning speed and hinder the model from using experience from irrelevant states when training, several early termination conditions were implemented. The episode would terminate early if:

- The joint limit of the robot was reached.
- The probe deviated significantly from the desired trajectory position. More precisely, terminate if:

$$e_p > 1.0 \tag{4.1}$$

- The orientation of the probe deviated significantly from the desired goal orientation while in contact with the torso. More specifically, terminate if:

$$e_o > 0.10 \quad \textit{and} \quad \text{probe is in contact with torso} \tag{4.2}$$

- The probe loses contact with the torso.

For the trajectory randomization shown in Fig. 3.6, a 50×50 grid of data points was created. Note that the following position coordinates are given in the default world frame defined in robosuite. The center of the soft body was located

at $[0, 0, 0.857]$. The z-position of all the data points was set to $z_{grid} = 0.896$, which corresponds to a z-position moderately below the surface of the body. The grid was defined over the ranges $x = [-0.12, 0.15]$ and $y = [-0.09, 0.09]$. The noise applied to the initial z-position of the probe was sampled from a Gaussian distribution with $\mu = 0$ and $\sigma = 0.015$, while the noise applied to the x- and y-positions was sampled from the same type of distribution with $\sigma = 0.00375$.

The stiffness and damping parameters used to randomize the physical properties of the torso were from the prior work [32]. For the stiffness parameter, the values were chosen from the interval $[1300, 1600]$, while the damping parameter was picked from the interval $[17, 41]$.

As stated in the reward function, several goal values need to be chosen. The trajectory goal position is extracted from the trajectory generator at each timestep, while the goal orientation quaternion was set to

$$\mathbf{q}_{goal} = (-0.692, 0.722, -0.005, -0.11). \tag{4.3}$$

This orientation corresponds to an upright probe position. Note that the quaternion is given on the (x, y, z, w) form. Through empirical testing, the goal contact force in the z-direction was chosen as

$$f_{goal} = 5\,\mathrm{N}. \tag{4.4}$$

Since it is desirable to keep a constant contact force (i.e. no change in the force), the goal derivative contact force was set to

$$f'_{goal} = 0\,\mathrm{N/s}. \tag{4.5}$$

The goal mean velocity was chosen as

$$v_{goal} = 0.04\,\mathrm{m/s}. \tag{4.6}$$

All training was done on an AMD Ryzen Threadripper 3970X processor with a NVIDIA GeForce RTX 3090 and 128 GB RAM. Each model was trained until sufficient convergence was obtained. Essentially, each model was trained for 40 million timesteps, which would correspond to 40 000 episodes if none of the episodes terminated early. On average, it took 32 hours to train each model.

4.4 Model Overview

To summarize, a total of three different models were trained; a baseline model, a variable impedance model and an extended variable impedance model. The variable impedance models share the same control law u. The action space $\mathcal{A}$ is different for all three models, and the torques applied to the manipulator joints are given by (3.1).

As stated, the baseline model has no low-level control layer and outputs a desired force and a desired torque that is directly applied to the end-effector. The control law is given by (3.7) and the model's action space is defined in (3.8).

The variable impedance model uses the VICES as its low-level controller, making (3.2) the control law. The policy outputs the k_p gains used in the control law, and the model's action space is therefore given by (3.3).

The extended variable impedance model uses the same low-level controller as the previous model, where the control law is given by (3.2). The action space, however, is extended with an additional parameter Δ_z in (3.5).

Chapter 5

Results

This chapter presents the results from each of the three models, before they are quantitatively summarized in the last section. Each model was set to follow a predetermined linear trajectory with the following start point $\mathbf{p}_{\text{start}} = (0.062, -0.052, 0.896)$ and end point $\mathbf{p}_{\text{end}} = (-0.032, -0.075, 0.896)$. This corresponds to a trajectory that goes on a diagonal across the soft body. The code used to produce the results is found by following the link given in Appendix A.

5.1 Reinforcement learning

The training curves for the three models are shown in Fig. 5.1. The mean episodic reward of the three models are plotted against each other to allow comparison of sample-efficiency and performance.

Figure 5.1: Training curve for each of the three models. The curves show how the models learn as they are being exposed to more simulation data. The baseline model is used as a reference to quantify the effect of using an appropriate low-level controller.

Further, the variable impedance model was trained with both a full observation space and a reduced observation space, as presented in Section 3.4. More specifically, the reduced observation space does not include the difference between the running mean contact force and goal force, and the difference between the running mean velocity and goal velocity. The training curves resulting from changing the observation space are shown in Fig. 5.2.

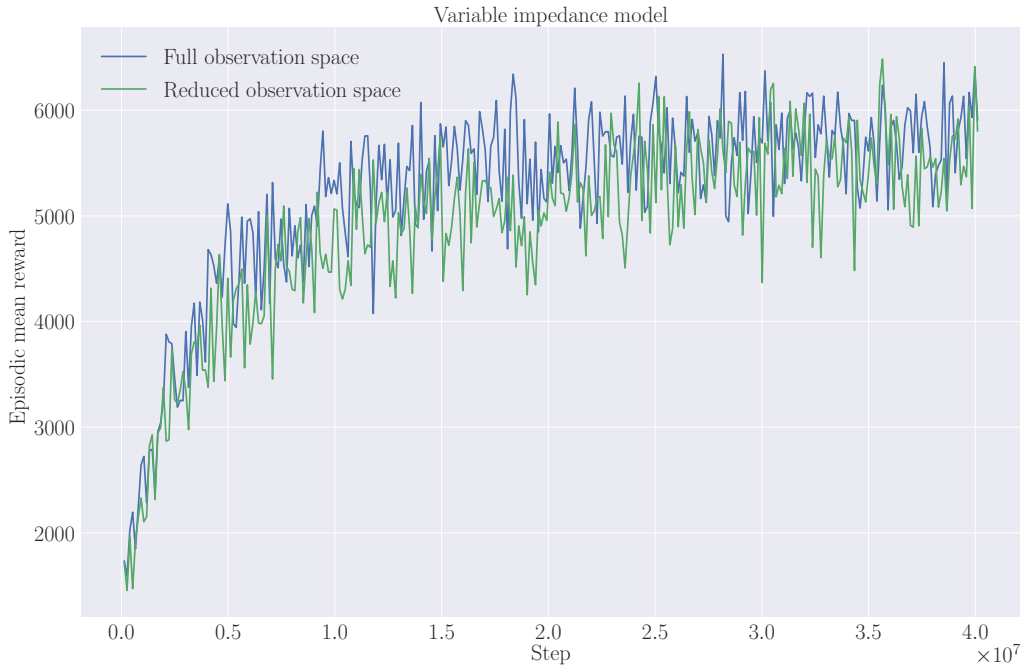
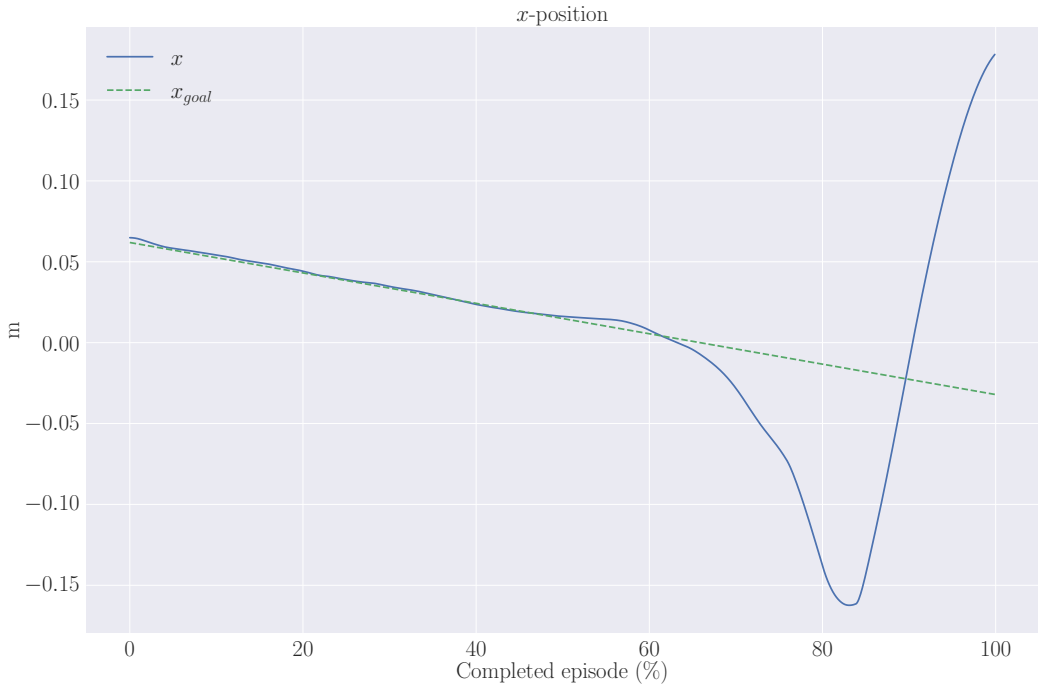


Figure 5.2: Training curves for the variable impedance model with full observation space and reduced observation space.

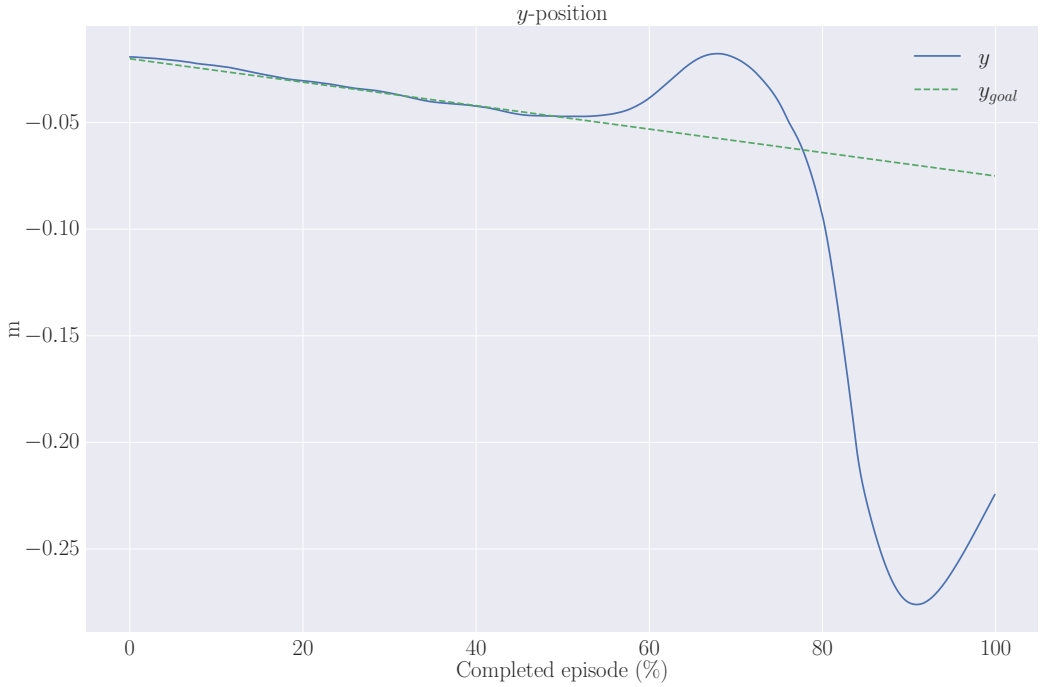
5.2 Baseline Model

As stated, the baseline model outputs a desired wrench, effectively ignoring the low-level controller layer. More specifically, the model's low-level control law is given by (3.7) and the model's action space is given by (3.8). The position tracking is shown in Fig. 5.3, and the orientation distance metric is shown in Fig. 5.4. The force tracking is displayed in Fig. 5.5, while the velocity tracking is shown in Fig. 5.6. The rewards are shown in Fig. 5.7 and the actions are shown in Fig. 5.8. Note that the behavior of the manipulator becomes unstable when approximately 60% of the episode is complete. The end-effector loses contact with the soft object, before slamming into it with an exaggerated force. A video showcasing the baseline model in action is uploaded to YouTube [1].

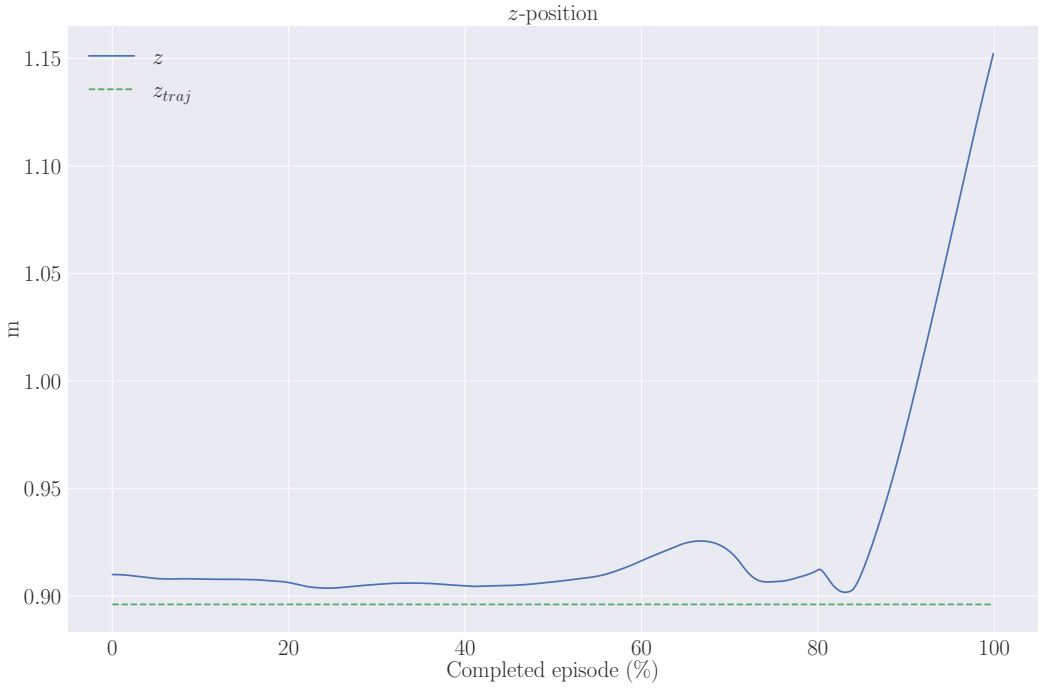
[1] `https://youtu.be/ntPEN4VkD3g`



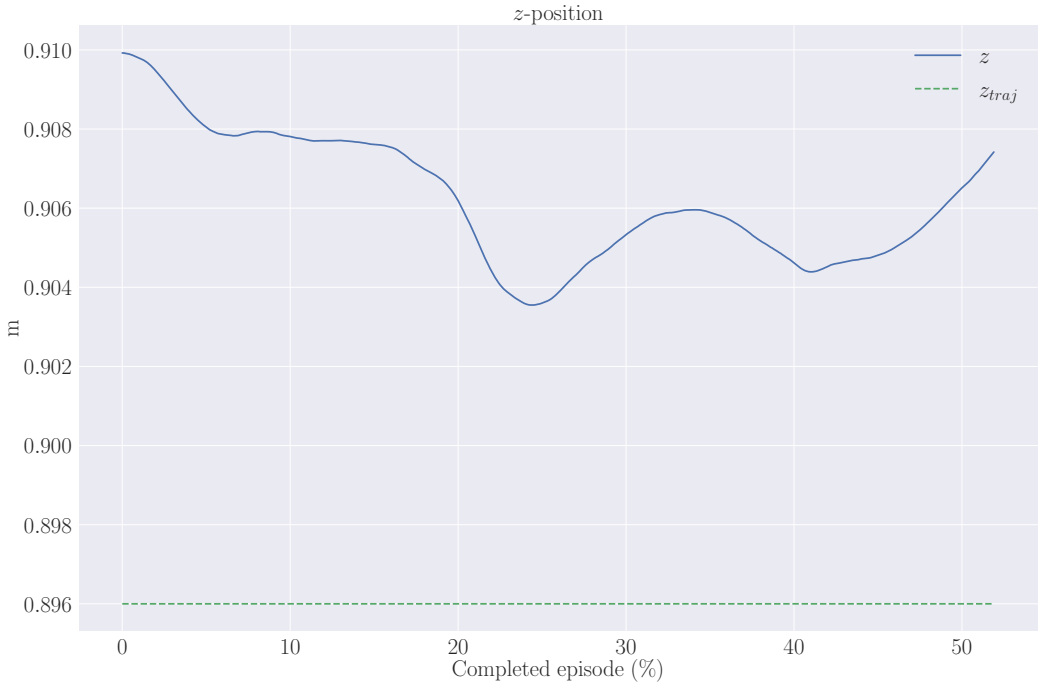
(a) x-position tracking.



(b) y-position tracking.



(c) z-position.



(d) Zoomed in z-position.

Figure 5.3: Position tracking for the baseline model. The goal position is marked by a dotted-line, while the end-effector position is represented by a fully drawn line. Note that there is no reward for tracking in the z-direction.

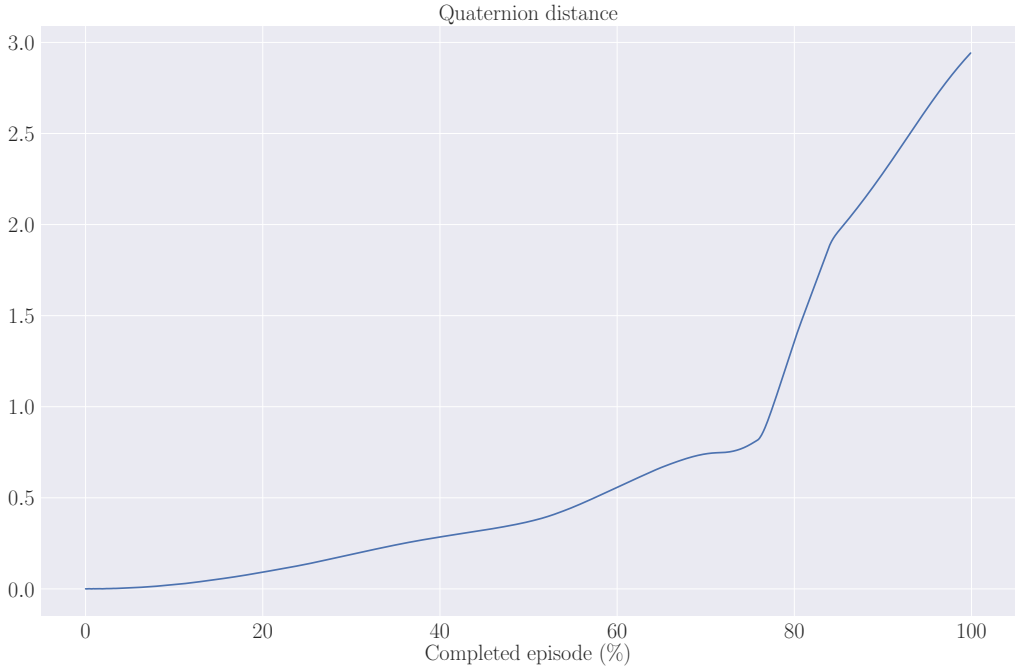
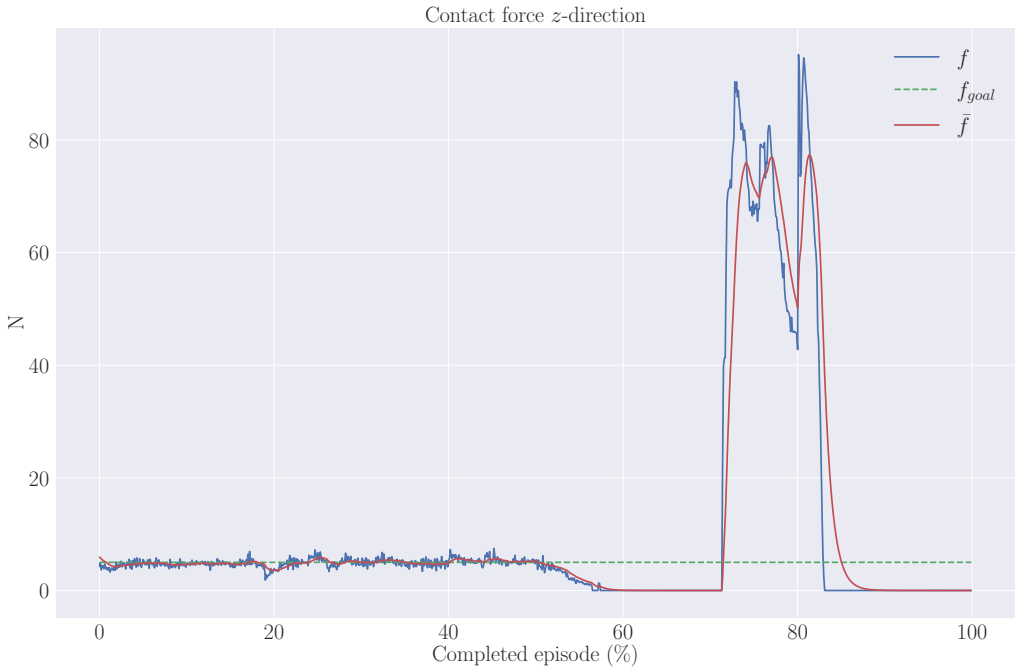
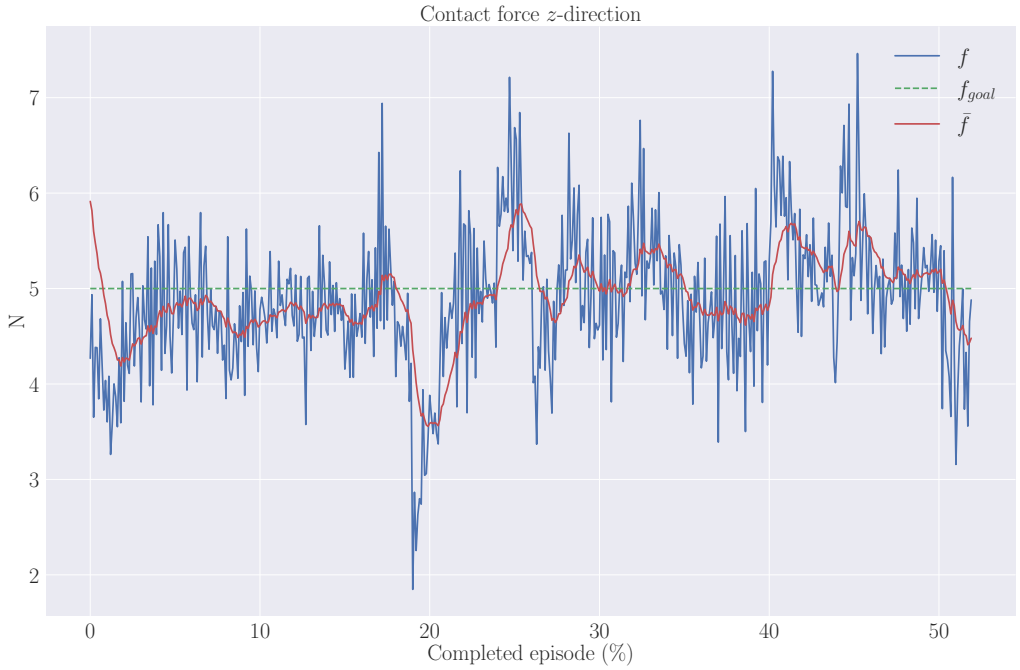


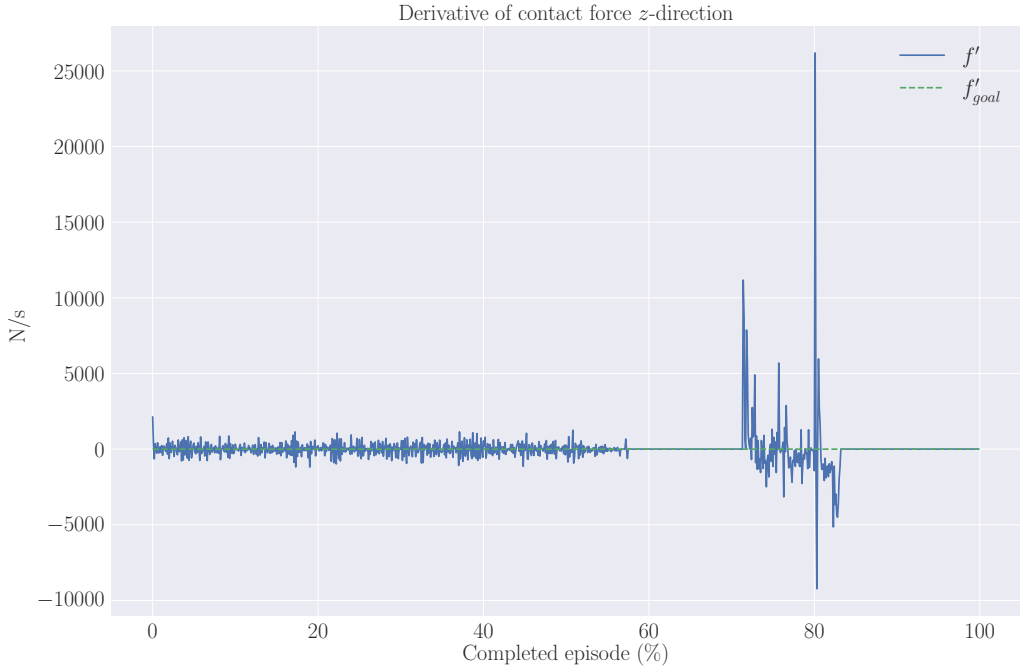
Figure 5.4: Quaternion distance between the end-effector orientation and the goal orientation for the baseline model.



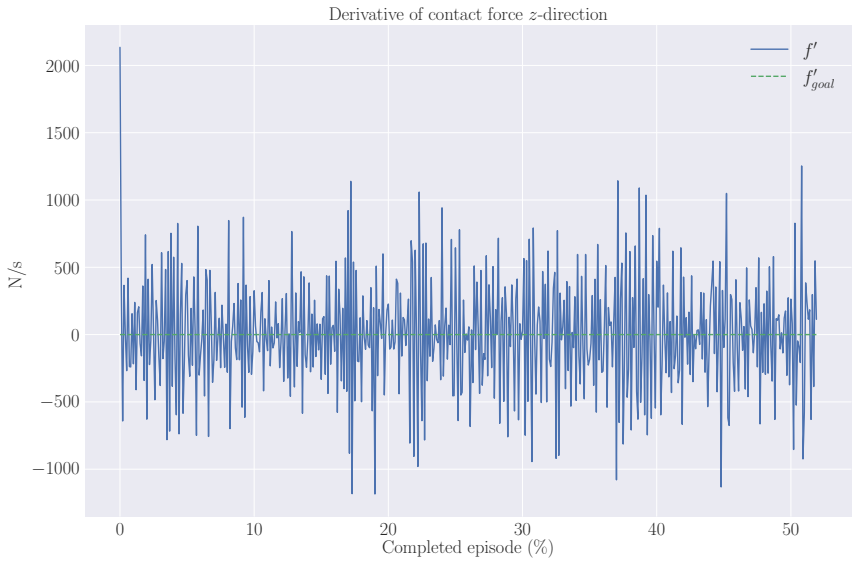
(a) Tracking of contact force in z-direction. Note that the tracking is based on the running mean $\bar{f}$ of the force.



(b) Zoomed in plot of the force tracking.



(c) Tracking of the derivative of the contact force in z-direction.



(d) Zoomed in plot of the derivative force tracking.

Figure 5.5: Force tracking for the baseline model. The goal force is marked by a dotted-line, while the end-effector measurements are represented by a fully drawn line.

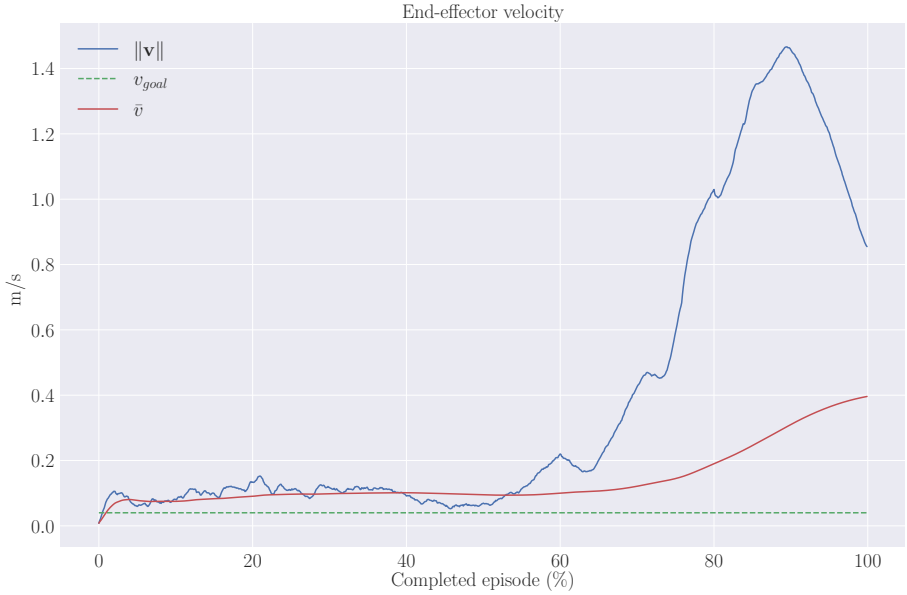


Figure 5.6: Velocity tracking for the baseline model. Note that the mean $\bar{v}$ is used for the tracking.

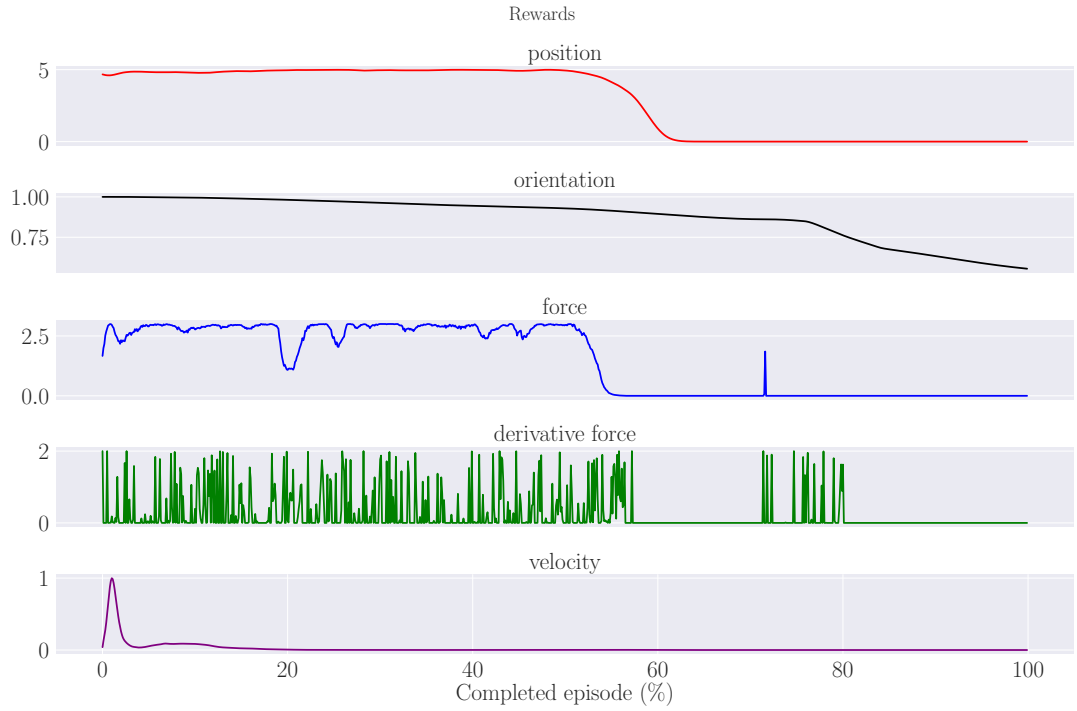
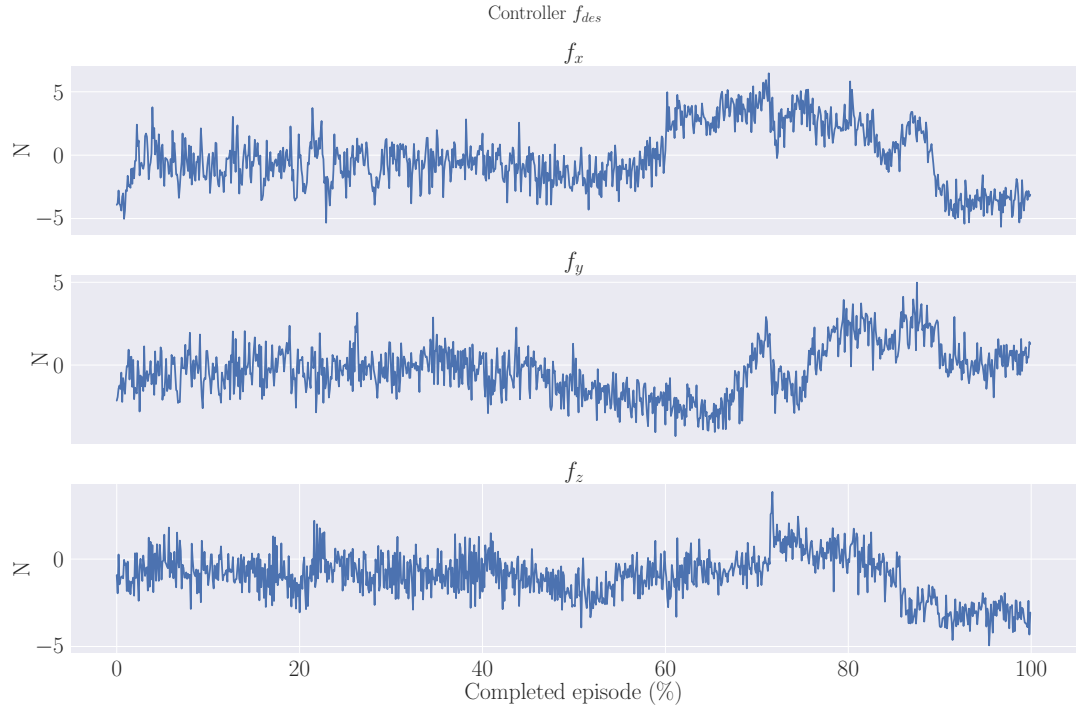
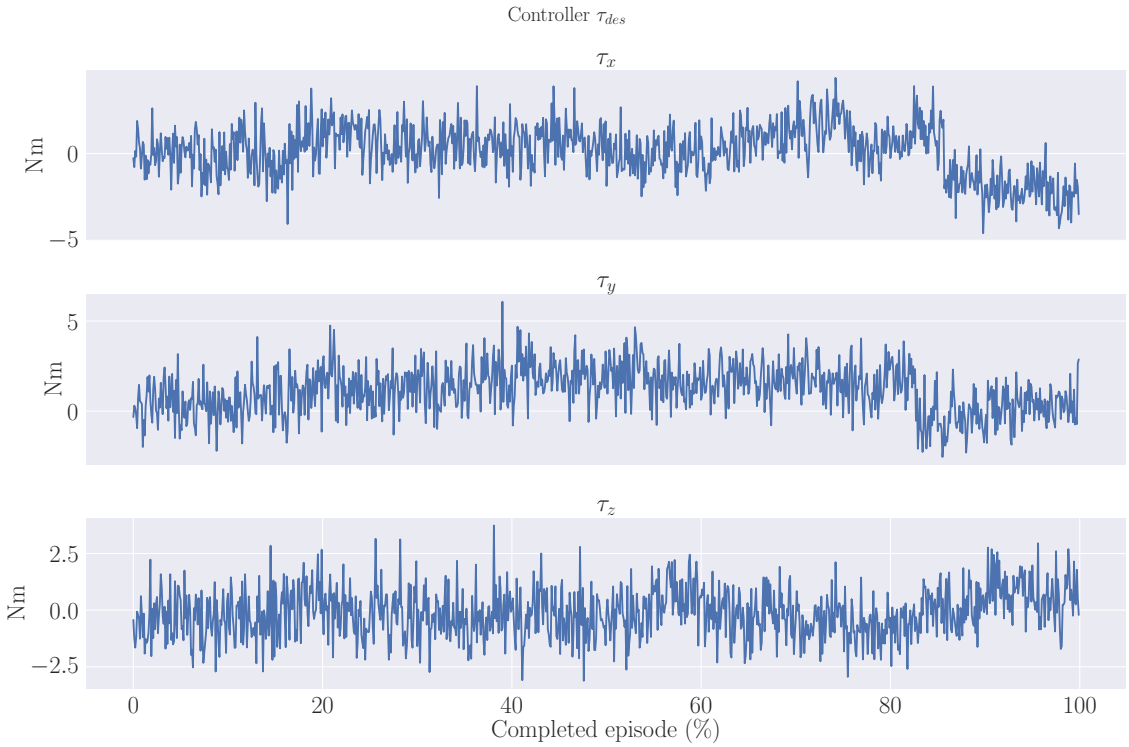


Figure 5.7: Rewards for the baseline model. The position graph corresponds to $w_p r_p$, the orientation chart corresponds to $w_o p_o$, the force and derivative of the force graphs correspond to $w_f r_f$ and $w_d r_d$ respectively, while the velocity graph corresponds to $w_v r_v$.



(a) Force actions.

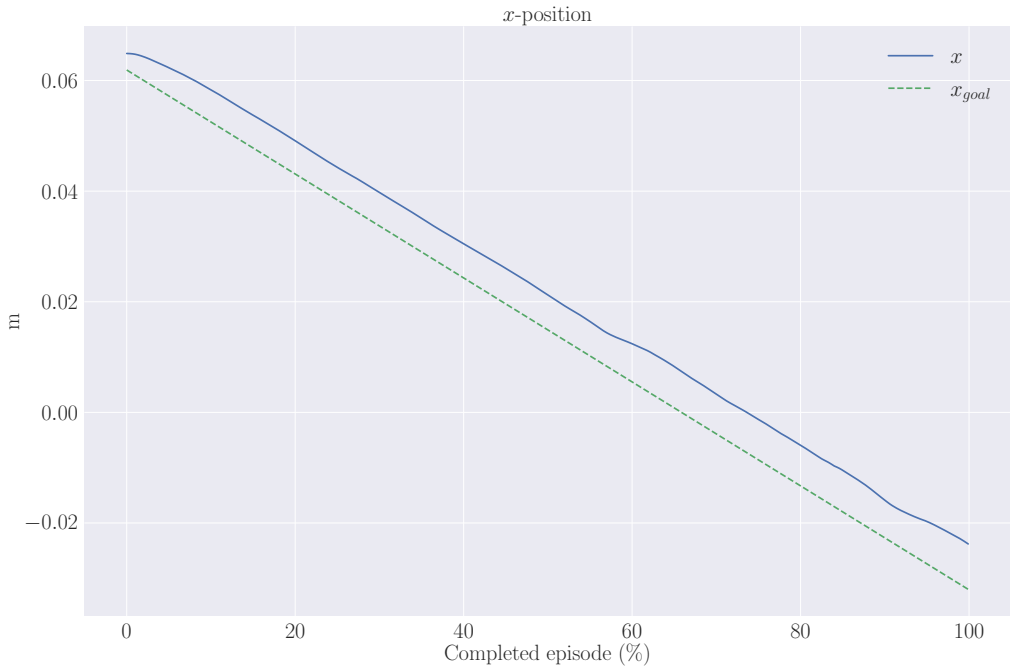


(b) Torque actions.

Figure 5.8: Actions chosen by the baseline model. The action space consists of a desired force and a desired torque in the x-, $y-$ and z-direction, respectively.

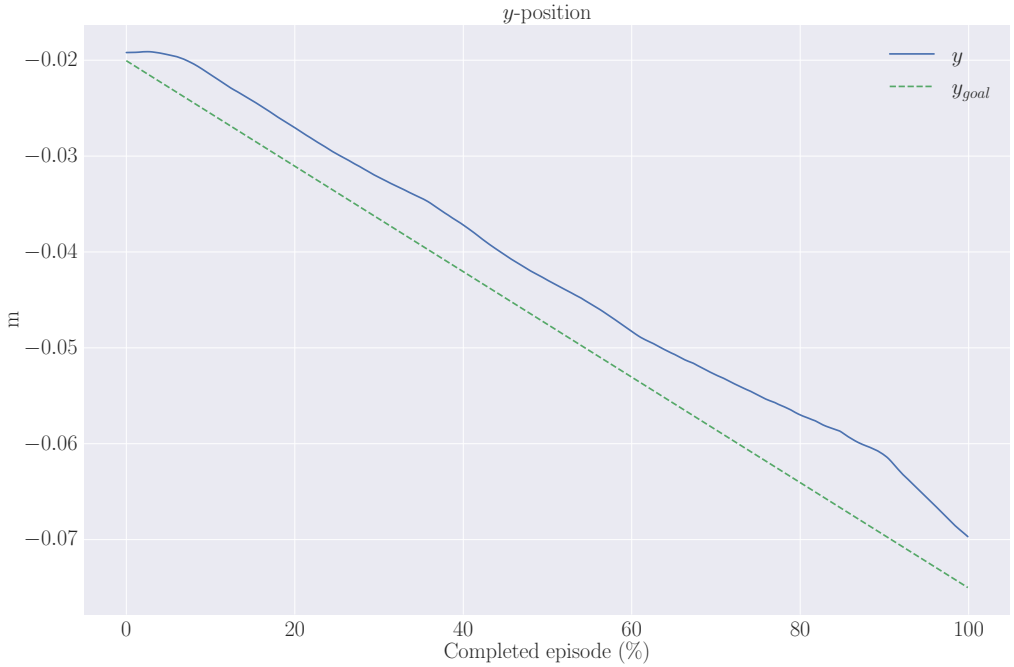
5.3 Variable Impedance Model

Recall that the model's low-level control law is given by (3.2) and the model's action space is defined in (3.3). The position tracking is shown in Fig. 5.9, while the quaternion distance metric used in (3.12) is displayed in Fig. 5.10. The force tracking is shown in Fig. 5.11, and the velocity tracking is presented in Fig. 5.12. The rewards are displayed in Fig. 5.13, and the actions chosen by the policy is shown in Fig. 5.14. A video of the model is uploaded to YouTube[2].

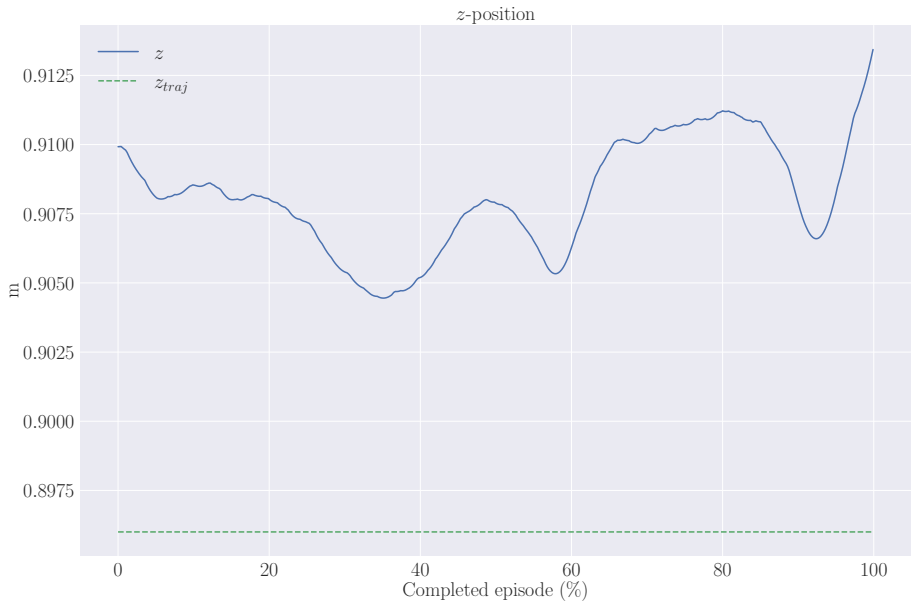


(a) x-position tracking.

[2] https://youtu.be/YRhWgWNj-pY



(b) y-position tracking.



(c) z-position.

Figure 5.9: Position tracking for the variable impedance model. The goal position is marked by a dotted-line, while the end-effector position is represented by a fully drawn line. Note that there is no reward for tracking in the z-direction.

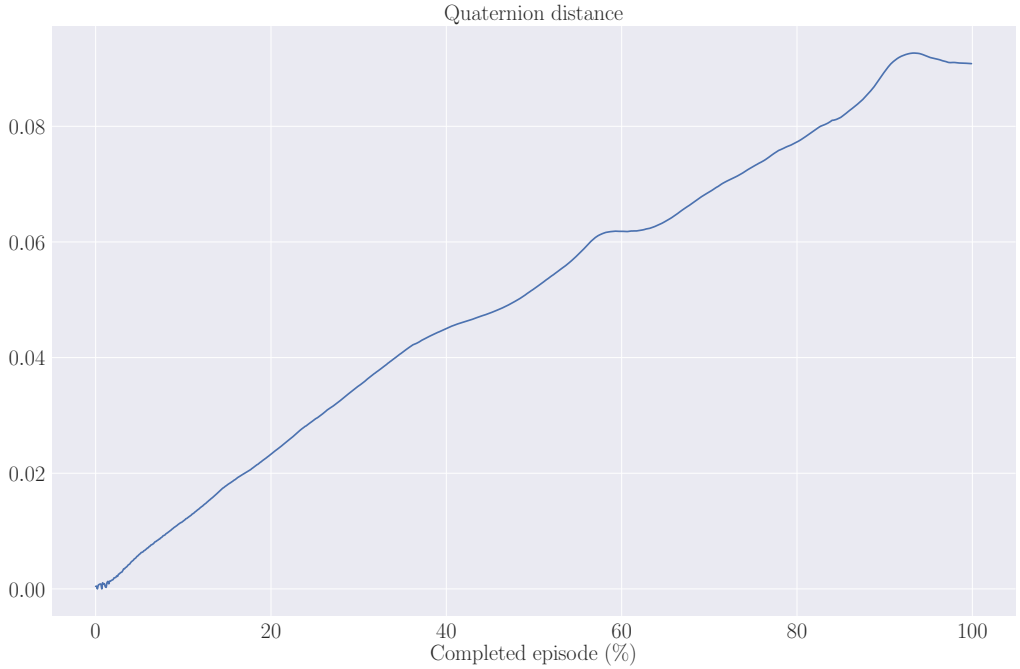
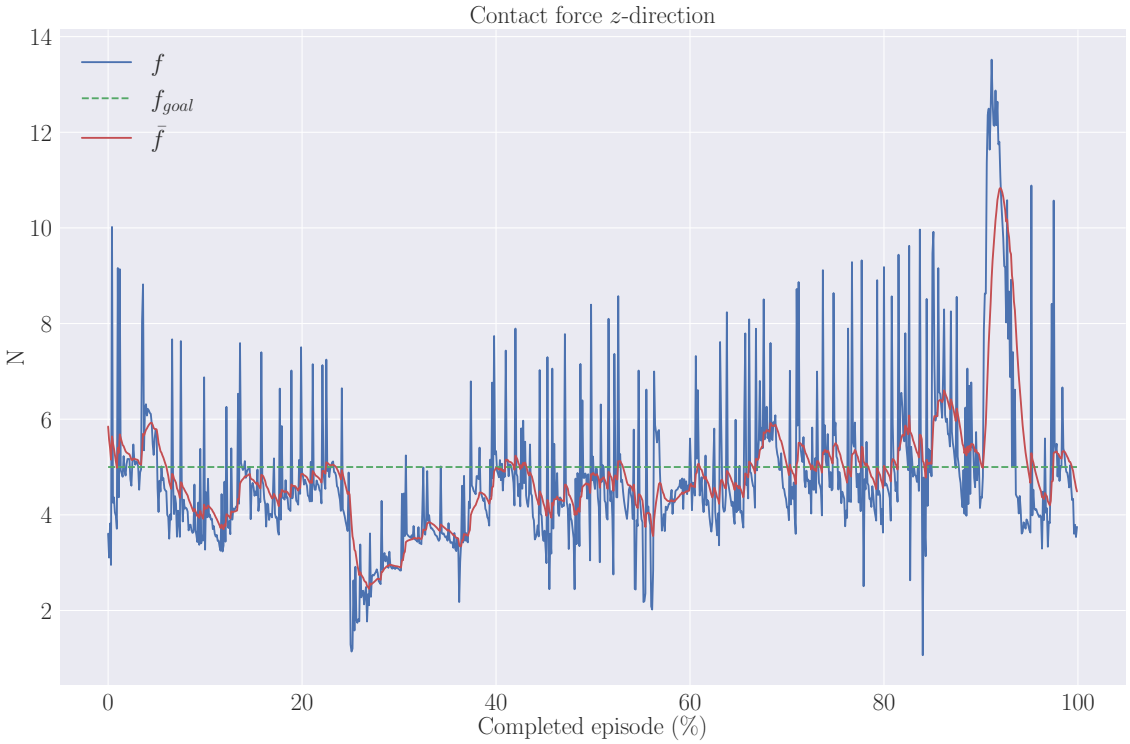
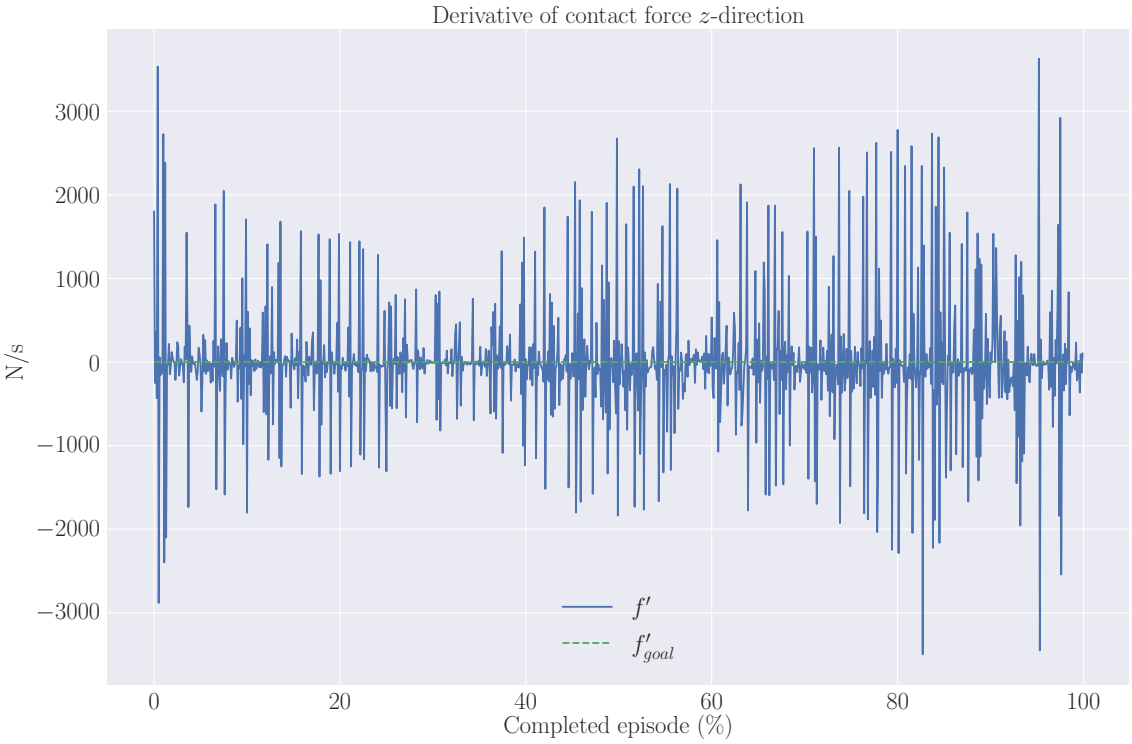


Figure 5.10: Quaternion distance between the end-effector orientation and the goal orientation for the variable impedance model.



(a) Tracking of contact force in z-direction. Note that the tracking is based on the running mean $\bar{f}$ of the force.



(b) Tracking of the derivative of the contact force in z-direction.

Figure 5.11: Force tracking for the variable impedance model. The goal force is marked by a dotted-line, while the end-effector measurements are represented by a fully drawn line.

Figure 5.12: Velocity tracking for the variable impedance model. Note that the mean $\bar{v}$ is used for the tracking.

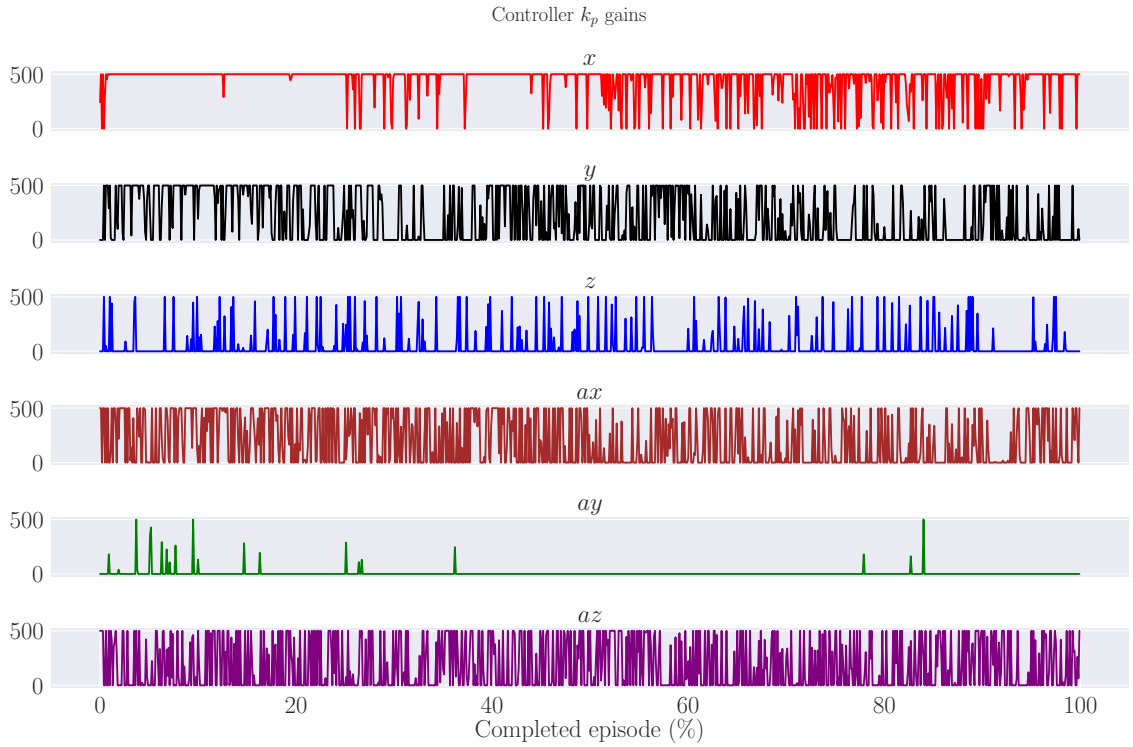


Figure 5.14: Actions chosen by the variable impedance model. The action space consists of six proportional gains. The first three actions corresponds to position tracking, while the last three actions correspond to orientation tracking. Note that the orientation tracking error is described by a set of axis-angles.

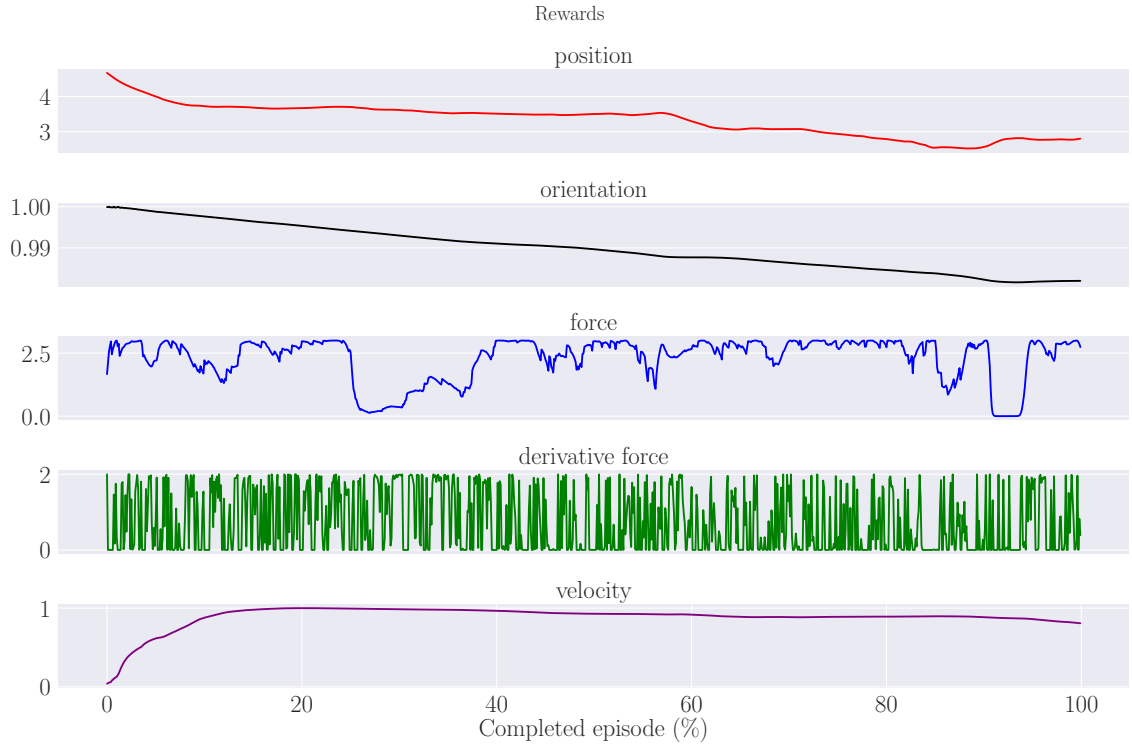
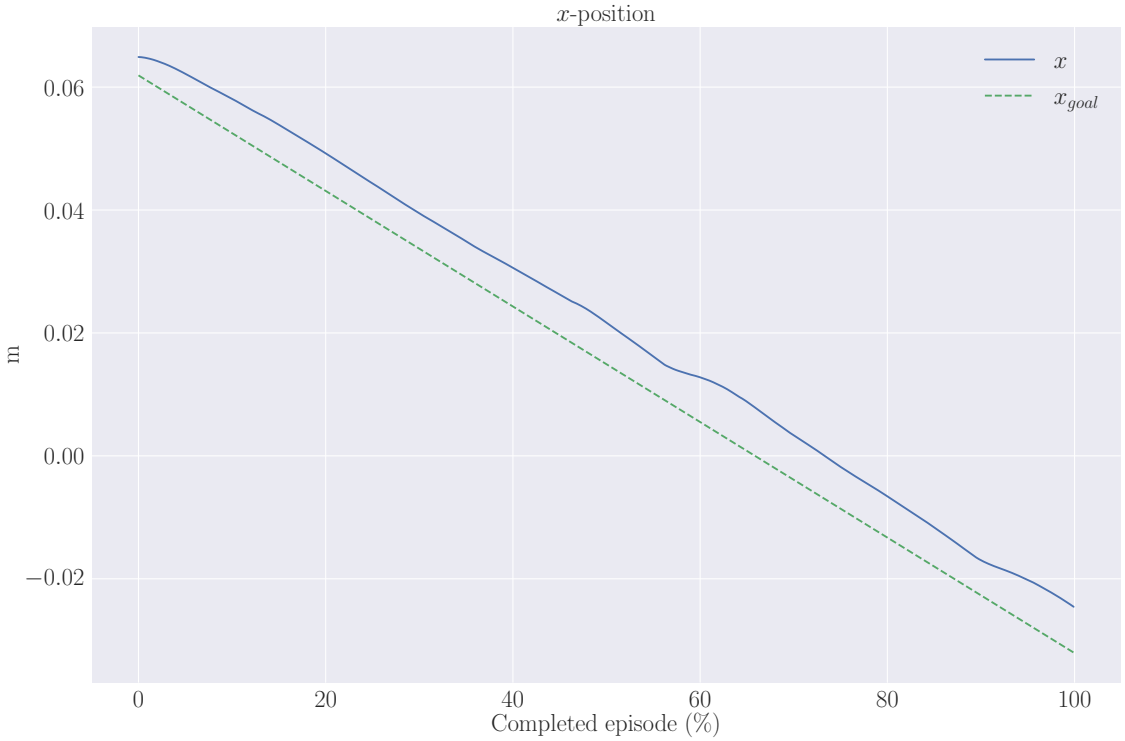


Figure 5.13: Rewards for the variable impedance model. The position graph corresponds to $w_p r_p$, the orientation chart corresponds to $w_o p_o$, the force and derivative of the force graphs correspond to $w_f r_f$ and $w_d r_d$ respectively, while the velocity graph corresponds to $w_v r_v$.

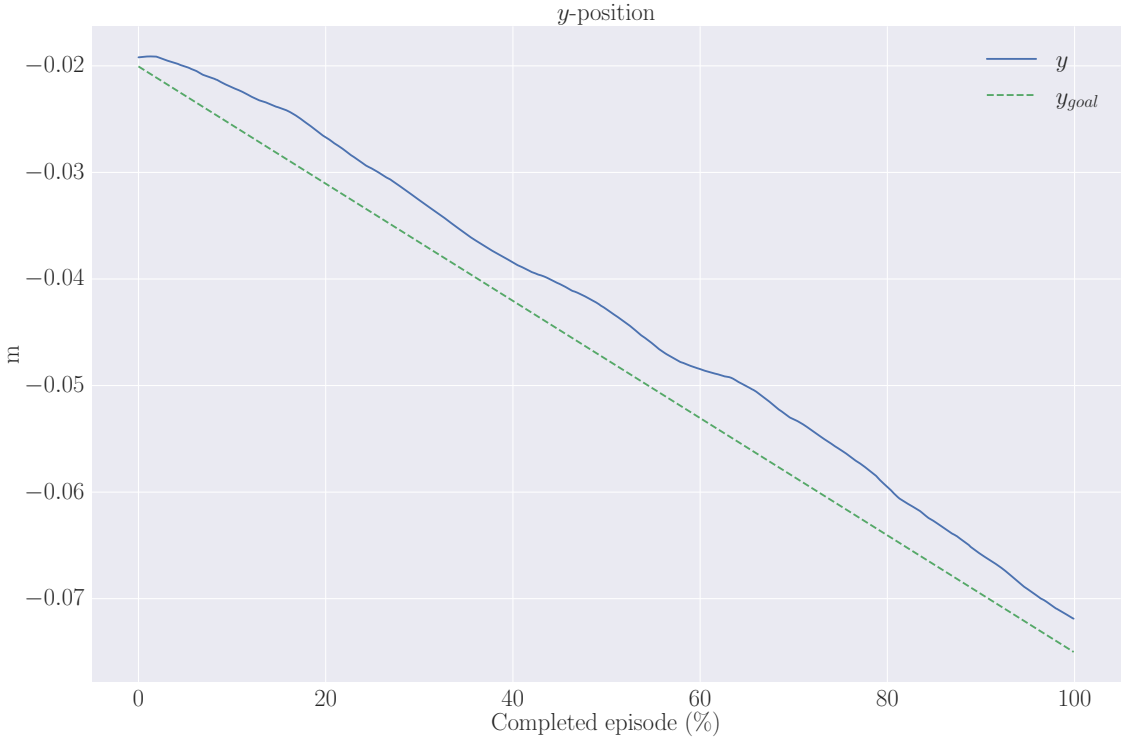
5.4 Extended Variable Impedance Model

As stated, the extended variable impedance model uses the same control law as the variable impedance model, given by (3.2). The action space, however, is augmented with a relative displacement in z-direction, and is defined in (3.5). Tracking of the position is shown in Fig. 5.15, and the quaternion distance between the end-effector orientation and the goal orientation is presented in Fig. 5.16. The force tracking is shown in Fig. 5.17, while the velocity tracking is displayed in Fig. 5.18. The rewards are shown in Fig. 5.19, and the chosen actions are shown in Fig. 5.20. A video of the model is available on YouTube[3].

[3] `https://youtu.be/8ad9QNuLBdI`

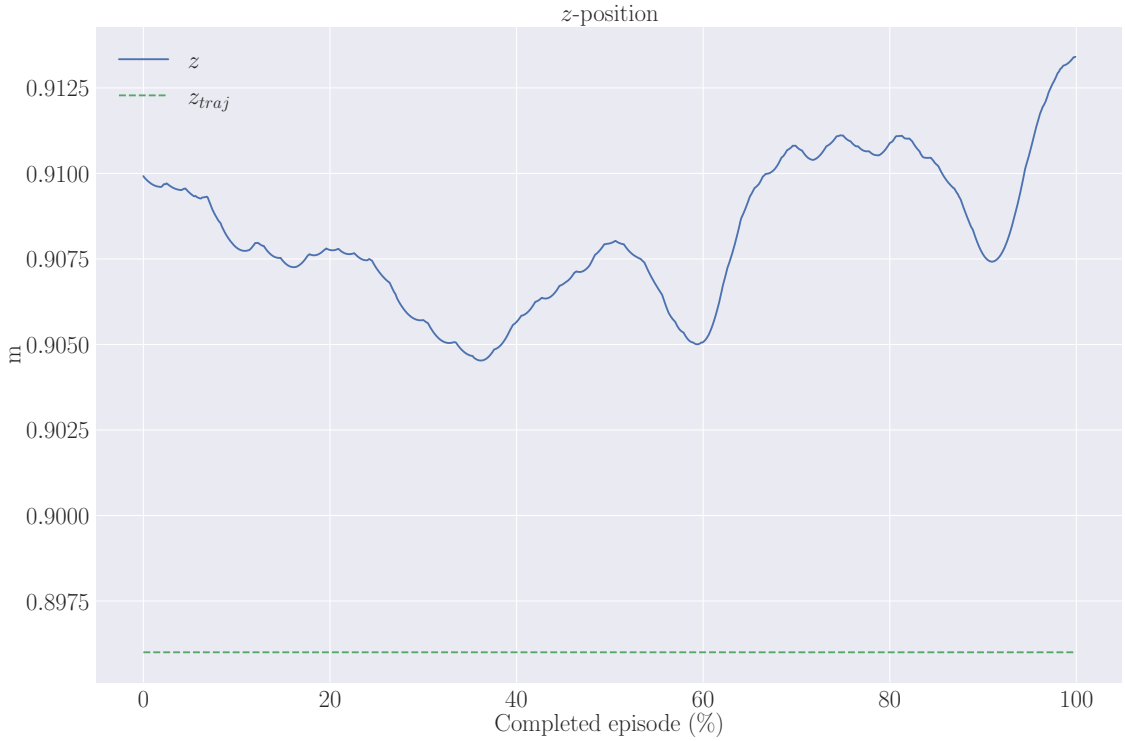


(a) x-position tracking.



(b) y-position tracking.

Figure 5.15: Position tracking for the extended variable impedance model. The goal position is marked by a dotted-line, while the end-effector position is represented by a fully drawn line. Note that there is no reward for tracking in the z-direction.



(c) z-position.

Figure 5.15: Position tracking for the extended variable impedance model. The goal position is marked by a dotted-line, while the end-effector position is represented by a fully drawn line. Note that there is no reward for tracking in the z-direction.

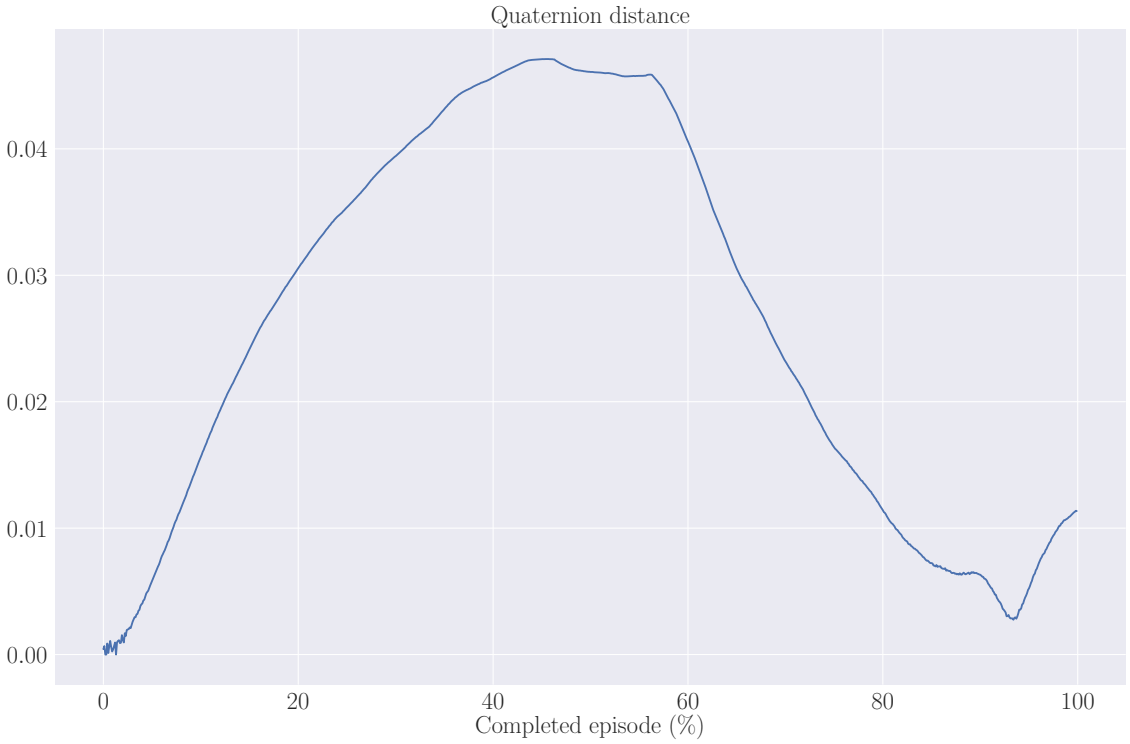
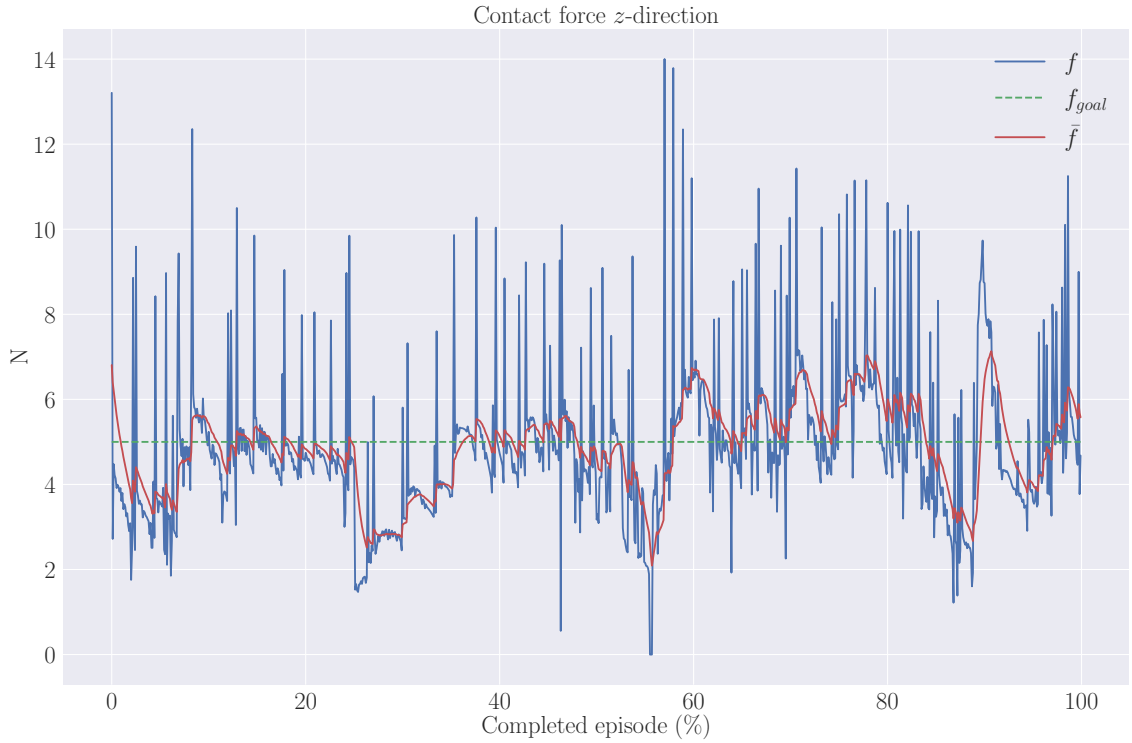
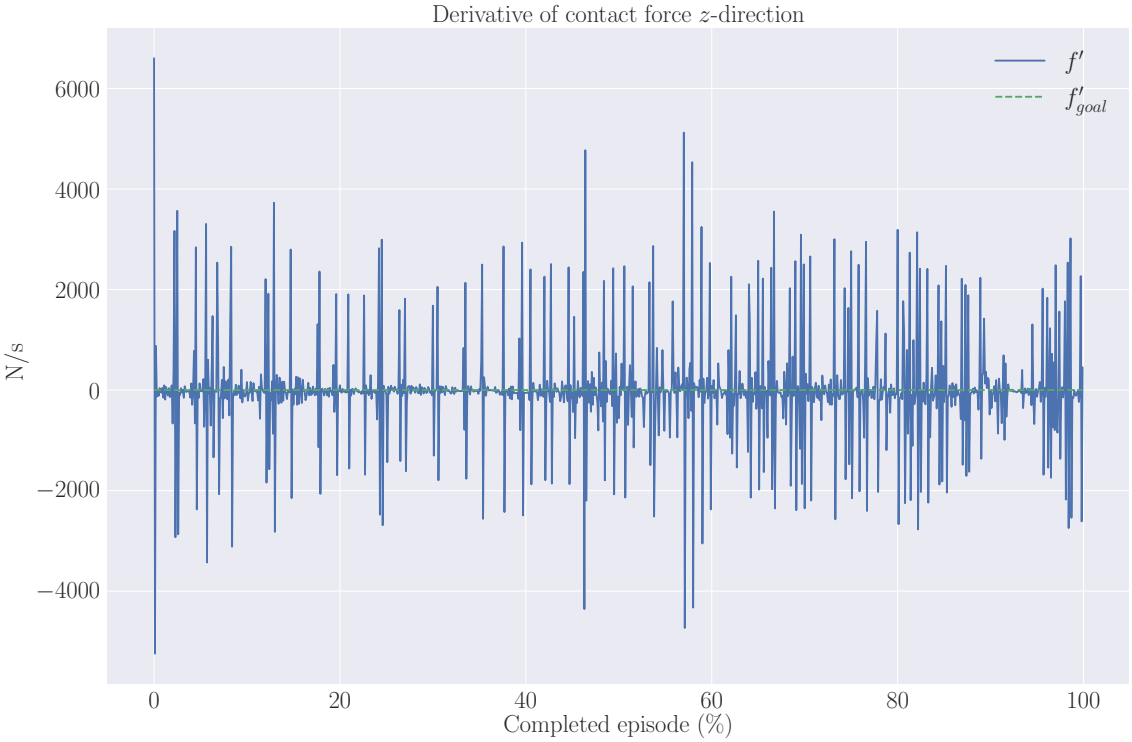


Figure 5.16: Quaternion distance between the end-effector orientation and the goal orientation for the extended variable impedance model.



(a) Tracking of contact force in z-direction. Note that the tracking is based on the running mean $\bar{f}$ of the force.



(b) Tracking of the derivative of the contact force in z-direction.

Figure 5.17: Force tracking for the extended variable impedance model. The goal force is marked by a dotted-line, while the end-effector measurements are represented by a fully drawn line.

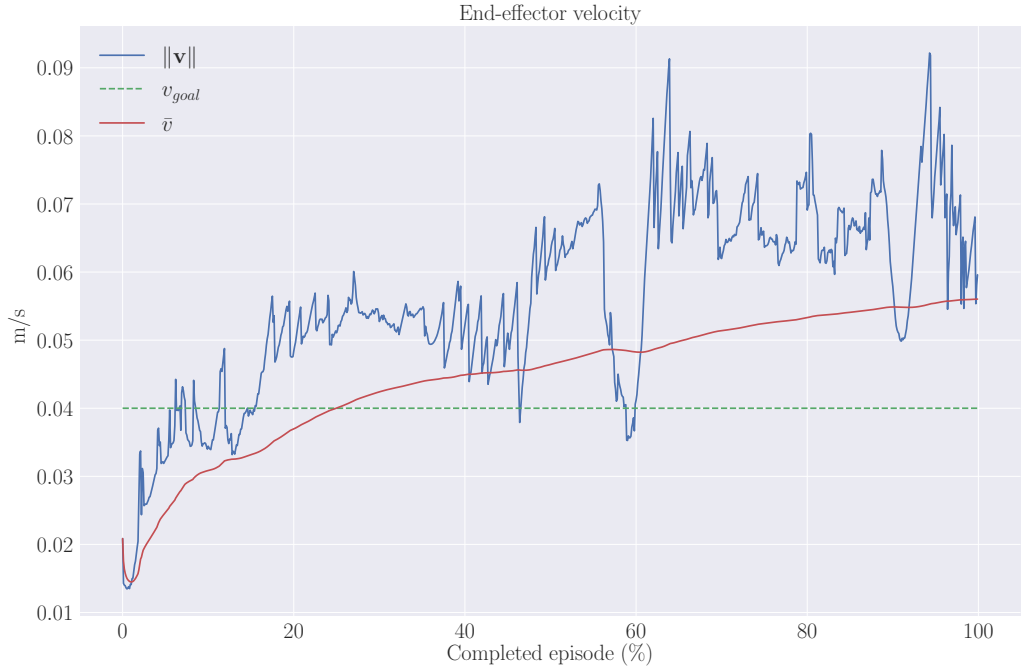


Figure 5.18: Velocity tracking for the extended variable impedance model. Note that the mean $\bar{v}$ is used for the tracking.

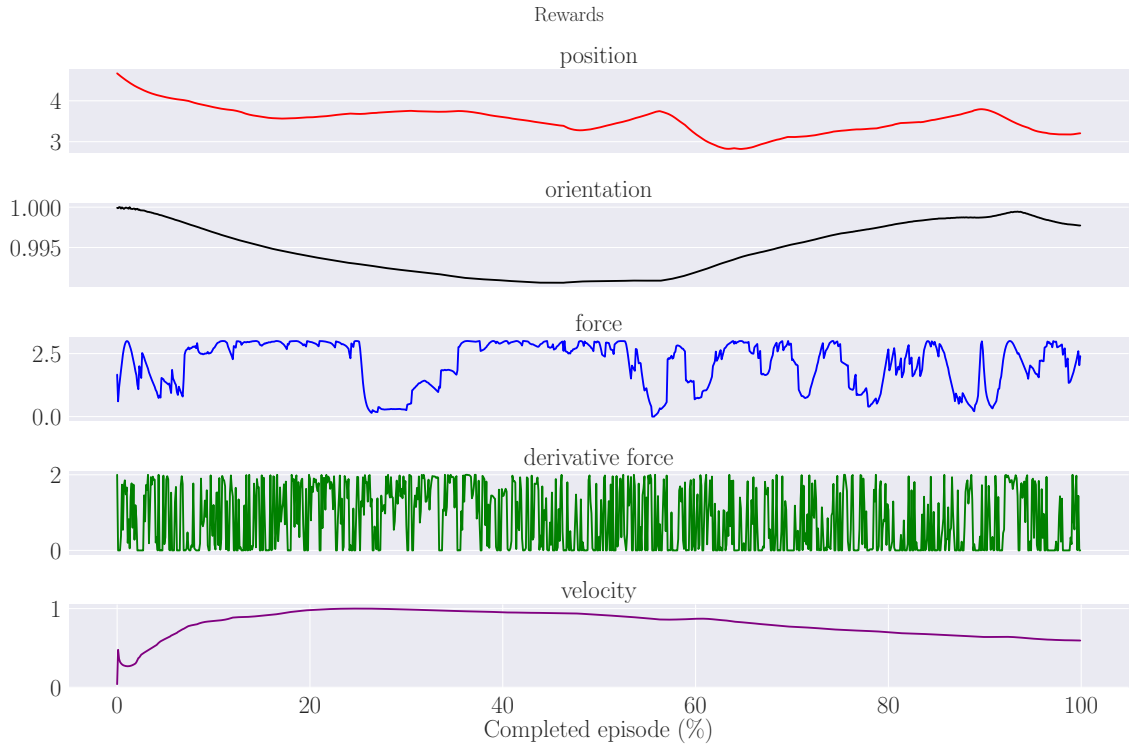
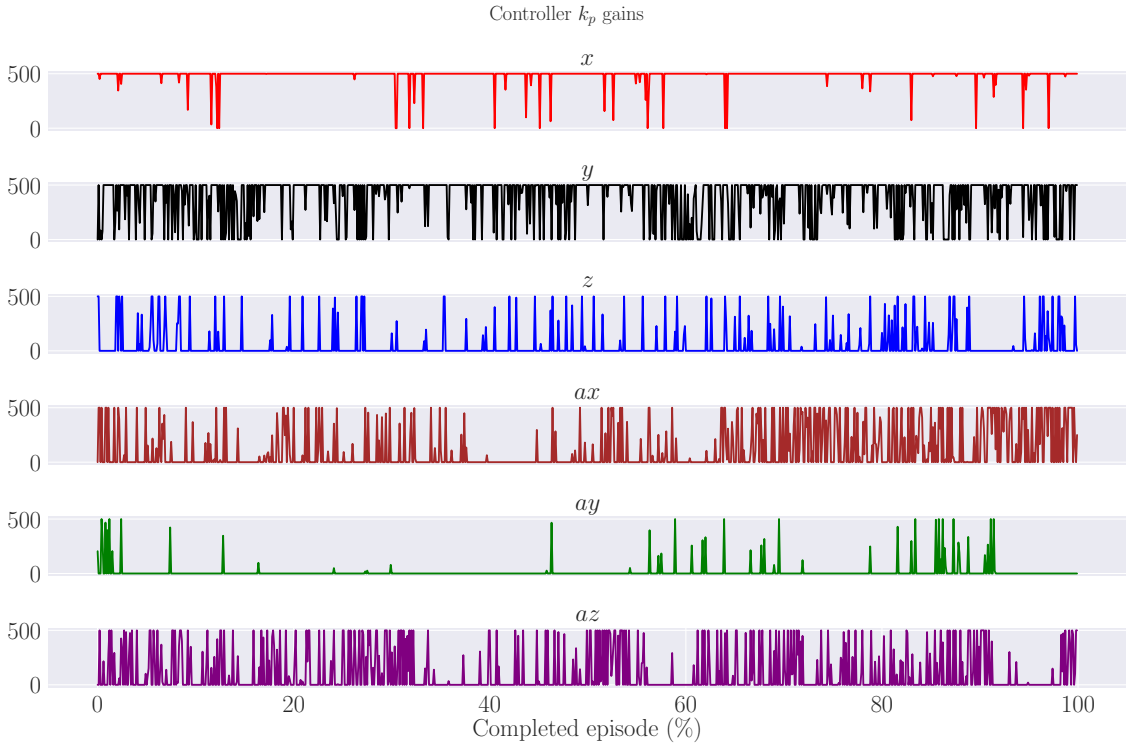
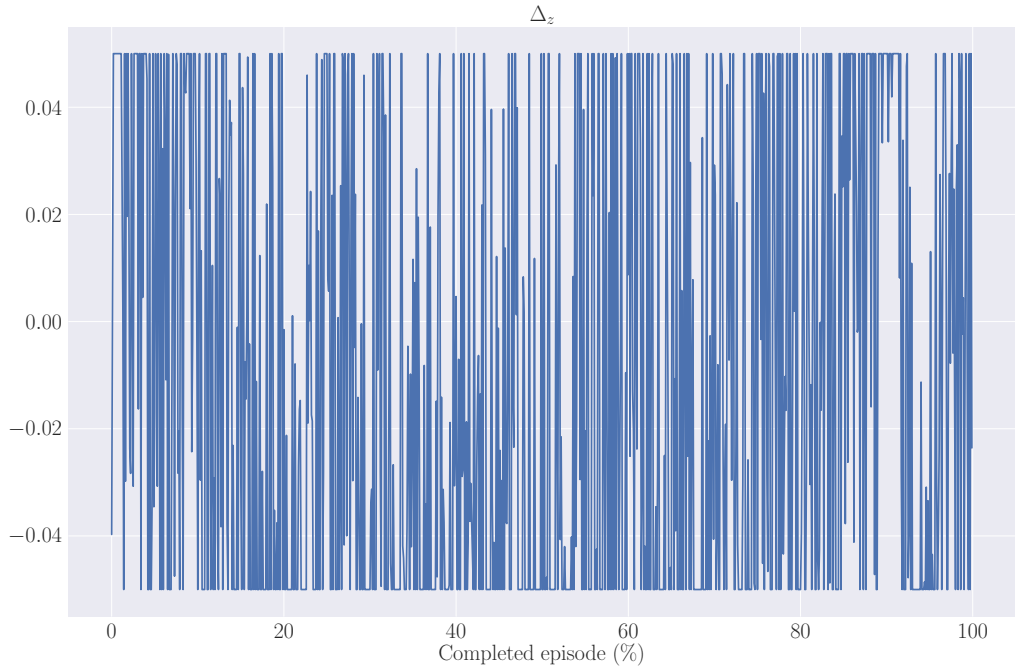


Figure 5.19: Rewards for the extended variable impedance model. The position graph corresponds to $w_p r_p$, the orientation chart corresponds to $w_o p_o$, the force and derivative of the force graphs correspond to $w_f r_f$ and $w_d r_d$ respectively, while the velocity graph corresponds to $w_v r_v$.



(a) Proportional gain actions.



(b) Relative displacement in z-direction action.

Figure 5.20: Actions chosen by the extended variable impedance model. The action space consists of six proportional gains and a parameter representing relative displacement in z-position. The first three proportional gains corresponds to position tracking, while the last three proportional gains correspond to orientation tracking. Note that the orientation tracking error is described by a set of axis-angles.

5.5 Summary Metrics

This section aims to quantify the results presented in the previous sections. The tracking errors, defined as the mean square error between the measurements and their corresponding goal values, are summarized in Table 5.1. The mean rewards for a timestep for each model is presented in Table 5.2. Recall that the maximum achievable reward for a timestep is 10.

Table 5.1: Tracking errors for all the three models; baseline (BL), variable impedance (VI) and extended variable impedance (EVI). The error is calculated as the mean square error between the measurements and their corresponding goal values. The orientation error, however, is calculated as the mean of the quaternion distance.

	x-pos	y-pos	Orientation	Force	Derivative Force	Velocity
BL	3.73e−3	5.91e−3	0.73	439.63	1.50e6	1.80e−2
VI	4.30e−5	2.86e−5	5.10e−2	1.33	**5.87e5**	**7.15e−5**
EVI	**4.09e−5**	**1.77e−5**	**2.63e−2**	**1.04**	1.11e6	1.13e−4

Table 5.2: Mean rewards for a timestep for all the three models; baseline (BL), variable impedance (VI) and extended variable impedance (EVI). The maximum total achievable reward is 10.

	Position	Orientation	Force	Derivative Force	Velocity	Total
BL	2.84	0.88	1.48	0.24	2.18e−2	5.46
VI	3.34	0.99	**2.31**	0.71	**0.89**	8.24
EVI	**3.53**	**0.99**	2.12	**0.80**	0.81	**8.25**

Chapter 6

Discussion and further work

This chapter is dedicated to discuss the performance of the reinforcement learning models. This includes comparing the models' training curves and tracking capabilities. The comparisons are done to investigate possible reasons behind the different model performances. Suggestions for improving the performances are also presented. Proposals for further development concludes the chapter.

6.1 Reinforcement learning

The increased performance gained by using a low-level controller is clearly shown in Fig. 5.1. The baseline model is only capable of achieving an episodic mean reward of 3500, slightly more than half of the episodic mean reward achieved by the models using a low-level controller. The models using a low-level controller also show better sample-efficiency, where they learn faster than the baseline model. During the first million training steps the models have a reward increase of approximately 3500. In contrast, the baseline model has a reward increase of only 1500 in the same training period.

A low-level controller introduces a simplified relationship between the action space and the robot dynamics, ultimately making it easier for the agent to learn. For instance, the controller used by the variable impedance models is in essence a PD-controller which tracks a reference position and orientation. For all the values in the action space, except $k_p = 0$, the controller automatically implements trajectory tracking. Hence, the agent does not need to learn the dynamics of trajectory tracking, it only needs to learn to find good k_p values for making the tracking optimal. The baseline model, however, has no low-level controller. In this case, the agent has to learn the force and torques needed to obtain the trajectory tracking, which is a far more complicated task compared to only learning gain values. Note that care should be taken when choosing a low-level controller and action space, as the performance and sample-efficiency of the agent is highly dependent on the mapping between the action space and the robot dynamics. The choice of action space and

controller is further discussed in [25] and [27].

Further inspecting Fig. 5.1, it can be seen that the variable impedance model and the extended variable impedance model converges to roughly the same episodic mean reward. Hence, the models have a similar performance. Note that it takes longer for the extended variable impedance model to converge. This is due to the extended action space, where the agent has an additional parameter to optimize. This showcases the importance of keeping the action space at a minimum. Adding parameters to the action space could potentially give the reinforcement learning algorithm more freedom in finding a better policy, but this comes at a cost of increased complexity. Ultimately, an action space with redundant or unnecessary parameters can lead to a decrease in sampling efficiency while not improving the agent's performance.

How the choice of the observation space affects the sample efficiency and performance of the variable impedance model, is shown in Fig. 5.2. It can be seen that the reduced observation space model consistently obtains lower rewards compared to the full observation space model, up until two million training steps are passed. After that, the models converge to approximately the same reward. Therefore, it can be concluded that adding observations which encompass the difference between the measurements and goal values can improve the learning speed, but not the overall performance of the model. Essentially, these observations are redundant as the information they convey can be deduced by combining information given by the other observations and the reward function. However, by including these observations, the training is simplified for the reinforcement learning algorithm. Instead of using computational power to learn intricate relationships between observations and reward functions, the algorithm can utilize this information directly through the added observations. As a result, the agent learns faster.

6.2 Position Tracking

Comparing Fig. 5.9 and Fig. 5.15, the position tracking for the extended variable impedance model and the variable impedance model is similar. Both models are capable of tracking the desired position in a satisfactory manner. However, both models also have a steady-state error, which is a consequence of the low-level controller choice. Recall that the controller for these two models is a PD-controller, and as as known, P- and PD-controllers are prone to steady-state deviations. The steady-state error could probably be reduced by increasing the maximum allowed k_p values, or removed completely by extending (3.2) with an integral term.

The compliant behavior of the controller is shown in Fig. 5.9c and Fig. 5.15c. As the probe sweeps across the soft object surface, the controller varies the z-position of the probe to dynamically compensate for the object's deformation. By closer inspection, the z-position of the extended variable impedance model in Fig. 5.15c

is slightly noisier than the z-position of the variable impedance model. This is due to the extended model being able to explicitly control the z-position through the parameter Δz. Rapid movements in the z-position, however, can prove itself disadvantageous when controlling the contact force, as the applied force is proportional to the acceleration of the probe.

As known, the baseline model has no low-level controller. The model outputs a desired wrench which is directly applied to the end-effector. Hence, the policy can be said to be in total control of the end-effector. As an effect, the position tracking has no steady-state error, which is shown in the first parts of Fig. 5.3. The z-position is also smoother compared to the variable impedance models, which can be seen in Fig. 5.3d. Potentially, this can yield better force tracking.

Unfortunately, the baseline model outputs actions resulting in unstable behavior after completing approximately 60% of the episode. Looking at Fig. 5.3, the chosen actions yield large deviations in both x- and y-position tracking. It also causes the end-effector to lose contact with the soft body. This shows one of the greatest disadvantages of not having a low-level controller. Since the policy is practically in full control of the end-effector, it is simple for the policy to give actions resulting in instability. A low-level controller and a carefully designed action space, on the other hand, will work as a protective layer between the policy and the manipulator, essentially ensuring stability. For instance, it is not possible for the variable impedance model to make the manipulator unstable when the policy only outputs k_p values in a range where the PD-controller is ensured to yield stability.

6.3 Orientation Tracking

Comparing Fig. 5.4 and Fig. 5.10, before the baseline model becomes unstable, the baseline model and the variable impedance model track the orientation fairly well and have similar orientation errors. This shows that it is possible to achieve orientation tracking without the need of a low-level controller. As seen in Fig. 5.16, the extended variable impedance model's orientation tracking is moderately better compared to that of the regular variable impedance model. A clear reason for the superior performance is unknown.

6.4 Force Tracking

As with the position tracking, the force tracking for the variable impedance model and for the extended variable impedance model, shown in Fig. 5.11 and Fig. 5.17 respectively, are closely resembled. Generally, both models are able to keep the mean force in a tight interval around the desired force. Unfortunately, both models are also prone to noisy force measurements, characterized by large spikes in Fig. 5.11a and Fig. 5.17a. The occurrences of the force spikes corresponds well with spikes in

the force derivatives shown in Fig. 5.11b and Fig. 5.17b, as expected.

Note that the force tracking of the extended variable impedance model has sparse, but larger force spikes compared to the force tracking of the regular variable impedance model. The increased magnitude of the spikes can be a direct consequence of the more rapid movements in z-direction resulting from the model being able to control the z-movement directly with the action Δz. Smoothening out the z-direction movement for both models could therefore potentially reduce the force noise. Reducing the maximum allowed k_p value in z-direction can be a possible way to achieve smoother movements. Lower k_p values generally yields longer response times for controllers, ultimately making the movement slower, but smoother.

Before the unstable behavior, the baseline model achieves superior force tracking compared to the other two models. As seen in Fig. 5.5b, the noisy spikes has a smaller magnitude and the mean force is in a tighter interval around the desired force. Recall that the movement in z-direction was much smoother for this model, strengthening the hypothesis that the rapid movements in z-direction is the cause for the noisy force measurements.

6.5 Velocity Tracking

Again, the performance of the variable impedance model and of the extended model share similarities. Both the models yield a mean velocity that lies over the desired velocity, as seen in Fig. 5.12 and Fig. 5.18. However, it should be noted that the variable impedance model stabilizes at a velocity closer to the goal, compared to the extended model. The extended model also has more variance in its velocity measurements, which could be related to the rapid movements in the z-direction. Similarly to the other two models, the baseline model also stabilizes at a velocity above the desired velocity, before yielding unstable behavior, as shown in Fig. 5.6.

Attention should be brought to the unwanted coupling between the end-effector velocity and the desired pose given by the trajectory generator. At each timestep, the controller gets a new desired position and orientation from the trajectory generator. However, if the end-effector is able to reach the desired position during the simulation timestep, the maximum velocity of the end-effector is implicitly given by the trajectory. Phrased differently, if the agent wants to achieve optimal position tracking, the desired velocity of the end-effector is defined by the distance between the current trajectory point and the next trajectory point, divided by the timestep. Possibly, the consequence of this is seen in Fig. 5.12, Fig. 5.18 and Fig. 5.6, where the velocity stabilizes at a velocity different from the goal velocity. To quantify how the coupling effects the velocity of the end-effector, the model could be trained with and without the velocity reward term. If the agent's resulting velocity is independent of being trained with a velocity reward term, the velocity is most likely decided by the desired pose coupling.

6.6 Reward Function

As stated, the reward function weights and error multipliers were empirically chosen. The weights were chosen based on how important each term was for the completion of the task, while the error multipliers should be chosen such that the reward terms were neither too sensitive nor too unresponsive. A reward term that is too sensitive or too unresponsive will make it difficult for the agent to learn. Since the reward function design was empirical, it potentially exists a better choice of weights and error multipliers. Hence, it is fair to assume that the reward function is suboptimal. Finding a better reward function design could potentially increase the performance of the models.

Looking at Fig. 5.7, Fig. 5.13 and Fig. 5.19, the reward function performs fairly well. Excluding the force derivative, the reward terms are smooth and corresponds satisfactorily with the scale of the errors. Large errors yield small rewards and vice versa. The performance of the reward function is also illustrated by the training curves in Fig. 5.1, where the agents are able to learn quite effectively.

As mentioned, the derivative force reward term is noisy, which is unwanted as it makes it complicated for the agent to learn. It is difficult to make this graph smooth, since the derivative force measurement itself is extremely noisy. To smooth out the graph, it could be beneficial to decrease the error multiplier c_d. Then, the error term (3.17) would be less sensitive to changes in the force derivative. Another possibility is to redesign the force reward term. For instance, by stating that no force reward should be given if the force measurement is outside a small interval around the desired force, the agent might learn to apply a smoother contact force without spikes. In this case, the derivative force term would probably be obsolete. The decay factor in (3.16) can also be tuned. By increasing this factor, more weight will be put on the previous force measurement, ultimately reducing the smoothening effect of the mean. The mean will in this way capture more of the force spikes, and the agent would need to smoothen out the force measurement in order to get rewards.

6.7 Policy Actions

Generally, policy actions are somewhat noisy due to the exploratory nature of reinforcement learning algorithms, where the algorithms wishes to search the whole action space to find a good policy. In addition, the algorithms often contain stochastic components, resulting in non-deterministic actions.

The k_p gains output by the variable impedance model in Fig. 5.14 and Fig. 5.20a are generally quite noisy. Both the models output high gains in x- and y-direction, which is necessary to achieve position tracking. The gain in z-direction, however, is mostly zero with some occasional spikes. For the variable impedance model, these spikes correspond to movement in z-direction. Looking at the 90% completed

episode mark in Fig. 5.14 and Fig. 5.9c, it can be seen that the z-position moves towards the trajectory position as the gain value spikes. This is expected as a higher gain will lead to a low-level controller input which reduces the error between the current position and the reference position. For the extended model, the z-position movement is a combination of the gain value and the Δz action in Fig. 5.20b.

Similarly, the force and torque output from the baseline model is also noisy, as shown in Fig. 5.8. Looking at the f_z component, it is easy to see the relationship between the force and the movement of the end-effector. When the z-force is negative, the end-effector moves higher up (i.e. larger z-position value). The same type of relationship applies for the movement in x- and y-direction as well. The torques, on the other hand, control the orientation of the end-effector. Inspecting Fig. 5.8 more closely, a jump in the action values can be seen as the manipulator behavior becomes unstable. As to why the policy chooses actions that results in instability, is a combination of the reward function, observation space and training setup.

6.8 Summary Metrics

Table 5.1 summarizes the results presented in Chapter 5. From the training curves in Fig. 5.1, it was expected that the variable impedance models would perform better than the baseline. In Table 5.1, it is shown that the variable impedance models achieve significantly smaller tracking errors compared to the baseline model. Overall, the extended variable impedance model can be said to perform best, where it is only beaten by the regular impedance model for derivative force tracking and velocity tracking.

The mean rewards presented in Table 5.2 corresponds well with the errors in Table 5.1, where smaller errors result in larger rewards. The only discrepancies are for the force tracking and the force deriavtive tracking. Here, the extended variable impedance model achieves the smallest tracking error, but receives less reward than the regular variable impedance model. This is due to the design of the reward function, and the choice of error multiplier specifically. Comparing Fig. 5.11a and Fig. 5.17a, the mean force for the variable impedance model generally lies closer to the desired force with larger spikes occurring sporadically. The spikes in the mean will result in a higher tracking error, but since the mean generally lies close to the desired force, the model will accumulate more rewards.

The variable impedance model and the extended variable impedance model achieves the same total mean reward for a timestep, as seen in Table 5.2. As the extended model has an additional action space parameter, one might think that this would enhance the performance. According to Table 5.2, this is not case. Even though the additional action space parameter gives the agent more freedom in finding a better policy, the extended model's overall performance is not improved compared to the regular variable impedance model. This might be due to the coupling between

the additional action space parameter Δz and the k_p gain in z-direction, where both parameters control the movement in z-direction. By choosing the k_p gain, the model might already have sufficient control of the movement in z-direction, and adding the Δz parameter could therefore be redundant.

It should be noted that the performance of the models are dependent on the chosen trajectory. Due to how the grid of rigid bodies are defined for the soft object, there are some part of the object that are behaving softer than others. At these parts, the models will often need to push the end-effector further into the soft body to apply a sufficiently large force. When the models then try to simultaneously track the trajectory position by across the surface, the probe will often get stuck on the soft body. This phenomenon can probably be reduced by tuning the physical properties of the body, or reduce the friction between the body and the probe.

6.9 Related work

Similarly to results achieved in related work, superior performance is obtained when learning variable impedance in task space. Compared to directly applying a desired wrench in task space, the variable impedance controller proposed in [25] gives robust performance, safe exploration of the environment and sample efficiency when training a policy. It should be noted that the baseline model showed better tracking capabilities before yielding instability, illustrating some of the behavior limiting disadvantages of using the variable impedance controller. Inspired by [26, 27], it would be interesting to quantify the model behavior of other action space and controller combinations and see how the choice of the action-space-controller pair affects performance.

Overall, the models struggle to track the desired contact force, where the applied contact force is noisy. This might be due to the lack of explicitly implemented force control. In the framework, force control is implicitly achieved by varying the impedance. In order to improve the force tracking capabilities, it might be worthwhile to adopt some of the techniques proposed in [24]. Concatenating the force/torque information in the second last layer of the actor's neural network was said to mitigate random motions, as opposed to directly treat the force/torque readings as an input to the first layer of the network. This could potentially smooth out the movement in z-direction, yielding a less noisy applied contact force. It could also be beneficial to introduce direct force control, by implementing the proposed hybrid motion/force controller.

6.10 Further Work

The previous sections discussed the model's tracking capabilities, and also included some suggestions on how to improve their performance. The following subsections

are restricted to present feasible extensions and improvements to the overall framework and simulation environment.

6.10.1 Closing the reality gap

As mentioned, reducing the reality gap is important for making the transfer from simulation to reality as effective as possible. In the project thesis [32], a set of stiffness and damping parameters from a real-life phantom torso were quantified through a calibration task. However, the physical properties of the soft body object in the simulation environment is dependent on more than only these parameters. The behavior of the soft body is also decided by the constraint impedance of the joints connecting the rigid bodies to the center, and the amount, size, spacing and mass of the rigid bodies themselves. Quantifying these additional parameters is important in order to make the behavior of the simulated soft body resemble that of a real life body.

To minimize the reality gap, it is also important to improve the realism of the interaction between the end-effector probe and the soft body. The interaction and contact forces between a probe sweeping across the surface of a soft body is highly dependent on the friction forces between the two objects. In real life, the probe is often coated with a lubricant to reduce the friction. Thus, the friction model and parameters used in the simulation environment should be chosen such that this property is preserved.

In [32], the implementation of breathing motion is also presented. If the simulation environment is to be used to train models for clinical applications, it is essential that the soft body deformation resembles that of a real person's torso. Breathing motion is a primary part of how the human body behaves, and is, as mentioned, also one of the factors making it difficult for robot manipulators to interact with human bodies. A breathing motion for the soft body could therefore be implemented to increase the realism of the simulation environment. In practice, the motion can be implemented by applying an outwards force on the rigid bodies making up the soft body. Applying the force periodically will cause the soft body to repeatedly expand and regress, ultimately simulating a breathing motion.

6.10.2 Framework Extensions

After a model shows satisfactory performance in the simulation environment, a natural next step is to test the model performance with a real life manipulator. To test the model with a manipulator, there are several things things that need to be in place. Naturally, the operational space controller in (3.1), together with the model's control law u, must be implemented for the manipulator. Further, the model needs access to the observation space. That is, measurements from the manipulator such as the contact force, the end-effector's pose error and end-effector velocity must

be made available to the model. Often, these measurements are easily extracted from in-built sensors. Lastly, communication between the model and the robot manipulator must be established. The interfaces will be equal to Fig. 3.1, where the "Environment" and "MuJoCo Engine" block will be substituted for a real life manipulator.

As of now, the position of the soft body is directly extracted from the physics engine. To remove this dependency and automate the position extraction, computer vision can be integrated into the framework by setting up cameras around the scene. Different techniques can be applied to the extracted images in order to derive the position of the body, comparable to what is done in [29]. Images can also provide other types of valuable information, such as body deformation. It could therefore be beneficial to extend the observation space with image information when training a model.

The choice of the desired orientation $\mathbf{q}_{\text{goal}}$ was empirical and primitive. More specifically, the goal orientation was chosen such that the probe would be held at an upright position, pointing straight down on the torso regardless of the surface curvature. To make the desired orientation choice more adaptive, a similar approach to that proposed in [31] could be implemented. Before the execution of the sweep, a depth camera can be used to create a 3-D contour of the soft body's surface. For each point along the trajectory, a desired probe orientation could be determined based on the surface contour. Instead of being a constant value, the desired probe orientation would then vary along the trajectory. This could result in better contact with the soft body surface, and ease the contact force tracking.

The proficiency at simultaneously tracking contact forces and end-effector velocity while interacting with soft objects has the potential to be used in several medical applications, for instance automatic ultrasound scans. Often, ultrasound scans are performed by free-hand, and the quality of the scan can potentially be deteriorated by involuntary motion [45]. The acquisition of ultrasound images is affected by the acoustic contact between the probe and the patient, and the variable velocity of the probe. As a first step of being able to train a model capable of conducting automatic ultrasound imaging, ultrasound images can be implemented as a part of the reinforcement learning training loop. As the probe slides across the body, real-life ultrasound images could be displayed according to the probe's position relative to the body. Then, the sequence of the obtained images could work as a measurement of how well the ultrasound procedure is being executed, similarly to the feedback in [30].

Chapter 7

Conclusion

The work done in this thesis builds on the simulation framework created as a part of the project thesis. The overall goal of this thesis was to explore how deep reinforcement learning could be used as robot control for soft body interaction tasks. More specifically, the goal was to make a robot manipulator slide an ultrasound probe across the surface of a soft body, while both applying a desired contact force and keeping a desired velocity.

A total of three reinforcement learning models were trained to complete the interaction task. A baseline model, outputting a desired wrench, was used as a reference to quantify the sampling efficiency and performance gain of using an appropriate low-level controller. The remaining two models were used to investigate how a variable impedance control law would perform at the interaction task. Both models had an action space consisting of proportional gains, making it possible to vary the impedance along the execution of the task. One of the models had an additional action space parameter, allowing control of the end-effector movement in the z-direction directly.

The baseline model showed promising results before yielding unstable behavior. The variable impedance models showed equally good performances, meaning the effect of adding the additional action space parameter is negligible. For each simulation timestep, it was possible to achieve a total reward of 10. The baseline model achieved a mean reward of 5.46 per timestep, while the simple variable impedance model obtained a mean reward of 8.24. The model with extended action space achieved a mean reward of 8.25. Generally, the models struggled to track the desired contact force, where noise characterized the applied force.

Overall, the framework looks promising. Together with ways to improve the model performances, several extensions for further development and improvement have been introduced. After minimizing the simulation-to-reality gap, a natural next step is to test trained models with a real-life manipulator. By introducing computer vision and ultrasound image feedback, the trained models could facilitate automatic ultrasound scans.

Bibliography

[1] Sejal P. Dharia and Tommaso Falcone. Robotics in reproductive medicine. *Fertility and Sterility*, 84(1):1 – 11, 2005.

[2] Aude Billard and Danica Kragic. Trends and challenges in robot manipulation. *Science*, 364(6446), 2019.

[3] Wael Farag. Complex trajectory tracking using pid control for autonomous driving. *International Journal of Intelligent Transportation Systems Research*, pages 1–11, 2019.

[4] Vineet Kumar, BC Nakra, and AP Mittal. A review on classical and fuzzy pid controllers. *International Journal of Intelligent Control and Systems*, 16(3):170–181, 2011.

[5] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine. Residual reinforcement learning for robot control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6023–6029, 2019.

[6] W. . Lu and Q. . Meng. Impedance control with adaptation for robotic manipulations. *IEEE Transactions on Robotics and Automation*, 7(3):408–415, 1991.

[7] Martin H. Myrestrand. Compliant robotic manipulation, 2020.

[8] Hassan K Khalil. Lyapunov stability. *Control Systems, Robotics and AutomatioN–Volume XII: Nonlinear, Distributed, and Time Delay Systems-I*, page 115, 2009.

[9] Luigi Villani and J De Schutterm. Handbook of robotics, chapter force control, 2008.

[10] P. Song, Y. Yu, and X. Zhang. Impedance control of robots: An overview. In *2017 2nd International Conference on Cybernetics, Robotics and Control (CRC)*, pages 51–55, 2017.

[11] Neville Hogan. Impedance Control: An Approach to Manipulation: Part I—Theory. *Journal of Dynamic Systems, Measurement, and Control*, 107(1):1–7, 03 1985.

[12] Fares J. Abu-Dakka and Matteo Saveriano. Variable impedance control and learning—a review. *Frontiers in Robotics and AI*, 7:177, 2020.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[14] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer, 1999.

[15] Ivo Grondman, Lucian Busoniu, Gabriel A. D. Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

[16] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[18] Kullback-leibler divergence. `http://hanj.cs.illinois.edu/cs412/bk3/KL-divergence.pdf`. Accessed: 05/05-2021.

[19] Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE, 2019.

[20] Oren Spector and Miriam Zacksenhouse. Deep reinforcement learning for contact-rich skills using compliant movement primitives. *arXiv preprint arXiv:2008.13223*, 2020.

[21] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[22] Xin Zhao, Huan Zhao, Pengfei Chen, and Han Ding. Model accelerated reinforcement learning for high precision robotic assembly. *International Journal of Intelligent Robotics and Applications*, 4:202–216, 2020.

[23] Mrinal Kalakrishnan, Ludovic Righetti, Peter Pastor, and Stefan Schaal. Learning force control policies for compliant manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4639–4644, 2011.

[24] Jianlan Luo, Eugen Solowjow, Chengtao Wen, Juan Aparicio Ojea, Alice M. Agogino, Aviv Tamar, and Pieter Abbeel. Reinforcement learning on variable impedance controller for high-precision robotic assembly. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3080–3087, 2019.

[25] Roberto Martín-Martín, Michelle A. Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks, 2019.

[26] João Silvério, Leonel Rozo, Sylvain Calinon, and Darwin G. Caldwell. Learning bimanual end-effector poses from demonstrations using task-parameterized dynamical systems. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 464–470, 2015.

[27] Miroslav Bogdanovic, Majid Khadiv, and Ludovic Righetti. Learning variable impedance control for contact sensitive tasks. *IEEE Robotics and Automation Letters*, 5(4):6129–6136, 2020.

[28] Brijen Thananjeyan, Animesh Garg, Sanjay Krishnan, Carolyn Chen, Lauren Miller, and Ken Goldberg. Multilateral surgical pattern cutting in 2d orthotropic gauze with deep reinforcement learning policies for tensioning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2371–2378. IEEE, 2017.

[29] Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pages 734–743. PMLR, 2018.

[30] Mojtaba Akbari, Jay Carriere, Tyler Meyer, Ron Sloboda, Siraj Husain, Nawaid Usmani, and Mahdi Tavakoli. Robotic ultrasound scanning with real-time image-based force adjustment: Quick response for enabling physical distancing during the covid-19 pandemic. *Frontiers in Robotics and AI*, 8:62, 2021.

[31] Qinghua Huang, Jiulong Lan, and Xuelong Li. Robotic arm based automatic ultrasound scanning for three-dimensional imaging. *IEEE Transactions on Industrial Informatics*, 15(2):1173–1182, 2018.

[32] Herman K. Jakobsen. Towards a realistic simulation framework for robot-assisted medical procedures, 2020.

[33] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[34] Panda - franka emika. `https://www.franka.de/technology`. Accessed: 11/06-2021.

[35] The ur5e - a flexible collaborative robot arm. `https://www.universal-robots.com/products/ur5-robot/`. Accessed: 11/06-2021.

[36] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

[37] Akhil S Anand, Guoping Zhao, Hubert Roth, and Andre Seyfarth. A deep reinforcement learning based approach towards generating human walking behavior with a neuromuscular model. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*, pages 537–543, 2019.

[38] Aleš Ude, Bojan Nemec, Tadej Petrić, and Jun Morimoto. Orientation in cartesian space dynamic movement primitives. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2997–3004, 2014.

[39] Single exponential smoothing. `https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm`. Accessed: 06/05-2021.

[40] Single moving average. `https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc421.htm`. Accessed: 06/05-2021.

[41] B. P. Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[42] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. `https://github.com/DLR-RM/stable-baselines3`, 2019.

[43] Default hyperparameter values for ppo algorithm. `https://github.com/DLR-RM/stable-baselines3/blob/378d197b00938579a6a5e04c739f41fec23fd805/stable_baselines3/ppo/ppo.py#L68`. Accessed: 19/06-2021.

[44] Default hyperparameter values for low-level controller. `https://github.com/ARISE-Initiative/robosuite/blob/2a34f81983877852009b64ec3a7eae7efcffe001/robosuite/controllers/osc.py#L107`. Accessed: 19/06-2021.

[45] Pai-Chi Li, Cheng-Yen Li, and Wen-Chun Yeh. Tissue motion and elevational speckle decorrelation in freehand 3d ultrasound. *Ultrasonic imaging*, 24(1):1–12, 2002.

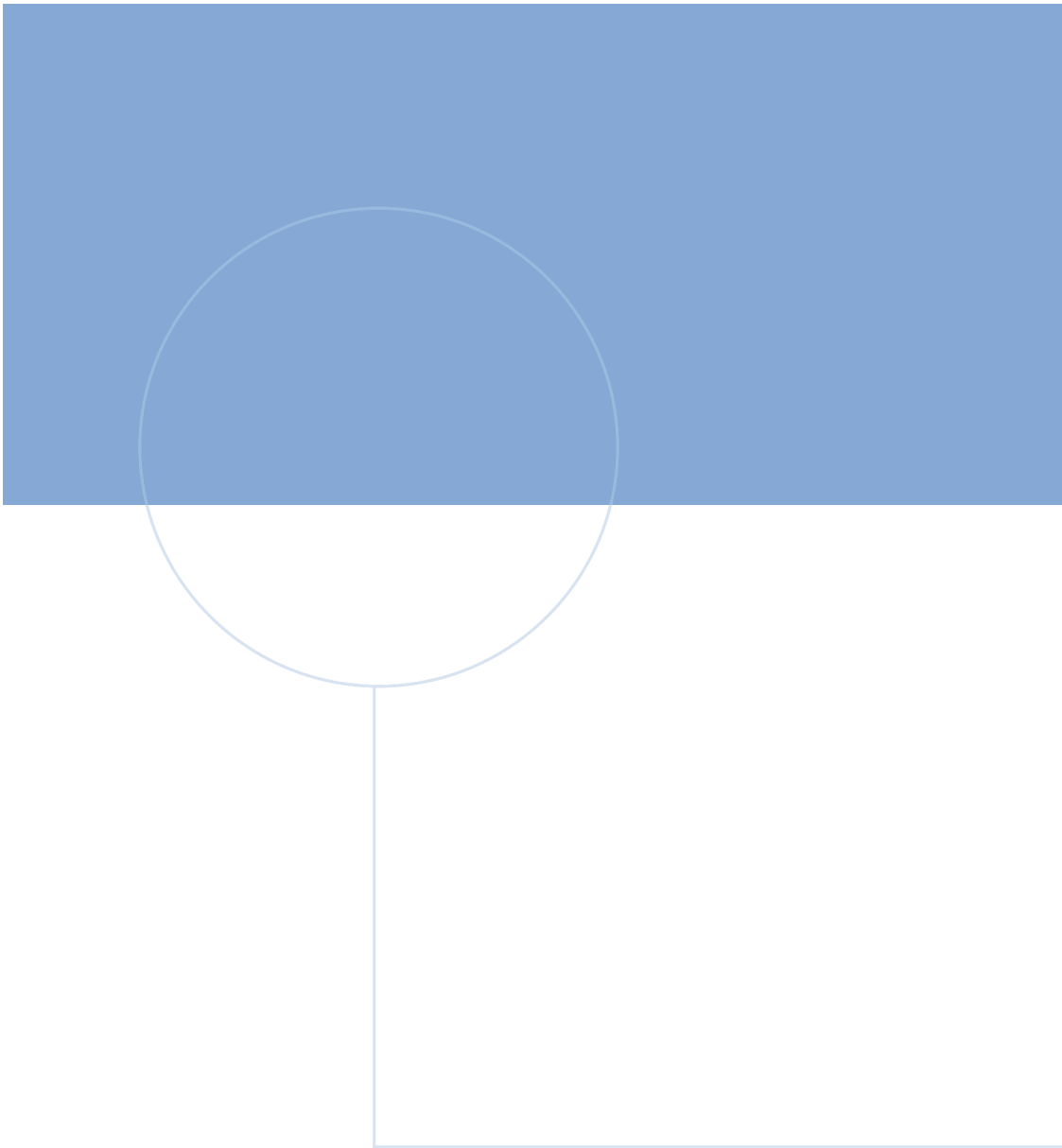
Appendix A

Digital appendix

The link below gives access to the code for the simulation framework. The code is used to produce the results given in Chapter 5.

Simulation framework: *https://github.com/hermanjakobsen/robotic-ultrasound-imaging*

Please do not hesitate to ask if there are any problems or questions. I can be reached on my email address hermankjakobsen@gmail.com.



NTNU

Norwegian University of
Science and Technology