Towards a realistic simulation framework for robot-assisted medical procedures

Herman Kolstad Jakobsen

A report presented as part of the specialization project TTK4550
for the study programme Cybernetics and Robotics.

Herman Kolstad Jakobsen. Towards a realistic simulation framework for robot-assisted medical procedures.
Dept. of Engineering Cybernetics, NTNU, 2020.

Supervisors:
Jan Tommy Gravdahl
Andreas Østvik
Lars Eirik Bø
Akhil S. Anand

Trondheim, Norway

Abstract

The integration of autonomous robot systems in medical procedures are challenging due to the complex scene involved. The handling of moving objects and soft materials, combined with variations between patients, 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 possible source for acquiring the necessary training material and experience for robot manipulators to adapt to a real-life scenario. Employing sophisticated robotic systems could show great benefit in medical procedures, and possibly help reduce the increasing workload in the health sector.

In this project, a prototype simulation framework for robot-assisted medical procedures has been developed. The framework utilizes MuJoCo as its physics engine, and the open-source simulation framework robosuite as the foundation of the development process. Object models of an ultrasound probe and a soft torso has been created and integrated into a simulation environment. In order to minimize the reality gap, a calibration task of the modelled soft-body was designed. The simulation framework was later integrated with baseline reinforcement learning algorithms, and a simulation environment was created to showcase the functionality of the integrated framework. This included designing a reward function and extracting relevant observations from the simulation environment.

The resulting simulation environment consists of a robot manipulator with an ultrasound probe as its tool, a soft body representing the patient and a table. The framework has a modular structure, facilitating seamless interchangeability of its components. The calibration task yielded a ratio specifying the relationship between the stiffness and damping of a real soft body, enabling realistic imitation of its properties in simulation. As a proof-of-concept showing the integration of reinforcement learning in the simulator, an UR5e robot manipulator was trained to push a cube.

The simulation framework looks promising for exploring the use of reinforcement learning in a medical setting. The proof-of-concept exhibits the capability, however, a vast amount of work and further investigations needs to be conducted to fully exploit the potential and transfer the technology into clinical applications.

Preface

This report concludes my specialization project for the study direction Robotic Systems at the Norwegian University of Technology and Science, and will hopefully serve as a good foundation for my upcoming master thesis.

To start, I would like to thank my supervisor Jan Tommy Gravdahl for providing me with helpful suggestions and tips when I presented my work mid-semester. Your contributions have helped guiding the project in the right direction.

I would also like to express my sincere gratitude to my other two supervisors Andreas Østvik and Akhil S. Anand. Your knowledge and experience haven proven to be invaluable, and it would not be possible to complete this project without your help. Your passion for the project is contagious, keeping the motivation and drive at a constant high. Thank you, Andreas, for keeping a cool head through some stressful periods and guiding me through this roller-coaster of a semester. The end-product would have been a shadow of itself without your inputs. Thank you, Akhil, for sharing your never-ending expertise within reinforcement learning and also for setting up the robot used in the experiment. Hopefully, I can be able to repay you by showing you some tips and tricks at the gym. I would also like to thank Lars Eirik Bø for providing CAD models when needed and always being ready to help. Your jokes and infectious good mood during our meetings are also highly valued.

Lastly, I would like to send a warm thank you to "Gutta på Fjellet". This semester has been different and tough due to the ongoing pandemic. I could not wish for a better group of guys to share apartment with. Your support and encouraging pep-talks have been crucial for the completion of this semester, and doubtfully you understand how important your company have been for my well-being. Therefore, thank you, Aksel, Iver, Vebjørn and Sjur.

Trondheim, December 17, 2020

Herman K. Jakobsen

Herman Kolstad Jakobsen

Contents

Chapter 1

Introduction

Robots and automated processes are becoming increasingly integrated into society, where they are capable of performing cumbersome and repetitive tasks historically done by humans. In medicine, robot-assistants were introduced already in the mid-eighties [1]. Later, they have evolved into becoming an established part of clinical procedures. Robots have a huge potential within the healthcare sector, where they are capable of executing precision demanding tasks with high repeatability. Compared to humans, robots have a higher endurance making them both capable of meeting the ever-growing demand for treatments and medical procedures, and prove themselves economically beneficial. Generally, the cost of industry-standard robot manipulators is decreasing and thus making robot manipulators more accessible. Developed robotic systems could potentially prove to be a viable choice compared to current solutions in terms of expenditure.

However, automation of tasks using robot manipulators, like ultrasound imaging, has not been widely adopted within the healthcare sector. One of the reasons is that current robot systems are still not proficient enough at handling moving objects and soft materials. This makes it challenging to interact with body parts and organs that are moving and changing shape, both because the patient does not necessarily lie completely still, but also because of breathing motion and pulsation. The body will in addition be manipulated as part of the procedure.

Machine learning has proven itself useful in creating robot controllers that generalize well and are able to handle new and unseen events [2]. By learning from experience, machine learning algorithms are unlocking a whole new set of possibilities. These algorithms, however, often require large amounts of data to obtain sufficient results, and how to efficiently acquire such amounts of data is still an unsolved problem. Using simulators to replicate the real world has proven to be a feasible solution, where the robot can learn from simulated experiences.

The world population is both increasing and ageing. By 2050, the number of older persons are expected to double [3]. Instinctively, this will create a higher demand for medical procedures and thus greater workload in the healthcare sector.

Introducing sophisticated and autonomous robot-assisted medical procedures could therefore have a significant impact on the healthcare sector in the following years.

1.1 Goal of the project

The goal of this project is to develop a realistic simulation framework for robot-assisted medical procedures, with especially focus on ultrasound imaging. The framework will include a robot arm holding an ultrasound probe in various realistic settings. One of the development criteria is that the framework will facilitate development, training and evaluation of Reinforcement Learning (RL) algorithms, making it possible to control the robot arm while dynamically compensating for motion and changing shapes of objects. The project will contribute more knowledge and methods in the field of robot-assisted medical procedures, and hopefully facilitate further research and development on the use of reinforcement learning combined with robots in the medical setting.

1.2 Outline

This project report is organized into six chapters. The second chapter contains a literature survey presenting the field of use and performance for a selection of physics engines. The chapter further presents modeling concepts for the chosen physics engine, an overview of a utilized simulation framework and a short introduction to reinforcement learning. The chapter serves as the general background needed to solve the project's objective. The third chapter describes the implementation of the simulation framework, together with the design process and execution of a calibration task. Features of the resulting framework, together with the results from the calibration task is shown in chapter four. The fifth chapter discusses the choice of physics engine, the resulting framework and the data from the calibration task. Suggestions for further work is also included. The report is concluded with closing remarks in the last chapter.

Chapter 2

Background

2.1 Literature survey

There exists a wide collection of physics engines; ranging from game-focused engines excelling at rendering large groups of rigid bodies to robotics-focused engines specializing in simulating accurate contact and interaction dynamics between robots and objects. With a growing selection of both physics engines and simulation environments it is important to get an overview of the optimal engine for the problem at hand. The choice of engine must be based on specified performance on the simulation side, and minimum requirements on the problem side.

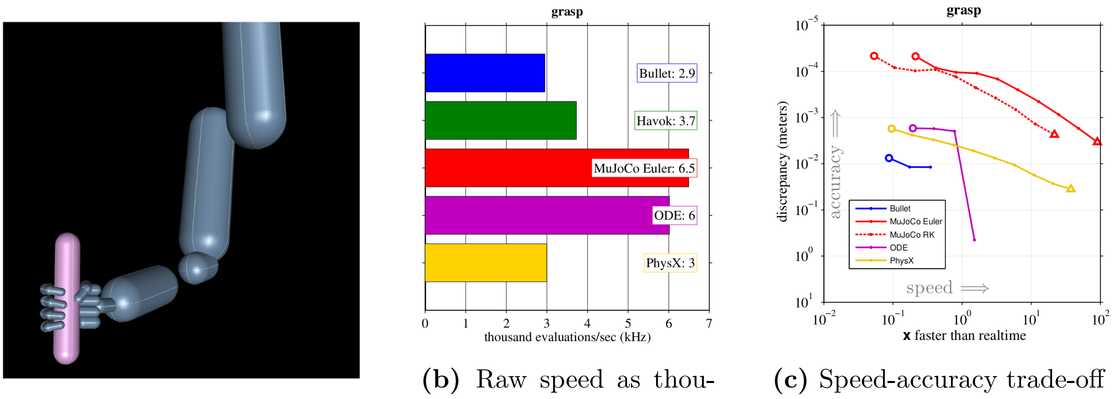
A literature survey was conducted in order to get an overview of some of the most popular physics engines, their field of use and how they perform. Some of the engines included in the survey were MuJoCo, Bullet, PhysX and ODE.

2.1.1 Speed, consistency and energy and momentum conservation

The metrics self-consistency and speed-accuracy trade-off of the numerical integrator used in the physics engine are two quantitative measures used to evaluate simulation performance. These focus on the numerical challenges typical for robotics, as opposed to multi-body dynamics and gaming [3]. In the study by Erez *et al.* [3], they use an engine-specific reference trajectory obtained by simulating the system with a very small timestep h. The timestep is then increased and the deviation from the reference measured, iteratively. Note that this measurement does not capture model errors, where non-physical modeling, like ignoring Coriolis forces, will go unpunished. The different engines' energy and momentum drift were also measured in separate tests. In total, the paper tested the self-consistency of the engines in four different model systems. Two of these model systems were further used to quantify the engines' energy and momentum conservation.

The grasping model system, shown in Fig. 2.1a, is the most relevant modelling system presented in [4]. It consists of a 35 degree of freedom (DOF) robotic arm grasping a capsule using fixed spring-dampers. The system has large mass ratios and involves many simultaneous contacts, making dynamic simulation difficult. Hence, this system has many similarities with the final environment of this project, and the obtained results could therefore provide a good indication on how the physics engines would perform in the finalized environment.

The results from the grasping model system is shown in Fig. 2.1b and Fig. 2.1c. Here, MuJoCo is the fastest physics engine while also consistently providing the best accuracy by a significant margin. The reason for some of the engines having partial graphs in Fig. 2.1c is instability. For instance, Bullet and ODE yielded unstable simulations for larger timesteps and it was therefore no longer meaningful to measure the consistency.



(a) A 3D illustration of the grasping model system.

(b) Raw speed as thousands of evaluations per second for each engine.

(c) Speed-accuracy trade-off in terms of the consistency measure.

Figure 2.1: Results from the grasping model system.

Source: [4]

Results from the other model systems are in accordance with the results in Fig. 2.1. The MuJoCo engine achieves the best overall results. When it comes to consistency, MuJoCo outperforms the other engines by orders of magnitude. The least consistent engines were Havok and PhysX. On systems relevant to robotics, MuJoCo is the fastest engine. However, ODE is the fastest engine when simulating gaming relevant systems, while MuJoCo is the slowest. Further, MuJoCo with the 4^{th}-order Runge-Kutta integrator [5] outperforms the other engines in energy conserving systems. Engines using Cartesian coordinates, like Bullet, performs best when preserving linear momentum.

2.1.2 Reality gap

The necessity to abstract, approximate or remove certain physical phenomena will lead to discrepancies between the simulations and the real world. As an effect, control systems created in simulation will often not perform to the same standard in reality. The difficulty of transferring simulated experiences into the real world is often referred to as a ”reality gap”. Gaps essentially relates to actuators, sensors and physics that determine interactions between robots and objects in the environment. Examples are gear backlash, sensor noise and interactions with soft objects. Reality gaps are of increasing importance [6] as deep learning algorithms need large amount of relevant data to achieve acceptable performance. In order to efficiently use data from simulations, the resemblance to the real life task must be high.

Quantification of the reality gaps found in robotic grasping is done in [7]. A total of three robotic manipulation tasks performed by a 6-DOF Kinova Mico2 arm was recorded by a highly accurate motion capture setup and used as a ground truth. The tasks include single-joint control, multi-joint control and interaction with an object (i.e. a cube). The same scenarios were simulated across a range of popular simulators, and the simulated data was then compared to the real movements given by the ground truth. Note that the three simulators used to conduct the simulations expose the following five physics engines: Bullet, ODE, Vortex, Newton and MuJoCo. It is further worth noting that three different implementations of the Bullet engine was used in the simulations: PyBullet, Bullet278 and Bullet283.

The accumulated euclidean errors of the different simulator and engine combinations are shown in Table. 2.1. Overall, PyBullet, which exposes the Bullet engine, achieves the lowest total accumulated euclidean error followed closely by V-Rep running the Newton and Vortex engine, respectively. In other words, PyBullet, Newton and Vortex were considerably and consistently better at controlling the manipulator compared to the other engines. MuJoCo, on the other hand, accumulates a large error for the multi-joint control task, effectively increasing the engine's total accumulated error. ODE yielded an unstable simulation for all the scenarios, while V-Rep with the Bullet engine gave consistently large accumulated euclidean errors.

Table 2.1: Accumulated (over timesteps) euclidean error (m) compared to ground truth. Source: [7]

	1 Joint	**2 Joints**	**Cube**	**Total**
MuJoCo	24.237	49.430	23.471	97.138
PyBullet	**18.249**	7.000	**20.084**	45.513
V-Rep (Bullet278)	27.412	81.166	25.034	133.611
V-Rep (Bullet283)	26.698	80.004	25.215	131.916
V-Rep (Newton)	18.810	**5.579**	21.069	45.458
V-Rep (Vortex)	18.887	5.664	21.130	45.680
V-Rep (ODE)	1.31e+17	1.19e+18	1.88e+18	3.20e+18

However, the results do not tell the whole story, and there are several characteristics from the simulations that are worth noting. For instance, as shown in Fig. 2.2 only MuJoCo and Bullet283 were able to interact with the cube. PyBullet, which achieved the best results in Table. 2.1, was not capable of interacting with the cube at all where the robot's gripper moved over the cube. Further desirable simulation characteristics can also be found in the simulations of the single- and multi-joint control tasks. An example is that MuJoCo and Bullet283 are capable of replicating the oscillations generated by the real controller. MuJoCo is the only engine capable of reaching the goal positions before changing trajectory in the multi-joint control task.

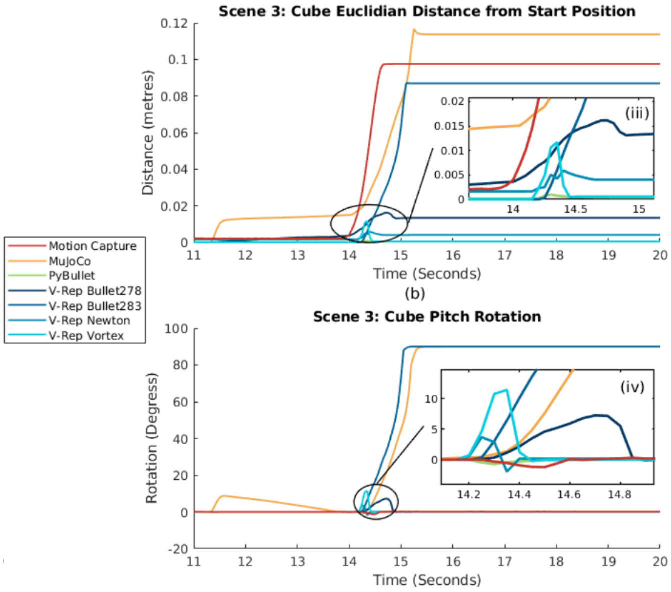


Figure 2.2: Two plots of scene 3 - three joints move in a sequence to push a cube along a flat plane. In the upper plot, the x-axis represents the start position of the cube with lines the cubes euclidean distance away. The lower plot shows the pitch of the cube (where pitch is the rotation around the y-axis).

Source: [7]

2.1.3 Other factors

Overall, the engines offer documentation to some extent, either via web pages or manuals. The documentation generally describes the use of functions together with modelling examples and use cases. It is worth noting that MuJoCo also offers explanation of the engine's mathematical foundation and an in-depth modelling tutorial. Well-written documentation lowers the barriers to entry, making it easier to start creating quality models with desired properties.

MuJoCo also offers integrated soft body dynamics and the possibility to model soft objects, both who are crucial components for the final environment. Bullet, on the contrary, only contains soft body dynamics to a certain degree as expressed in [8]. Better soft body dynamics are not imminent as the inventor states that the focus is on improving the rigid/multibody/collision detection parts of the engine.

Similar to Bullet, ODE also offers soft body dynamics to some degree where soft materials can be simulated by specifying soft constraints for the joints of a rigid body.

2.1.4 Summary

Based on the results from [4] and [7], and other factors presented in Section 2.1, MuJoCo was chosen as the preferred physics engine. The engine is fast and shows great consistency in a range of tasks, especially in the grasping task which resembles the dynamics needed in the final environment. The engine can, however, be regarded as suboptimal for closing the reality gap. In the results from [7], MuJoCo accumulates a large euclidean error compared to the ground truth. Having said that, MuJoCo resembles reality well when simulating interaction tasks between a robot with gripper and an object, a scenario that is highly relevant for the final environment.

Additional reflections on the choice of physics engine can be found in Section 5.1.1. In the following section, background theory for the MuJoCo physics engine will be presented.

2.2 MuJoCo

MuJoCo (Multi-Joint dynamics with Contact) [9] is a physics engine aiming to aid research in areas where fast and accurate simulations are needed. These areas include robotics, biomechanics, graphics and animation. MuJoCo is designed for the purpose of optimization through contacts, making it possible to apply computationally-heavy techniques, such as optimal control, in contact-rich environments at a large scale.

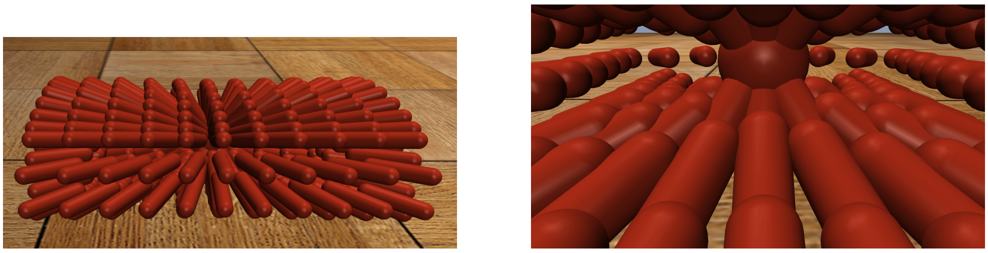
2.2.1 Modeling

In MuJoCo, models are defined using XML files written in either the physics engine's native MJCF format, or the more well-known URDF format[1]. MJCF models can represent complex dynamical systems with a wide range of features and model elements, and the MJCF modelling language can be thought of as hybrid between a modelling format and a programming language. At compile time, the plain text XML file, stated in either the MJCF or URDF format, is compiled into a low-level mjModel which is used for simulations and computations.

MuJoCo offers the possibility of modelling soft bodies. In MuJoCo, soft bodies are modelled as composite objects, who are made up of a collection of rigid bodies. The collection of rigid bodies can be organized in a 1D, 2D or 3D grid depending on the type of soft body it is desired to model. For instance, the bodies could be

---

[1] `https://wiki.ros.org/urdf`



(a) Particle system of rigid bodies. **(b)** Center of particle system.

Figure 2.3: Example of a soft 3D object modelled in MuJoCo. The rigid bodies are organized in a 3D grid along the outer shell, with sliding joints between each rigid body and the center in order to simulate a flexible object.

organized in a 1D grid in order to model a whip, while a piece of cloth would require the bodies to be organized in a 2D grid. Thus, a composite object is essentially a particle system of rigid bodies. However, the particles can be constrained to move together in a way that simulates a flexible object.

Soft 3D objects are of special interest as they can be used to simulate a human torso. An example of a soft 3D object is shown in Fig. 2.3. Here, the particles form a grid along the outer shell, efficiently reducing the number of particles needed to model the 3D object, compared to a dense 3D grid. In order to mimic the softness of a flexible object, each particle has a single sliding joint pointing away from the center of the grid. These joints allow the surface of the soft object to compress and expand at any point. For the soft object to both maintain its shape after deformation and for the deformation itself to be smooth, the joints are both equality-constrained to their initial position and their neighbour joints. Finally, the sum of all joint displacements should be constant in order to preserve the volume of the object.

2.2.2 Constraints

MuJoCo has a constraint model that includes several types of constraints, such as equality constraints, friction loss constraints, limit constraints and contact constraints. Each conceptual constraint can contribute one or more scalar constraints towards the total constraint count n_C, and each scalar constraint has a corresponding row in the constraint Jacobian $\mathbf{J}$.

MuJoCo models equality constraints in the general form

$$\mathbf{r}(\mathbf{q}) = 0 \tag{2.1}$$

where $\mathbf{r}$, connoted as a residual, can be any differentiable scalar or vector function of the position vector $\mathbf{q}$. Each equality constraint contributes $dim(\mathbf{r})$ elements to n_C.

The equality constraint for locking a single sliding joint into a constant position, as used in the composite 3D object, is simply given by

$$y - y_0 = a_0$$

where y is the joint's position and y_0 is the joint's initial position. The left-hand side can be interpreted as a residual, while the right-hand side is a model parameter that is default set to zero. Hence, the equality constraint is stated on the form presented in eq. (2.1).

The general constraint dynamics are set to follow

$$a_i^1 = d_i a_i^* + (1 - d_i) a_i^0, \quad i = 1, 2, ..., n_c \tag{2.2}$$

The variable a_i^1 represents constrained acceleration, while the unconstrained and reference acceleration is denoted a_i^0 and a_i^*, respectively. It should be noted that the accelerations are expressed in constraint space. Model parameters d_i, referred to as impedance, are introduced to directly control the interpolation between the unconstrained and the reference acceleration. The impedance satisfies $0 < d_i < 1$ and the impedance vector $\mathbf{d}$ has dimensionality equal to n_C. Small values of d_i correspond to soft or weak constraints, while large values of d_i correspond to strong or hard constraints.

The reference acceleration is modeled as a spring-damper

$$a_i^* = -b_i (\mathbf{J}\mathbf{v})_i - k_i r_i \tag{2.3}$$

where b_i are the damping coefficients and k_i are the stiffness coefficients. The position residuals are denoted r_i, while $(\mathbf{J}\mathbf{v})_i$ are the joint velocities projected in constraint space. Here, the indexing notation refers to one component of the projected velocity vector. An illustration of a mass-spring-damper system is shown in Fig. 2.4. The relationships between the reference acceleration model and a general mass-spring-damper system can then be summarized as

$$\begin{aligned} (\mathbf{J}\mathbf{v})_i &= \dot{x} \\ b_i &= \frac{c_s}{m} \\ k_i &= \frac{k_s}{m} \end{aligned}$$

where m is the mass, k_s is the stiffness of the spring, c_s is the damping coefficient and $\dot{x}$ is the velocity of the mass.

Focusing on a single scalar constraint, inserting eq. (2.3) into eq. (2.2) yields the following dynamics in constraint space

$$a^1 + d(bv + kr) = (1 - d) a^0$$

The constraint dynamics can therefore be adjusted by choosing values for d, b, and k. As an example, the softness and flexibility of a composite object are directly related to the values of these parameters.

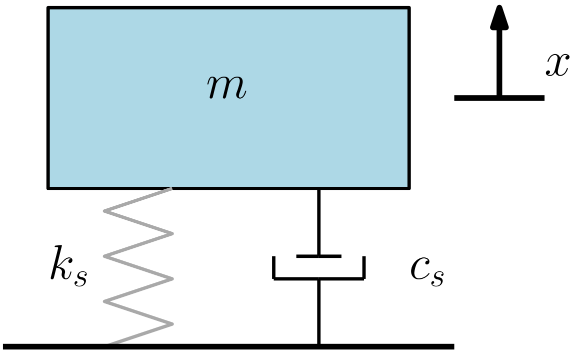


Figure 2.4: An illustration of a mass-spring-damper system, where a mass is connected to a spring and a dampener. The mass is denoted m, while k_s and c_s is the spring and damping coefficients, respectively. The position of the mass is specified as x. Applying an external force on the system will cause a displacement in x.

The impedance is implemented as a function $d(|r|)$ parameterized by five numbers. These numbers specify the shape of the impedance function and corresponds to the following properties

$$dmin,\ dmax,\ width,\ midpoint,\ power$$

Examples of different parameterized impedance functions are shown in Fig. 2.5. The parameterized impedance function depends on the absolute value of the quantity r, which is computed according to the type of constraint at hand. For equality constraints, r equals the constraint violation.

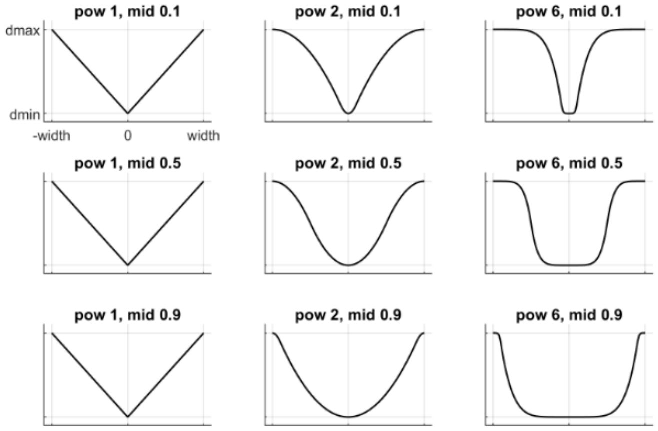


Figure 2.5: Examples of parameterized impedance functions. The impedance $d(|r|)$ is plotted along the vertical axis, as a function of the absolute value of r.

Source: http://www.mujoco.org/book/image/modeling/impedance.png

The damping parameter b and stiffness parameter k are scaled such that the same numbers can be used with different impedances. The scaling formulas are

$$b = \frac{\text{damping}}{\text{dmax}}$$

$$k = \frac{\text{stiffness}}{\text{dmax}^2}$$

Hence, the parameters can be chosen by specifying stiffness and damping assuming the impedance function is already defined.

2.2.3 Contact model

MuJoCo uses a soft contact model. These type of models are often used to model the contact between a soft finger and a rigid object, where the finger is allowed to apply an additional torsional moment with respect to the normal at the contact point.

Mujoco's contact model differs from the *de facto* standard LCP (Linear Complementarity Problem) contact models by dropping the complementarity constraint in its entirety. The complementarity constraint states that force and velocity in the contact normal direction cannot be simultaneously positive, and is an essential part of the LCP formulation. Removing the constraint transforms the LCP model into a convex model. MuJoCo's contact model can therefore be regarded as convex.

The complementarity constraint is removed based on the argument that all physical materials allow some deformation in reality, essentially making all types of physical contact soft. For soft contact, the complementarity constraint has to be violated: when there is penetration and the material is pushing the contacting bodies apart, both the normal force and velocity are positive. The constraint would therefore impose an unrealistic restriction on the simulation when it is assumed that all contacts in the real world are soft.

The contact model is further based on point contacts. A contact point is defined geometrically by a point between the two contacting rigid bodies and a spatial frame centered at that point. Both the point between the contacting bodies and the spatial frame is expressed in global coordinates. The x-axis of the spatial frame points in the contact normal direction, while the two remaining axes span out the tangential plane. The contact distance is defined as positive when the two bodies are separated, zero when they touch, and negative when they penetrate. The contact point is in the middle between the two surfaces along the normal axis.

It is worth noting that MuJoCo assumes that all objects recover their shape before the next contact. In some cases this will lead to an approximated contact force which deviates from reality. Suppose you push your finger into a soft material, pull it back faster than the material can recover its shape, and push again. The contact force experienced on the second push will then also depend on the material deformation created by the first push, and not only on the rigid-body positions of your finger and the object. The model implemented in MuJoCo fails to include this additional dependency.

2.3 robosuite

A Nature study conducted in 2016 [10] indicated that more than 70% of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments. robosuite [11] is a simulation framework powered by the MuJoCo physics engine for robot learning which aims to remove the problem of lack of standardization in papers, along with the need for better benchmarks. This is done by offering a suite of easy to set up and comparable benchmark environments facilitating reproducible research. The overarching goals of robosuite is to provide researchers with a standardized set of benchmarking tasks, flexibility in creating new environments, and a variety of implemented robot controllers and learning algorithms to lower the entry barriers.

2.3.1 Overview

A structural overview of the framework is shown in Fig. 2.6. The framework offers two main categories of Application Programming Interfaces (APIs). The modeling APIs are used for defining simulation environments in a modular and programmatic fashion. The simulation APIs, on the other hand, are used for interfacing with external inputs, such as inputs from a policy or an I/O device. A simulation model, specified by the modeling APIs, is instantiated by the MuJoCo engine. The result is a simulation runtime, which is referred to as an environment. The environment generates observations with the use of sensors such as cameras and proprioception. Action commands are generated by policies or external devices and sent to the environment through the controllers of the robots.

A simulation model is defined by a task object, which encapsulates three crucial components of robot simulation. The components include robot models, object models and an arena. Hence, a task can consist of one or more robots, none to many objects and a single arena. A robot model consists of both a model of the robot and a model of its corresponding gripper, both specified as an XML file. The object model can either be loaded from an XML file, or procedurally generated with programmatic APIs. The workspace of the robots is defined in the arena. The arena includes environment installations and their placements, for instance a tabletop with position and orientation. The task class combines the robot models, object models and the arena into one XML object specified in MuJoCo's MJCF modeling language.

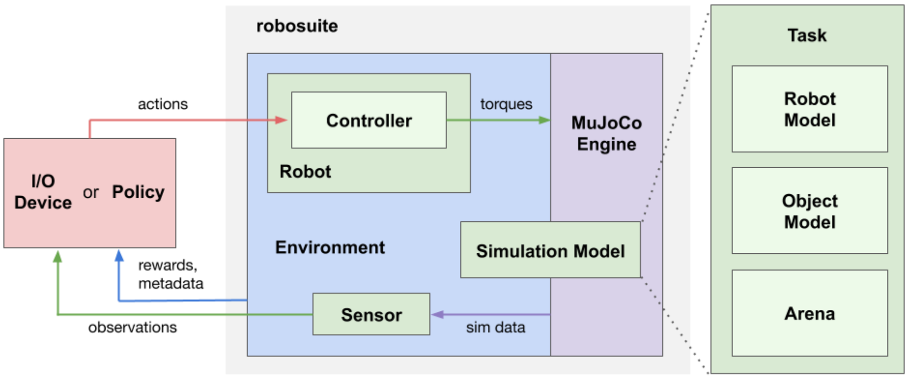


Figure 2.6: Structural overview of the robosuite framework. The diagram illustrates the key components in the framework and their relationships.

Source: https://robosuite.ai/docs/images/module_overview.png

2.4 Reinforcement learning

Reinforcement learning is a type of machine learning [12]. In essence, reinforcement learning aims at getting an agent to learn how to behave in an environment, where the only feedback consists of a scalar reward signal. The distinction between the environment and the agent might not always be intuitive. As a rule of thumb, everything the agent cannot control is a part of the environment.

The goal of the agent is to maximize its reward. How RL algorithms operate can be summarized by Fig. 2.7. The agent chooses to perform an action. This action can either be chosen at random or by the agent's policy. A policy is a function that maps a state, or observations, into an action. The agent then performs its chosen action and the environment returns a reward and observations based on the action. The reward is a measure on how good the performed action was according to some goal. Given the set of action, reward and observations, the agent can update its policy to try to maximize the reward.

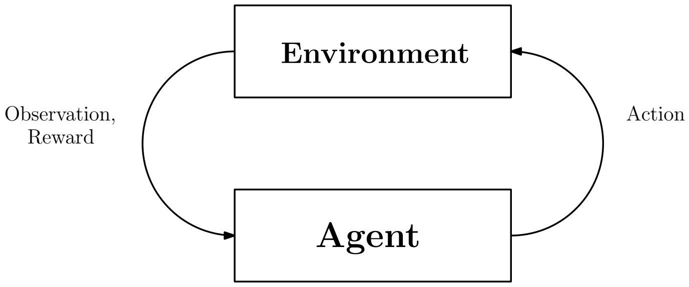


Figure 2.7: Reinforcement learning

An example of a set-up suitable for reinforcement learning is a robot manipulator trying to lift a cube from a table. In this case, the robot manipulator will be the agent, while the cube and the table will be part of the environment. An action would be a set of torques applied to the joints of the robot and its gripper. Joint angles and the position of the cube could be examples of observations, while the reward could be a scalar proportional to how high the cube is lifted.

Chapter 3

Methodology

Based on the project goals, the development process started out with defining the framework requirements. By setting initial requirements for the framework, it was then possible to draft the framework architecture and essential components. Due to its usability and integration of reinforcement learning libraries, the framework was developed using Python. The robosuite framework, with its structure shown in Fig. 2.6, offers a modular architecture which fulfils many of the initial criteria and was therefore used as a foundation for further development.

As already stated, the overall goal of the project is to establish a simulation framework for reinforcement learning within robotic ultrasound imaging. In order to obtain a functional simulation framework, there are several components that need to be implemented and integrated together. To develop a medical ultrasound task by extending the robosuite framework, the further steps include:

- Model relevant objects and environments, such as a robot holding an ultrasound probe gripper, a patient and a room with a table defining the workspace of the robot.
- Extract relevant sensor-data from the environment.
- Integrate the simulation environment with OpenAI gym for performing RL.

With these requirements laying the foundation for further development, the necessary components and framework architecture were specified precisely.

3.1 Modeling

The robosuite framework offers a wide collection of industry-standard robot models, including the popular UR5e [13] and Franka Emika Panda [14] robots. The framework also contains a set of pre-defined robotic environments intended for specific tasks. One of the environments, the *lift-environment* consists of a table, a cube placed on the table and a robot. The environment shows great resemblance to the desired environment, where the cube could represent a patient lying on a table. The lift-environment was therefore used as a basis. Remaining modifications to the lift-environment were then to attach an ultrasound probe as the robot's end-effector and substitute the cube for a soft torso.

Hence, the first task is reduced to modeling an ultrasound probe gripper and a soft torso object. In order to specify the shape of the ultrasound probe, its mesh was designed digitally and saved as a CAD model. By specifying an XML file written in MuJoCo's modelling language MJCF, the mesh was integrated into MuJoCo with additional inertial properties. Utilizing MJCF again, the soft torso was modelled as a 3D composite object. Contrary to the probe, the soft torso object was specified solely in an XML file, as support for specifying the shape of soft objects with the use of meshes is not yet implemented in MuJoCo. The shape of the soft object is then defined by the amount, shape, size and spacing of the rigid bodies making up the object. The modelling process of the soft object then included choosing the amount, shape, size and mass of the rigid bodies, and the stiffness and damping parameters for the composite object's constraints.

How the two models are connected to the robosuite framework is shown in Fig. 3.1. Using robosuite's API, the ultrasound probe gripper is integrated into the robot's XML file as an end-effector. The robot, with the ultrasound probe, is then loaded as a robot model. The soft torso, on the other hand, is a standalone object and its XML file is loaded as an object model. Ultimately, the simulation model consists of a robot with an ultrasound probe end-effector, a soft torso object and an already implemented arena containing a tabletop.

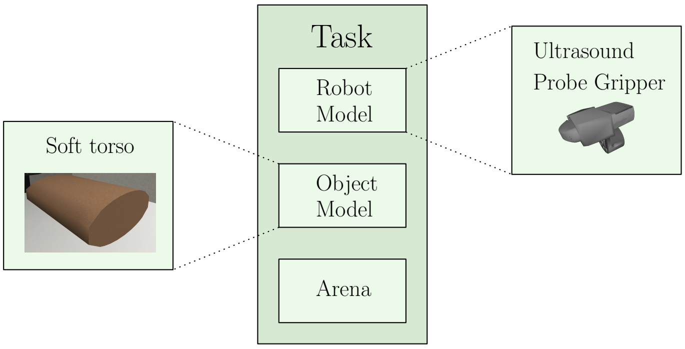


Figure 3.1: Illustration of how the modelled gripper and soft torso relates to the robosuite framework. The gripper is integrated as part of the robot model and loaded with the robot as a whole, while the soft torso is loaded as a standalone object.

3.2 Sensor data

Extracting relevant sensor data from the environment is crucial for implementing RL algorithms, as the algorithms use knowledge about the environment to improve the control policy. robosuite's APIs facilitate uncomplicated extraction of desired and customized data while also simplifying the integration of different sensors. For instance, a Force-Torque sensor was equipped at the tip of the ultrasound probe, making it possible to extract torques and forces acting on the probe. Other implemented data extractions include proprioceptive values, such as joint and gripper positions, and RGB-D camera images. With the use of the framework's APIs, it is possible to extract additional data as they prove useful.

3.3 Integration with baseline RL algorithms

To incorporate the use of RL algorithms with the simulation framework, an RL framework [15] was integrated into the architecture as shown in Fig. 3.2. The RL framework offers a collection of RL algorithms and options to log helpful training information. During training, the selected algorithm collects observations and rewards from the simulation environment. The algorithm uses the values to update the policy while also sending a set of actions to the robot controller from the current policy or a separate policy, based on the type of algorithm used. The format of the actions depends of the type of controller for the robot. Based on the performed actions, the environment will output new observations and rewards. This cycle is performed until an optimal policy is found. Support for multiprocessing has also been included, where multiple environments can be simulated simultaneously to increase the training speed. During testing, the simulation framework is controlled with a trained/learned policy. The learned policy computes a set of actions for the controller based on the observations given by the environment.

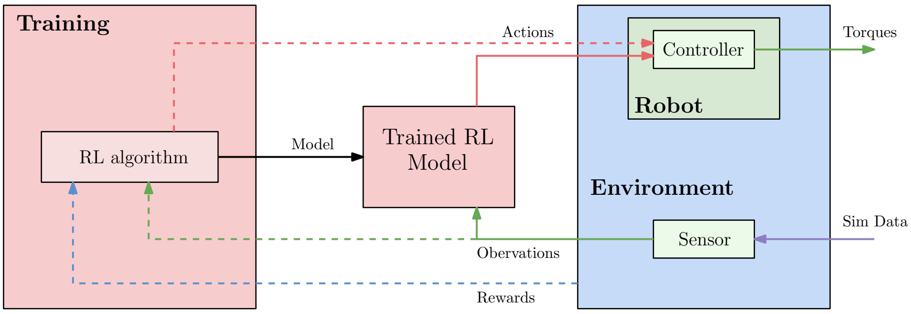


Figure 3.2: Simplified illustration of the relationships in the simulation framework with integrated RL framework.

To test the RL framework and to show a proof-of-concept, the simulation model was simplified. In short, the soft body was substituted for a cube, and a goal position for the cube was added. The robot's task was then redefined to push the cube to the goal position using the probe. The task can be regarded as a simplified version of performing ultrasound imaging, where instead of sliding a probe across a torso, the robot will push a cube to a goal using the probe and hence slide across the table.

3.4 Calibration of body softness

The softness of the modelled torso should resemble the softness of a real torso in order to minimize the reality gap. To accomplish this, a calibration task was designed. By pushing a probe down on a real-life body, the body can be approximated to follow the dynamics of the mass-spring-damper system [16] illustrated in Fig. 2.4 .

Revisiting section 2.2.2 and assuming that the real-life body behaves as a mass-spring-damper system, the constraint dynamics presented in (2.2) can be rewritten into

$$\frac{a^*}{r} = -c_s\frac{\dot{x}}{r} - k_s \tag{3.1}$$

Then, by measuring the acceleration of the body deformation a^*, the body penetration r and the body deformation velocity $\dot{x}$, as the probe pushes down on the body, it is possible to fit a curve to (3.1) using linear regression. The slope of the regression curve will specify

$$c_s = b \cdot m$$

while the intersection of the curve will determine

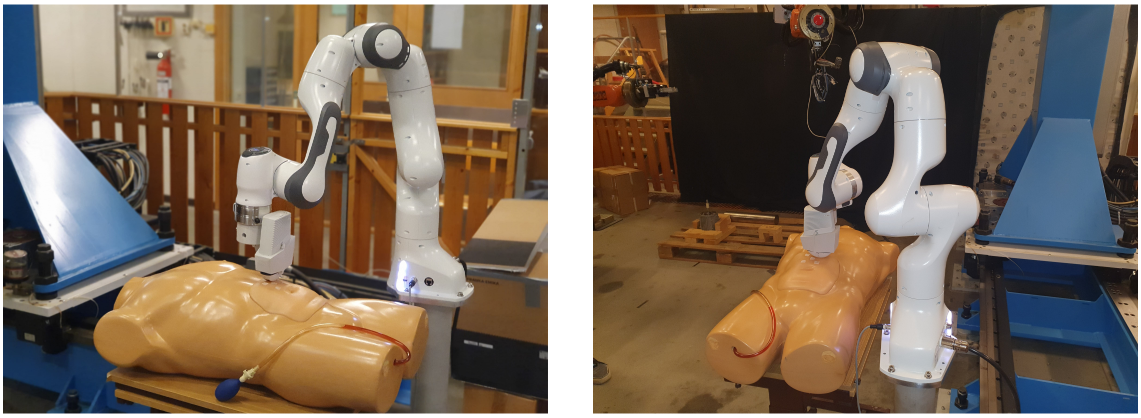
$$k_s = k \cdot m$$

Hence, the values for the slope and the intersection have to be divided by the amount of deformed body mass to obtain the true value of k and b, which then can be used in the MuJoCo model.

The calibration set-up is shown in Fig. 3.3. An ultrasound probe was attached to a Panda robot and a dummy was used to represent a real human torso. Measurements were taken from a total of five sampling locations, each marked on the torso. The probe was manually placed at a sampling location, before being pushed down by applying torques to the robot's joints. In practice, it is difficult to measure the acceleration and velocity of the deforming body. Measurements from the probe were used as an approximation instead. For each sampling location, the force acting on the probe, the position of the probe and the velocity of the probe were measured. The mass of the deformed body was not quantified. To combat the effect of gravity on the force measurements, an offset was subtracted from the force measurements such that the force was equal to zero when the probe was held still. Lastly, the same experiment was simulated in the framework, to allow comparisons of the measurements.

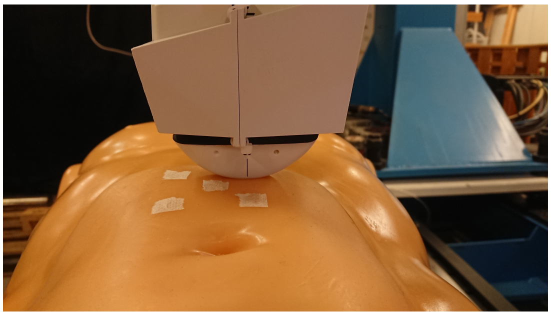
Due to the mass of the deformed body not being quantified, it was not feasible to obtain the true values of the stiffness and damping parameter to be used for the soft body model in MuJoCo. However, the slope and intersection of the linear regression can yield a ratio between the damping and stiffness parameter where the following relationship can be stated,

$$\frac{k_s}{c_s} = \frac{k \cdot m}{b \cdot m} = \frac{k}{b}\,. \tag{3.2}$$



(a) Side view.

(b) Front view.



(c) Manually placed probe at the upper-right sampling location. Joint torques was applied to the robot, making the probe penetrate the body.

Figure 3.3: Set-up for calibration task. A dummy was used to replicate a real-life torso, and an ultrasound probe was attached as the end-effector of a Panda robot. Measurements were taken from a total of five sampling locations.

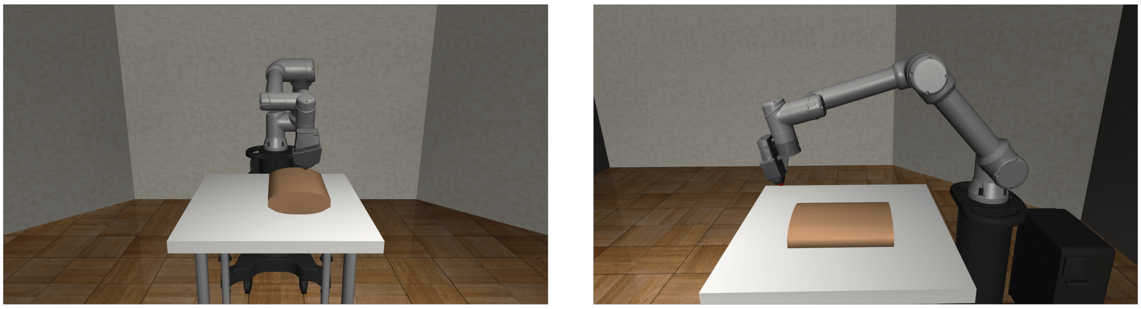
In essence, the calibration task is able to quantify the ratio between the stiffness and damping parameter to use for the soft body model in MuJoCo.

Chapter 4

Results

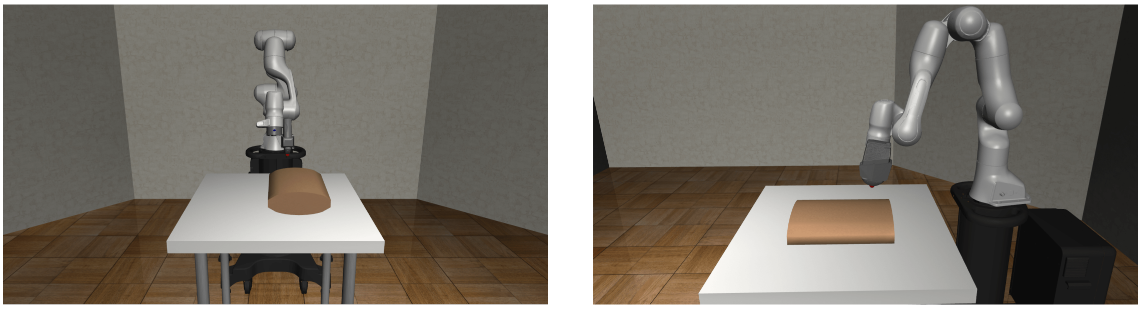
4.1 Simulation environment

The simulation environment for robotic ultrasound imaging is shown in Fig. 4.1. The environment includes a robotic manipulator, an ultrasound probe as the robot's tool, a soft body and a table. The soft body is designed to simulate a human torso and has therefore been coated with a skin. A red sphere on the tip of the ultrasound probe, as for instance seen in Fig. 4.1c, represents a force-torque sensor. The robot manipulators used in the environment are the UR5e and the Panda.



(a) Front view with the UR5e as the robot manipulator.

(b) Side view with the UR5e as the robot manipulator.



(c) Front view with the Panda as the robot manipulator.

(d) Side view with the Panda as the robot manipulator.

Figure 4.1: Simulation environment for robot-assisted medical procedures.

4.1.1 Soft body

To illustrate the softness of the body, the robot manipulator was commanded to apply a contact force on the body by pressing down with the ultrasound probe. A visualisation of the resulting scenario is shown Fig. 4.2, where the compliance of the body is visible at the contact area.

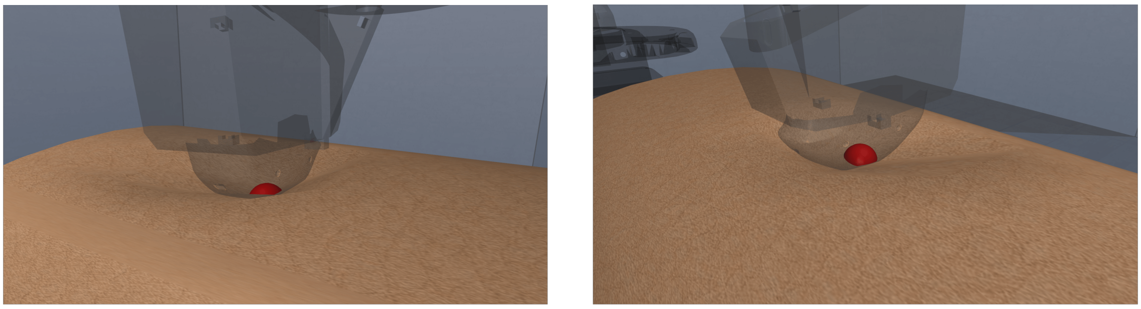
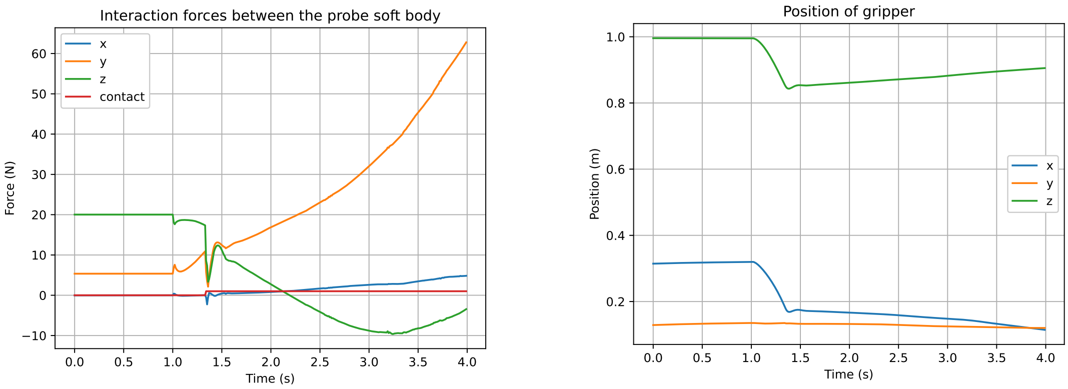


Figure 4.2: Ultrasound probe exerting a contact force on the soft-body. The probe, robot manipulator and tabletop have been made transparent to enhance visibility. The force and torque sensor is visualized as a red sphere.

4.1.2 Sensor data

It is possible to extract a wide variety of sensor data from the simulation environment. An example of extracted sensor data is given in Fig. 4.3.



(a) Interaction forces between the probe and sodt body, expressed in the end effector frame. The contact value is a Boolean variable which equal to one when the probe is in contact with the body and zero otherwise.

(b) Position of the probe gripper given in the global frame.

Figure 4.3: Example of extracted sensor data from the simulation environment. Note that the measurements are given in different coordinate frames.

Here, the robot manipulator was commanded to push the probe down and slide it downwards across the body. The probe comes in contact with the body after

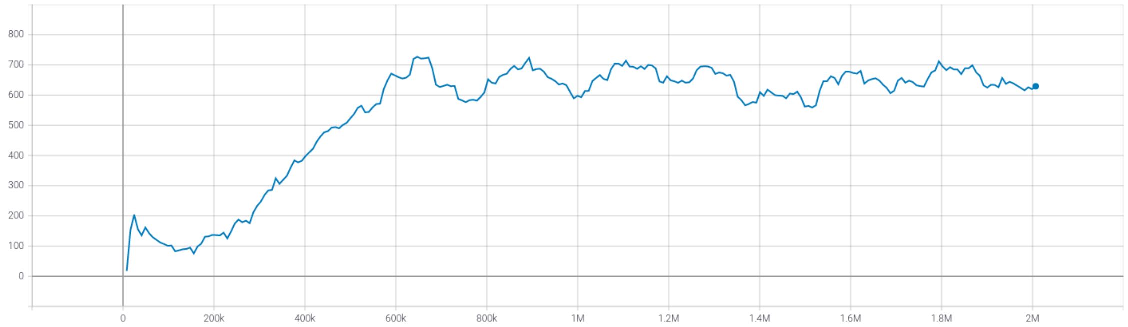


Figure 4.4: Mean reward for the UR5e manipulator learning to push a cube towards a goal. The mean is calculated over a batch of episodes. The x-axis shows the number of timesteps the manipulator was trained for.

approximately 1.5 seconds. It is worth noting that the measurements are given in different coordinate frames. The following list summarizes the available sensor data in the simulation framework:

- The robot manipulator's joint angle positions and velocities.
- The ultrasound probe's position and orientation.
- Force and torque applied to the tip of the ultrasound probe.
- Position and velocity of each rigid body making up the soft torso.
- RGB-D images of the scene.

4.1.3 RL integration

Using the PPO [17] algorithm, the UR5e robot manipulator was trained to push a cube towards a goal. The robot manipulator was trained for a total of 2M timesteps. A graph of the mean reward, calculated over a batch episodes, is shown in Fig. 4.4. The robot manipulator got a positive reward for both approaching and touching the cube with its gripper. Rewards were also given for pushing the cube toward the goal. If the robot manipulator was successful in its task, a large additional reward was given. A video demonstrating the robot in action is available on YouTube[1]. The robot manipulator was unfortunately not able to consistently push the cube towards the goal.

4.2 Calibration task

Plotting values for (3.1) for each sampling location yields the plots shown in Fig. 4.5. The mean value of the resulting slopes and biases, together with the slope and bias

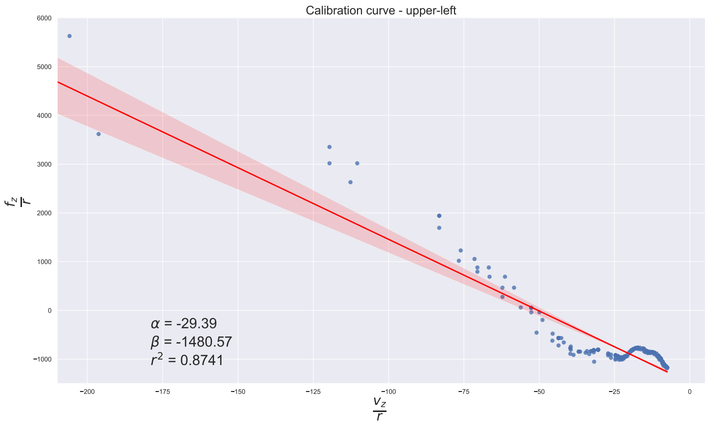
---

[1] `https://youtu.be/FjU5zna1ICk`

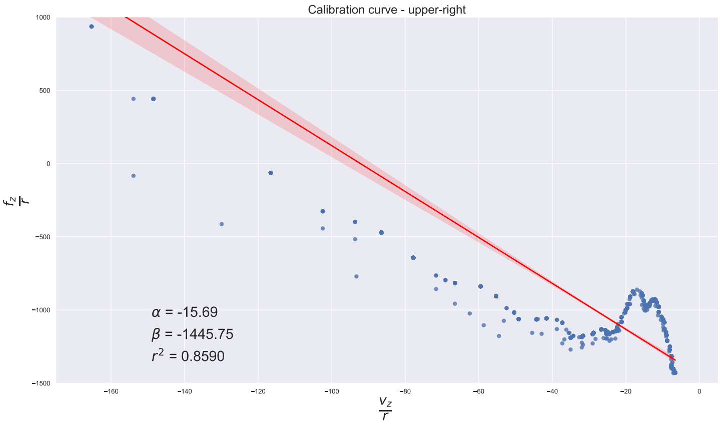
from the simulated experiment, are stated in Table 4.1. Applying the mean values of the slopes and the biases into (3.2), the resulting ratio between the stiffness and damping parameter is,

$$\frac{k}{b} = \frac{1485.64}{28.84} \approx 51.51 \tag{4.1}$$

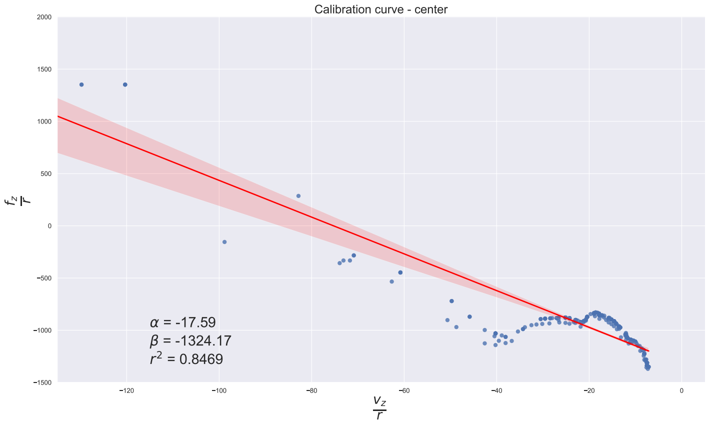
In the simulated experiment, the stiffness and damping parameters of the soft body were set to the findings from the center sampling location. That is, $b = 17.59$ and $k = 1324,17$. The scatter plot for the simulated experiment is shown in Fig. 4.5f.



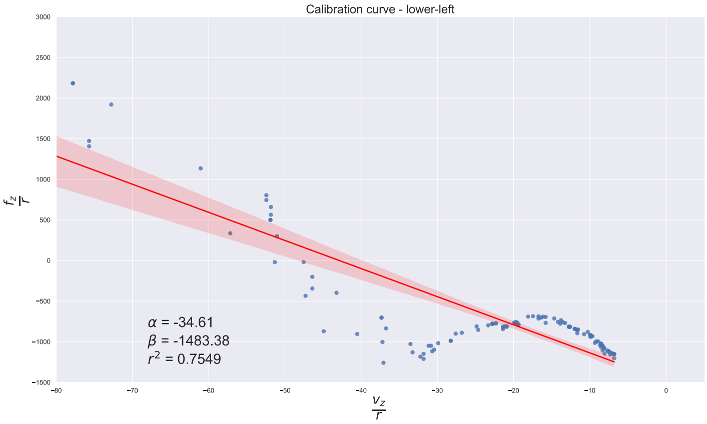
(a) Upper-left sampling location.



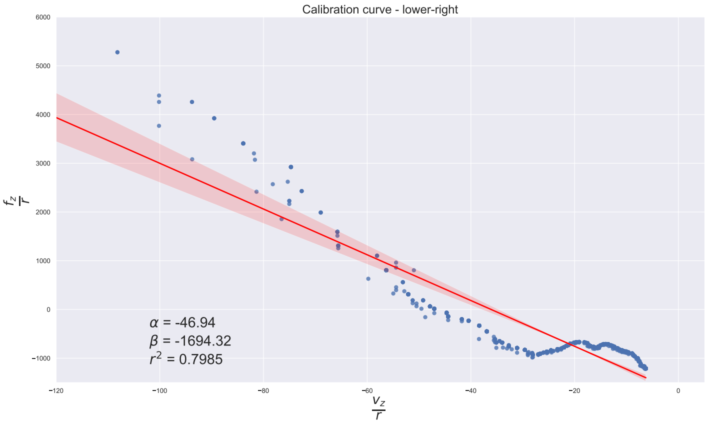
(b) Upper-right sampling location.



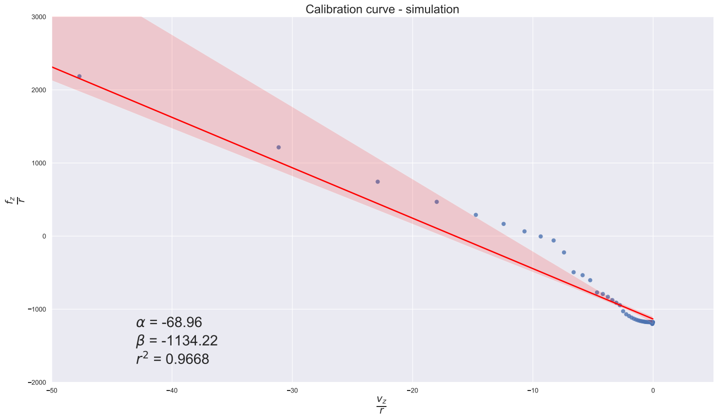
(c) Center sampling location.



(d) Lower-left sampling location.



(e) Lower-right sampling location.



(f) Sampling from simulation.

Figure 4.5: Linear regression of data points obtained from calibration task. The ratio between the velocity of the probe in z-direction and the penetration distance is plotted on the x-axis. The force acting on the probe in the z-direction divided by the penetration distance is plotted on the y-axis. The parameter α is the slope of the linear regression, while β is the bias. The coefficient of determination is denoted r^2. A confidence interval is specified by the translucent bands around the regression line.

Table 4.1: Slope $-c_s$ and bias $-k_s$ from performing linear regression on data points from calibration task. The average value $\bar{x}$ and standard deviation σ are calculated using measurements from the five sampling locations.

Sampling	$-c_s$	$-k_s$
Physical experiment ($\bar{x} \pm \sigma$)	-28.8 ±11.5	-1485.6 ±119.4
Simulation	-68.96	-1134.22

Chapter 5

Discussion and further work

5.1 Discussion

5.1.1 Literature survey

As stated in Section 2.1, MuJoCo was chosen as the preferred physics engine. The engine is fast and shows great consistency in a range of tasks, especially in the grasping task. Bullet is a popular engine and can be regarded as a contender for MuJoCo. However, compared to MuJoCo, Bullet was slow and achieved poor consistency. In addition, the engine yielded an unstable simulation for larger timesteps in the grasping task. Other contenders like ODE also showed poor consistency compared to MuJoCo. Based on consistency and speed, MuJoCo can therefore be considered as the better physics engine choice.

Note that the speed and consistency results from [4] should not be blindly trusted. For instance, the paper was published in 2015 meaning it is over five years old. Many of the compared engines are undergoing active development, ultimately enhancing their performance. Thus engines call for continuous efforts to evaluate them, and the result they achieve today might not be in correspondence with the results achieved in the paper.

Even though MuJoCo achieves great speed and consistency, the engine can be regarded as inferior for closing the reality gap. Compared with other physics engines like Bullet with the PyBullet implementation in [7], simulations in MuJoCo accumulates a large euclidean error compared to the ground truth. Based on this, MuJoCo can be regarded as mediocre if the focus is to control a manipulator while minimizing the reality gap, and PyBullet could be considered as the better choice. Having said that, MuJoCo resembles reality well when simulating interaction tasks between a robot with gripper and an object, a scenario that is applicable for the final environment. Here, all the other engines fall short except for Bullet283. However, Bullet283 achieves a consistently higher euclidean error for all the tasks compared to MuJoCo, and can therefore be regarded as a poorer engine choice.

Lastly, MuJoCo is well-documented making it easier to fully utilize its capabilities. The engine also offers the possibility to model soft bodies, where a crucial component of the final environment is the soft torso model. The fully integrated soft body dynamics available in MuJoCo could therefore prove itself beneficial by making realistic soft body simulations and models achievable. The other physics engines do not offer documentation and integrated soft body dynamics to the same extent as MuJoCo.

5.1.2 Simulation framework

Due to the choice of using robosuite as a foundation for further development, the simulation framework has inherited a modular structure. All the components in the framework are implemented as modules, facilitating seamless interchangeability. For instance, the simulation framework is not bound to use the UR5e as the robot manipulator, as it can be easily be substituted for other manipulators such as the Panda robot. The same applies for the robot controller, grippers, objects in the environment and the workspace of the manipulator. The modular structure makes it also easier to include new models, such as new robots when they are commercial released. In this way, it is possible to keep the simulation framework updated with the latest hardware used in robotics. Essentially, the simulation framework is capable of being configured into simulating a wide variety of environments, which can prove advantageous due to the various settings and environments encountered in medical procedures.

Unfortunately, the policy obtained from training the robot manipulator was not able to control the manipulator into consistently pushing the cube towards the goal. There exists several reasons for this suboptimal result. Looking at the accumulated reward graph in Fig. 4.4, the algorithm finds a local optimum after around 600k timesteps. Hence, to improve the policy, extra care should be taken when designing the reward function. By defining a more sophisticated reward function which makes the algorithm converge closer to the global optimum, it is possible to achieve a policy which makes the robot manipulator push the cube towards the goal. Alternatively, a reward can be given only when the robot arm manages to complete the task. This will give the manipulator exploration freedom while training. However, this type of training will require a substantial increase in the number of training steps and thus computational time. Additional attention to which observations are extracted from the environment and sent to the algorithm can also enhance the training result.

5.1.3 Calibration task

The scatter plots in Fig. 4.5 show a semi-linear relationship between the force applied to the probe and the velocity of the probe. Note that the linear relationship breaks for small velocities as shown clearly in Fig. 4.5d and Fig. 4.5b. This might be due to noise dominating the measurements for small values. Small velocities can also impose non-linearities to the system making the model in (3.1) inadequate.

The simulation was run with the stiffness and damping parameters found from measurements at the centre sampling point. It would therefore be fair to assume that the measurements acquired from the simulated environment will resemble the measurements from the sampling point. Even though the results from the physical and the simulated experiment have the same magnitude, they deviate. It should be noted that MuJoCo, as well as other physics engines, are not able to precisely simulate reality and their simulations are based on assumptions. Naturally, the results obtained in simulation will not be equal to the results obtained from the experiment. As an example, in MuJoCo a soft body is modelled as a collection of mass-spring-dampers, where each rigid body is attached to the center with a spring and a damper. The real-life body, however, is assumed to follow the dynamics of a single mass-spring-damper as stated in Section 3.4. Moreover, in MuJoCo, the stiffness and damping parameters are scaled by the constraint impedance. This value has not been included in the physical experiment and will therefore cause differences between the results.

That being said, the values being in the same order of magnitude shows the potential of the calibration task. The simulated results correlates somewhat to the physical results, meaning that the simulation reflects the reality to a certain degree. By refining the design and execution of the task, it should be possible to close the gap between the dynamics of the simulated soft body and the real soft body.

Attention should also be directed towards the validity of the calibration task. The calibration task can be regarded as somewhat primitive, where it is based on several assumptions and approximations. For instance, the constraint dynamics in (3.1) includes the acceleration of the mass-spring-damper system. It was deemed challenging to measure this quantity during the experiment, and the force applied to the probe was therefore used as an approximation. The acceleration could be derived from the force by dividing with the mass of the deformed body. However, the experiment also showed that it was difficult to measure the mass. As a result, the y-values in Fig. 4.5 are too large and therefore erroneous. Dividing the y-values by the mass will yield lower values, giving smaller slopes and intersections closer to zero for all the curves.

Note that the ratio derived in (4.1) does not have to perfectly resemble the stiffness and damping ratio of a physical torso in order to be used in the simulation framework. Reinforcement learning algorithms are capable of creating policies that generalize well in the physical world, even though they are trained with data from a simulation that does not perfectly model reality. Therefore, it is sufficient that the derived stiffness and damping parameters lie in an interval close to the true values. The parameters used in the soft body model can then be randomly chosen from the interval between simulations, essentially implementing softness variations for the body model accordingly to how the physicality of patients vary.

5.2 Further work

5.2.1 Closing the simulation-to-reality gap

As mentioned, further improving the model of the soft-body is essential in minimizing the reality gap and make the transfer from simulation to reality as effective as possible. The properties and behaviour of the soft body is not only dependent on the stiffness and damping parameters quantified in the calibration task, but also on the constraint impedance and the size, mass and spacing of the rigid bodies making up the soft-body model.

One clear improvement is to modify the calibration task. As of now, the task is dependent on several assumptions that deteriorate the results. Hence, minimizing the use of assumptions would prove advantageous when designing an improved calibration procedure. For instance, the mass of the deformed body should be quantified and thus making it possible to compute the acceleration of the deforming body. Then, it would no longer be needed to assume that the force is equal to the acceleration. Quantifying the mass would also make it possible to obtain the true values of the b and k parameter instead of only the ratio. The calibration task should also quantify the constraint impedance. With a known impedance, it would be possible to fully calculate the stiffness and damping coefficients used in the modelling from the b and k parameter obtained in the experiment.

Breathing motion is a central part of how the human body behaves, and is one of the factors making it difficult for robot manipulators to interact with the human body. To further increase the realism of the soft-body, it could therefore be beneficial to implement a breathing motion. In practice, this could be done by applying an outwards force on the rigid bodies making up the upper layer of the soft body. Applying this force with a given interval will cause the soft body to repeatedly expand and regress, essentially simulating a breathing motion.

To facilitate easier simulation to reality transfer, it is important to implement domain randomization. By initializing the scene with different joint configurations for the robot manipulator together with distinct start positions and softness specifications for the soft body, the policy derived by the reinforcement learning algorithm will have greater generalization capabilities. In this way, the robot controller will perform better in a wider variety of settings and scenes when applied to the real world.

5.2.2 Extensions of the framework

As stated, a simplified simulation environment was created in order to show a proof-of-concept of how the reinforcement learning framework could be developed with minimal effort on sim-to-real transfer. A crucial and natural next step for the development of the simulation framework is to fully integrate the ultrasound imaging environment with the reinforcement learning framework. This mainly includes defining a reward function and decide which observations should be extracted from the environment. The reward function should be defined such that the robot manipulator learns how to perform an ultrasound imaging procedure by gently sliding the probe downwards the body. The rewards should therefore be dependent on the probe pose relative to the body and how hard the probe is pushing down. Extracting the force applied to the probe can be beneficial, as this can be used as measurement on how hard the probe is pushing down. Images of the scene can prove useful for improving model training, and should therefore also be included as an observation. Other observations that obviously should be included are, for instance, the probe and the body position.

Another possible extension is to integrate ultrasound feedback with the framework. As the probe slides across the body, real-life ultrasound images could be displayed according to the position of the probe relative to the body. Such a feature would enhance the total simulation experience as the feedback would resemble how the medical procedure is done in real life. The ultrasound feedback could also be implemented as a part of the reinforcement learning training loop, where the sequence of obtained images could be a measurement of how well the ultrasound procedure is being executed.

Chapter 6

Conclusion

In this project, a simulation framework for robot-assisted medical procedures has been developed. The underlying purpose of the project was to create a simulation framework that could function as a building block for further research and development on the use of reinforcement learning combined with robots in the medical setting.

The developed simulation environment consists of a robot manipulator, an ultrasound probe as the robot's tool, a soft body and a table. The framework utilizes MuJoCo as its physics engine, and the open-source simulation framework robosuite has been used as a basis in the development process. The result is a simulation framework with a modular structure, making it seamless to substitute out components. A calibration task was conducted to quantify two of the parameters specifying the softness of the body model. Lastly, the simulation framework was integrated with a reinforcement learning framework, and a robot manipulator was trained to push a cube as a proof-of-concept.

The framework looks promising and several extensions for further development have been introduced. A natural next step is to fully integrate the ultrasound imaging environment with the reinforcement learning framework. In addition, improving the calibration task, implementing domain randomization and modelling human-like breathing motion are all believed to help minimize the simulation-to-reality gap.

Bibliography

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

[2] 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.

[3] Department of Economic United Nations and Population Division Social Affairs. World population ageing 2017 - highlights. *ST/ESA/SER.A/397*, 2017.

[4] T. Erez, Y. Tassa, and E. Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4397–4404, 2015.

[5] J.C. Butcher. A history of runge-kutta methods. *Applied Numerical Mathematics*, 20(3):247 – 260, 1996.

[6] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, Sergey Levine, and Vincent Vanhoucke. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *CoRR*, abs/1709.07857, 2017.

[7] Jack Collins, David Howard, and Jürgen Leitner. Quantifying the reality gap in robotic manipulation tasks, 2018.

[8] How to do simulation of flexible object in pybullet? `https://github.com/bulletphysics/bullet3/issues/2057`. Accessed: 11/11-2020.

[9] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[10] M. Baker. 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604):452–454, 2016.

[11] 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.

[12] Joohyun Shin, Thomas A. Badgwell, Kuang-Hung Liu, and Jay H. Lee. Reinforcement learning – overview of recent progress and implications for process control. *Computers & Chemical Engineering*, 127:282 – 294, 2019.

[13] The ur5e - a flexible collaborative robot arm. `https://www.universal-robots.com/products/ur5-robot/`. Accessed: 18/11-2020.

[14] Panda - franka emika. `https://www.franka.de/technology`. Accessed: 30/11-2020.

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

[16] Amaury Aubel and Daniel Thalmann. Realistic deformation of human body shapes. In Nadia Magnenat-Thalmann, Daniel Thalmann, and Bruno Arnaldi, editors, *Computer Animation and Simulation 2000*, pages 125–135, Vienna, 2000. Springer Vienna.

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

Appendix A

Digital appendix

The link below gives access to the code for the simulation framework.

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 student email address hermankj@stud.ntnu.no.