Thesis

Interactive Imitation Learning for Force control

Authors:

LANDER Niels (4316878)

Date :
May 2021

Supervisors : Dr. C.E. Celemin Paez, Dr. J Kober
Academic year:
2020-2021

Acknowledgements

Throughout the process of writing this thesis I have received a lot of support from different people. I would like to take this page as a thank you to everyone who has supported me.

First up, I would like to thank my daily supervisor, Dr. Carlos Celemin Paez, for being there to answer all my questions and proofread the drafts I made, to get this thesis in the shape it is now. I would also like to thank him for taking the time to meet with me on a weekly basis, during the most challenging times of the process.

Secondly, I would like to thank my supervising Professor, Dr. Jens Kober, for critically reviewing my methodology, and showing me areas of my research that needed some extra experiments. This has helped me improve the quality of the research quite a lot.

Then I would like to thank my parents, who have also helped me to integrate the structure to a more complete narrative. I would also like to thank them for offering me a place to write my thesis, when I needed it.

Finally I would like to thank my partner, ing. Maaike Luykx, for proofreading and correcting my thesis many times. I would also like to thank her for giving me mental support at the times I needed it most, and challenging me to be a better version of myself.

Summary

To generalize the use of robotics, there are a few hurdles still to take. One of these hurdles is the programming of the robots. Most robots on the market today employ position control, with a set of controller parameters tuned by an expert. This programming is quite expensive, only suited for a single task, in a single configuration, and not interaction safe.

This thesis tries to solve these problems, by introducing *Position And Stiffness Teaching with Interactive Learning (PASTIL)* and *History Aware PASTIL (HA-PASTIL)*, a novel interactive way of learning scalable variable impedance policies. The system is able to learn both positional reference trajectories and stiffness trajectories at the same time.

PASTIL and HA-PASTIL learn these policies from positional corrections applied by a human teacher, through *physical Human Robot Interaction (pHRI)*. For the measurement and extraction of these corrections only the proprioception sensors of the robot are used, so no force/torque sensors are required. To learn from these corrections, the intention of the teacher is estimated, by segmenting the correction space in three parts. Each of these three parts correspond to a set of update rules for the policy, that fit the intention of a correction in that segment.

In this thesis, the proposed algorithms are validated through a series of experiments with sample tasks, and compared with baseline algorithms. The main conclusions from these tests are that PASTIL and HA-PASTIL, as introduced in this thesis, outperform the baseline algorithms on task performance for all tasks and that the learned stiffness makes a positive contribution to task performance. This means that the algorithms proposed here allow for simple systems, with only proprioception sensors, to be instructed by users, instead of experts. This makes it possible for robotics to be applied at lower cost, with less expertise needed to program and operate.

These algorithms, however, still have some aspects that could use further research. The most important example is that they are not yet tested on an actual robot, with physical human robot interaction.

There is still quite some work left to do, but the proposed algorithms might pave the way for more, and better, algorithms that aim to learn force control behaviour form only positional corrections.

List of Acronyms

COACH COrrective Advice Communicated by Humans. 8, 19

DMP Dynamic Movement Primitives. 6, 7, 9, 10, 12, 23, 38

DOF Degrees of Freedom. 4–7, 10–13, 21, 38

HA-PASTIL History Aware PASTIL. iii, vi, 12, 15, 18, 19, 27–38

HCRL Human Centered Reinforcement Learning. 7

IIL Interactive Imitation Learning. 3, 7–9

IL Imitation Learning. 9, 19

LWR Locally Weighted Regression. 6

MDP Markov Decision Process. 6, 7

MOA Mixture of Attractors. 6

MSE Mean Square Error. 22, 29

NN Neural Networks. 6

PASTIL Position And Stiffness Teaching with Interactive Learning. iii, vi, 12, 14, 15, 18, 19, 28, 30, 32, 36, 37

pHRI physical Human Robot Interaction. iii, 8, 9, 39

ProMP Probabilistic Movement Primitives. 6

RBF Radial Basis Function. 6, 10, 14

ROS Robot Operating System. 18, 26

SABL Strategy-Aware Bayesian Learning. 8

TAMER Training an Agent Manually via Evaluative Reinforcement. 8

TP-GMM Task Parameterized Gaussian Mixture Model. 6

Nomenclature

α	Learning rate for stiffnesses
α_z	DMP behaviour constant
β	Learning rate for trajectories
β_z	DMP behaviour constant
$\boldsymbol{\alpha}_i$	Control value for the Impedance control. $[m/s^2]$
$\dot{\boldsymbol{q}}$	Vector of joint radial velocities. $[rad/s]$
$\dot{\boldsymbol{v}}_d$	Desired end-effector acceleration, in Cartesian space. $[m/s^2]$
$\boldsymbol{\eta(q)}$	6-Vector containing Gravity compensation values, dependant on joint angles. $[N]$
$\boldsymbol{\Lambda(q)}$	Inertia matrix, dependent on joint angles
$\boldsymbol{\phi}_k$	Features from the stiffness representation
$\boldsymbol{\phi}_p$	Features from the trajectory representation
$\boldsymbol{\xi}_{corr}$	Trajectory, as executed by the robot, with human correction
$\boldsymbol{\xi}_{nc}$	Trajectory, as executed by the robot, without human correction
$\boldsymbol{\xi}_{ref}$	Reference trajectory.
$\boldsymbol{dK}$	Weight update for the stiffness representation
$\boldsymbol{dP}$	Weight update for the trajectory representation
$\boldsymbol{E}_a$	Extracted correction, used in weight update calculation
$\boldsymbol{E}_r$	Reference error, used in weight update calculation
$\boldsymbol{h}_c$	Wrench of control forces, in end-effector space. $[N]$
$\boldsymbol{h}_e$	Wrench of external forces, in end-effector space. $[N]$
$\boldsymbol{J}$	End-effector Jacobian matrix. This matrix relates joint-space to Cartesian space
$\boldsymbol{K}_d$	Controller damping matrix
$\boldsymbol{K}_M$	Virtual inertia matrix. Set to identity in this work
$\boldsymbol{K}_p$	Matrix of controller stiffness parameters. The learned stiffnesses are put on the main diagonal of this matrix
$\boldsymbol{q}$	Vector of joint angles. $[rad]$
$\boldsymbol{v}_e$	End-effector velocity in Cartesian space. $[m/s]$

$\boldsymbol{w}_k$ Weights vector of the stiffness representation, as used in PASTIL and HA-PASTIL

$\boldsymbol{w}_p$ Weights vector of the DMP forcing function, as used in PASTIL and HA-PASTIL

$\Delta\boldsymbol{v}_{de}$ Vector difference between the desired end-effector velocity, and the actual end-effector velocity. $[m/s]$

$\Delta\boldsymbol{x}_{de}$ Vector difference between the desired end-effector position, and the actual end-effector position. $[m]$

γ Secondary learning rate for trajectories

$\Gamma(q, \dot{q})$ 6-Vector containing centrifugal and Coriolis effect compensation, dependant on joint angles and joint velocities. $[N]$

λ_{seg} Segmentation parameter

τ DMP time parameter

c_s Cost for the stiffnesses

c_t Cost for the time

c_y Cost for the final Y-location

c_{maxz} Cost for the maximum Z deviation

c_{MSE} Cost for the MSE of the measured Z-trajectory with the desired Z-trajectory

f DMP forcing function

q DMP goal

$r(t, \boldsymbol{w})$ Function represented by linear function approximation

t_{max} Normalization parameter for c_t

t_{min} Normalization parameter for c_t

y The spatial degree of freedom that is represented by the DMP, could be any of the Cartesian dimensions

z DMP scaled velocity

Contents

1 Introduction

There are two main unsolved challenges that withhold collaborative robotics from realising its potential [1] and being used in many and diverse applications, from household tasks to industrial fabrication. The first challenge is interaction safety, while the second one is ease of programming.

This is mostly due to the fact that most current robot-arms achieve their accuracy by combining manually programmed high-gain positional control, with a fixed reference trajectory.

In this form of control a slight perturbation in the end-effector position, for example due to unexpected contact, generates large contact forces [2]. This introduces a safety risk in case of human interaction with robot-arms. Programming this kind of control scheme is also very expensive and labour intensive. [3] [4]

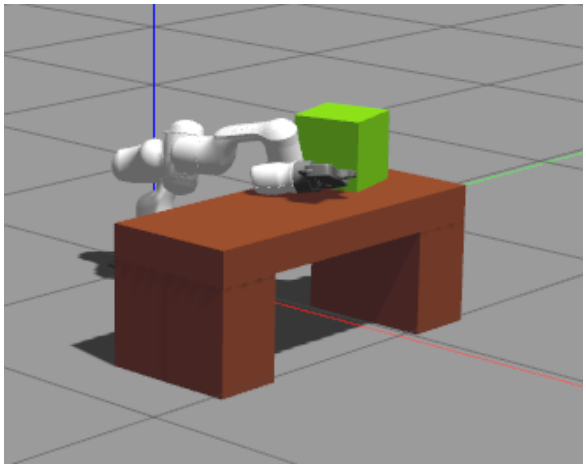


Figure 1.1: Example image of a simulated robot performing a simple task.

In conventional robot programming, the policy is represented as a reference trajectory and a set of fixed controller parameters. Robots programmed this way can only execute a single task, which they are not able to generalize beyond one exact configuration of the environment.

For a future of household robotics, that need to interact and collaborate with humans and perform a wide-variety of tasks in a safe manner and at low costs, this fixed reference high-gain positional control is not suitable and Machine Learning assisted Force Control would be required. This would ideally provide maximum flexibility, easy programming and learning, while having a minimum number of sensors and be operation safe.

Besides the interaction-safety, there are more areas where it would be beneficial to be able to learn a Force Control scheme. Some tasks require such a scheme to be performed well, mostly tasks that contain stochastic perturbations.

To realize this shift of paradigm, efficient Machine learning algorithms have to be developed, along with mathematical representations of the force control tasks. This thesis aims to contribute to solving the first of these two sub-problems. As a preparation for this manuscript a literature study was done to investigate the state of the art on interactive learning for force control. The conclusion of this research was that there are currently no methods that interactively learn both a reference trajectory, and the accompanying controller parameters. This conclusion led to the following research question:

> "*How can a system be designed that interactively learns the stiffness gains of an Impedance controller, along with the reference trajectory, from only position feedback?*"

which this thesis will answer. If such a system is designed, it is possible for non-expert end users to easily, and without the use of external haptic devices, program a robot to execute interaction safe tasks, such as preparing meals, or ironing. This would be a great step towards the automation of many simple household tasks, through a single multi-purpose device.

This work will start of by describing the current state of the art for interactive learning of force control tasks, in Chapter 2, where a brief summary of the literature review is presented. After this summary, a novel developed method for learning force control tasks interactively will be presented in Chapter 3. In Chapter 4 some tests will be discussed to experimentally validate the performance of the newly introduced method. These tests will mostly consist of short example tasks in a simulated world. An example of a simulated robot performing a simple task is provided in Figure 1.1. The results of these tests shall be shown in Chapter 5, and discussed in Chapter 6. Finally, some conclusions will be presented in Chapter 7.

2 | Literature Review

In this chapter the current state-of-the-art in both force control, and the interactive learning of this control, are shortly discussed. To fully grasp this state-of-the-art, it is split up in three pieces; Force Control, Policy Representation, and Interactive Imitation Learning. These three parts have to be combined to form a complete learning system.

2.1 Force Control

The field of Robotic Force Control focuses on the control of robotics, through contact-force feedback. This is complementary to positional control, which solely uses positional feedback. The use of force feedback is important in tasks where the robot interacts, makes contact with, and manipulates objects in the environment. A simple, and often used, manipulation example is the 'peg in a hole' task, where a robot has to insert a peg into a hole. Due to the tight tolerance of the peg-hole combination, and the inaccuracy of the position measurement in robotics, performing this task is non-trivial. Teaching a robot how to solve this task robustly, so that perturbations in hole position and orientation do not lead to task failure, is complicated if only positional feedback is used.

Another problem of pure positional control is that the greediness of reaching the desired position task in this type of controller is not taking into account the possible high contact forces that are generated, leading to possible damage of the system. This problem is also solved by using force control. Force control, overall, makes the execution of interaction tasks safer. One of the most important drawbacks in force control is that is more complex than position-only control, and thus more expensive and more difficult to program [1]. A general overview of the field of force control can be found in both [5] and [2].

2.1.1 Methods

The current state-of-the art in force control can be split up in two different categories: direct force control and indirect force control. Direct force control allows the user to directly specify desired contact forces, where indirect force control focuses more on interaction safety. The most important methods, along with their most defining attributes, are displayed in Table 2.1.

Indirect	Direct
Stiffness Control [6] • No Force/Torque Sensor • Easy to implement	Parallel Force/Motion Control [7] • Controls Force & Position at the same time • Sacrifices some accuracy
Impedance Control [8] • Only needs reference and stiffness • Interaction Safe	Hybrid Force/Motion Control [9] • Exact Control of Position/Force • Decoupled Force/Motion

Table 2.1: Subdivision of the force control methods

In this table, there are four methods. Two of which are direct methods, and two of which are indirect methods. These advantages and disadvantages of these methods shall be shortly highlighted in the following subsections.

Stiffness Control

The advantage of stiffness control is that it allows safe interaction with the environment, without the use of a force-torque sensor, and a simple control law. The downside of stiffness control is that it is a very limited method. It does not allow the specification of contact forces, nor does it allow the specification of the full dynamic behaviour of the system.

Impedance Control

The advantage of Impedance Control over Stiffness Control is that the dynamic behaviour is better defined, as opposed to just defining the static behaviour. In Impedance control the entire behaviour if the mass-spring-damper system can be regulated, rather than just the 'spring-stiffness'. The downside is that a force-torque sensor is required to measure the end-effector forces.

Parallel Force/Motion Control

The advantages of this method over Hybrid Force/Motion Control are that it can control both position and force on all DOFs, instead of having a pre-selected control modality per DOF, this means that contact detection is no longer necessary. The disadvantages are that the position and force control may be less accurate, as result from a constant trade-off between the two.

Hybrid Force/Motion Control

The advantages of Hybrid Force/Motion Control are that the different DOFs are controlled completely independently, and that the control-modalities, be it force or position control, can be selected freely and individually per DOF. This can also be used to switch control modality upon contact detection. The main downside is that a contact detection system has to be implemented to switch between control modalities.

2.1.2 Observations

The state-of-the-art leads to two major observations. The first observation is that indirect force control is more focused on limiting the contact force to a interaction-safe level, which is in line with the goals of this thesis. The second observation is that programming robotic force control is very non-trivial, and limits its use. This is where the potential lies of combining these control schemes with an intuitive interactive learning scheme. The control scheme best suited for this thesis seems to be Impedance control, since it is the indirect method that offers the most control over the dynamic behavior. A more in-depth explanation of Impedance control is given in the following paragraph.

2.1.3 Impedance control

The main advantages of Impedance control, as presented in [8], are that it does not require any form of contact detection, while providing most of the safety benefits of force control. The control law of Impedance control is presented in Equations (2.1) and (2.2).

$$\boldsymbol{h}_c = \boldsymbol{\Lambda}(\boldsymbol{q})\boldsymbol{\alpha}_i + \boldsymbol{\Gamma}(\boldsymbol{q}, \dot{\boldsymbol{q}})\boldsymbol{v}_e + \boldsymbol{\eta}(\boldsymbol{q}) + \boldsymbol{h}_e, \tag{2.1}$$

where $\boldsymbol{\alpha}_i$ is set according to

$$\boldsymbol{\alpha} = \dot{\boldsymbol{v}}_d + \boldsymbol{K}_M^{-1}(\boldsymbol{K}_d\boldsymbol{\Delta v}_{de} + \boldsymbol{K}_p\boldsymbol{\Delta x}_{de} - \boldsymbol{h}_e). \tag{2.2}$$

In these equations $\boldsymbol{q}$ is the 7x1 vector of joint-angles, $\boldsymbol{\Lambda}(\boldsymbol{q})$ is the 6x6 inertia matrix, $\dot{\boldsymbol{v}}_d$ is a 6x1 vector containing the desired values for the linear and angular accelerations of the end effector, $\Delta\boldsymbol{v}_{de}$ is a 6x1 vector containing the differences between the desired and actual values for the six linear and angular velocities of the end-effector, $\Delta\boldsymbol{x}_{de}$ is a 6x1 vector containing the differences between the desired and actual values for the six Cartesian DOFs, $\boldsymbol{\Gamma}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is a 6x6 matrix that includes the centrifugal and Coriolis effects, $\boldsymbol{\eta}(\boldsymbol{q})$ is a 6x1 vector containing the gravitational effects, $\boldsymbol{h}_c$ is a 6x1 vector containing all forces and torques represented in the end-effector task space and $\boldsymbol{h}_e$ are the six external forces and torques acting on the end-effector, as measured by a wrist-mounted F/T sensor, $\boldsymbol{K}_M$, $\boldsymbol{K}_P$, and $\boldsymbol{K}_D$ are 6x6 matrices with controller gains.

In this research $\boldsymbol{K}_M$ and $\boldsymbol{K}_D$ are calculated from the $\boldsymbol{K}_P$, in order to always achieve critical damping. This is done by setting $\boldsymbol{K}_M$ to the identity matrix, and $\boldsymbol{K}_D$ according to

$$\boldsymbol{K}_D = \sqrt{\boldsymbol{J}\boldsymbol{M}(\boldsymbol{q})\boldsymbol{J}^T} \cdot \sqrt{\boldsymbol{K}_P} + \sqrt{\boldsymbol{K}_P} \cdot \sqrt{\boldsymbol{J}\boldsymbol{M}(\boldsymbol{q})\boldsymbol{J}^T}[10], \tag{2.3}$$

where $\boldsymbol{J}$ represents the end-effector Jacobian, and $\boldsymbol{M}(\boldsymbol{q})$ is the joint-space mass matrix. This formula is the more complex version of the scalar critical damping formulation $d = 2\sqrt{km}$, from second order mass-damper systems. With these parameters set, only two parameters remain to be learned to fully define the behaviour of the controller; the reference trajectories, and the stiffness parameters $\boldsymbol{K}_P$.

2.2 Policy representation

To be able to generalize a learned task beyond the training setting, a policy must be learned. A policy is a function that prescribes an action, or a probability distribution over multiple actions, based on the current state of the system. There are multiple ways to represent such a policy. For a robotic setting, a continuous state-action space is needed [11]. This means that both the state, and the actions are continuous, rather than discrete. The robotic state action space is also quite large, so a tabular method cannot be used. This leaves two major approaches for policy representation; Function Parameterization, and Movement Primitives. The state-of-the-art for both shall be discussed shortly in this section.

2.2.1 Function Parameterization

In function parameterization, a policy function is represented as a combination of a set of features and a set of weights. These features live in the state-space, and give of a set of activation values based on the current state. These activation values are then processed, along with the weights, to obtain the outcome of the function for the current states. These weights can be incremented and decremented to change the behaviour of the function, and can thus be used to learn a different policy. Many forms of function parameterization exist, the most well-known being tilings [11], Radial Basis Function (RBF) [12] and Neural Networks (NN) [13].

The main advantage of these kinds of methods is that they can represent policies in large state spaces with a relatively small sets of parameters. The main downside is that these kinds of representations are not specifically designed to represent trajectories. This means that, for example, it does not guarantee convergence to a goal, or posses scaling properties.

2.2.2 Movement primitives

Movement Primitives aim to represent action policies in a parameterized way, while guaranteeing some dynamical and scaling properties, as well as convergence to the goal point. Movement Primitives can be state or time-dependent, and they may work with or without a Markov Decision Process (MDP). Movement primitives can be seen as the 'building blocks' of complex motions. They aim to parameterize a policy that executes a single motion, which either has a goal state, or is a constant periodic movement. These Movement Primitives can be learned individually, and can later be combined, or 'composed', into more complex movement schemes. This makes Movement Primitives suitable for learning all kinds of tasks, from the smallest low level movements, to complete motion plans. Some of the more prominent Movement Primitive approaches are; Dynamic Movement Primitives (DMP) [14], Probabilistic Movement Primitives (ProMP) [15],Task Parameterized Gaussian Mixture Model (TP-GMM) [16], and Mixture of Attractors (MOA) [17]. From these, the DMP framework was chosen for this thesis, since the idea is to make a representation-independent algorithm, and DMPs have the most community support.

Dynamic Movement Primitives

The idea behind the Dynamic Movement Primitives (DMP) framework, from [14], [18] and [19], is to augment a simple dynamic system with a nonlinear forcing function. This simple system is a point attractor for episodic tasks, or a limit cycle for rhythmic tasks. The key advantages of this approach are that the nonlinear function can model many different complex behaviours, while the point attractor guarantees the convergence to the goal point, as the time parameter increases. The nonlinear function is mostly implemented by using function approximation with Radial Basis Function (RBF), although other methods are possible. The parameters of the RBF approximation can be fitted to a single trajectory, using Locally Weighted Regression (LWR).
The basic equations of the DMP framework are

$$\begin{aligned} \tau \dot{z} &= \alpha_z(\beta_z(g - y) - z) + f \\ \tau \dot{y} &= z, \end{aligned} \tag{2.4}$$

where τ is a time parameter, y is the position of the DOF that the trajectory is modelled for, z is a scaled version of the velocity, α_z and β_z are constants related to the damping behaviour of the point attractor, and f is the non-linear 'forcing function' that allows the modulation of the trajectory. The system is 'critically dampened', meaning that there are no oscillations around

the goal point and no unnecessary dampening, if $\beta_z = \frac{\alpha_z}{4}$ [19].
Another important aspect of the DMP framework is that it does not hold an explicit dependence on the time, it instead depends on a 'phase parameter'. This phase parameter can be modified to, for instance, slow or stop execution of the system. This modification can also be done automatically, to make the system continue execution after outside perturbations, making it more robust. This phase parameter can also be used to synchronize multiple DMPs, which can coordinate different DOFs, control modalities, or completely different robotic agents. An example of combined DMPs being applied to a high-DOF system can be found in [20].
The advantages of the DMP framework are that it can represent complex trajectories, in a way that is time and scale invariant, and that the methods to fit and improve the fit of DMPs are well established. DMPs also require relatively little data to make an initial fit. This can already be done using only a single trajectory demonstration. Another advantage of DMPs is that a lot of extensions to the framework exist that make it suitable for specific purposes, such as [21], which incorporates obstacle avoidance in the framework.
Since this representation is not the main focus of the algorithm, the choice was mostly made on the facts that a DMP can be fitted to a single trajectory and that it is supported by a large community. Multiple reliable DMP libraries already exist within the community, for multiple different programming languages.

2.3 Interactive Imitation Learning

The goal of Interactive Imitation Learning (IIL) is to be able to learn from human feedback, with the human as teacher in the learning loop. This feedback may be applied in any way, but is mostly administered as either a bounded numerical value, or a binary flag.

These kinds of feedback signals do not always require an expert in performing the task at hand to be beneficial to the training of the agent. This is the case, since this kind of training is very intuitive for the trainer and the processes are mostly robust against limited wrong feedback [22].

The training of these kinds of agents is often not only interactive, but also 'incremental'. This means that the trainer can safely and clearly determine the performance of the robot and base the feedback on the current performance. This also means that 'wrong' feedback in the past can mostly be compensated by administering 'correct' feedback, making it robust against incorrectly administered feedback.These effects make these kinds of feedback relatively cheap to collect.

With IIL the end-user of a robotic product should be able to teach it new tasks, when desired, making robotic products more widely applicable. A brief overview of the field of Human Centered Reinforcement Learning (HCRL), which is a subdivision of Interactive Imitation Learning, can be found in [23].

One of the key design decisions of IIL is the kind of feedback that the system expects, since different kinds of feedback require different approaches for handling them. The most widely used kinds of feedback are evaluative feedback and corrective feedback. Evaluative feedback is mostly a continuous numerical signal. Evaluative feedback is seen as a direct measure of the desirability of visiting a certain state-action pair. As such it can directly replace the reward function in MDP-based learning, as is demonstrated in TAMER [24]. If the reward function is replaced by evaluative feedback, the agent should maximize direct undiscounted reward, since the long-term consequences are supposedly already incorporated in the feedback.

Corrective feedback, on the other hand, tells the agent which action it should have taken, and thus applying a correction, instead of evaluating the current performance. Corrective feedback is mostly incorporated in the parameter update rule of a parameterized policy, which

pushes the agent into the direction of the optimal policy, rather than labeling each state with the optimal value itself. In this thesis the choice was made to use corrective feedback, since it seems to be the most intuitive to give in the intended pHRI setting. In the intended setting, this kind of feedback shall be applied by the teacher, by grabbing the robot and moving it along a better trajectory.

2.3.1 Methods

In the literature a couple algorithms were found that use IIL. These methods can be split up in three different categories, that are relevant to this thesis. Firstly, general IIL methods, these methods are not specifically designed to work with robotics, but can learn many different tasks in many different settings. Secondly, methods that perform Interactive Imitation Learning through physical interaction. These methods are a bit closer to the goal of this thesis, in the fact that they employ physical interaction to learn from. Finally, methods that utilize Interactive Imitation Learning for Force Control. These methods are specifically created to learn force control policies for robotic interaction. These three categories shall now be discussed shortly.

General Methods

In Table 2.2 a summary of the general methods found in the literature study, with some of their most important features and differences, is presented. The current state of the art in Interactive

	TAMER [24]	**COrrective Advise Communicated by Humans [25]**	**COnvergent Actor-Critic by Humans [26]**	**SABL [27]**	**Policy Shaping [28]**	**DAgger [29]**
Feedback style	Evaluative	Corrective	Evaluative	Evaluative	Corrective	Corrective
Feedback type	Numeric	Numeric	Numeric	Binary	Binary	Actions
Deep Learning Compatible	Yes	Yes	Yes	No	No	Yes

Table 2.2: Summary of the IIL methods.

Imitation Learning (IIL) consists mostly of general algorithms, that can learn a policy based on corrections on this policy, which most often represents a reference trajectory. This policy, however, does not often include different kinds of information, such as both reference positions and stiffnesses. These kinds of algorithms do not appear to be suitable for learning these different kinds of information from corrections on only one of them. To achieve this, a new method has to be developed, that is designed to have this as its core feature.

It could be useful to compare such a novel designed algorithm to one of these algorithms, to prove that learning the controller stiffnesses increases the performance. The algorithm most suited for this seems to be COACH, since it uses the same feedback style. This algorithm would, however, have to be adapted a little bit, to fit with the general setting of this research.

Interactive Imitation Learning through physical interaction

Besides these general frameworks, there are quite a lot of methods that specialize in trajectory or objective function learning through physical Human Robot Interaction (pHRI). Examples of such methods include: [30], [31], [32], [33], [34], [35] and [36]. These methods are able to update, or learn, trajectory policies, or objective functions, based on physically applied corrections. The main difference between these methods, and the goal of this thesis is that these methods do not learn stiffness policies from only positional information from the proprioception sensors. In [30] the stiffnesses are learned partly from tactile information, from a separate tactile sensor, [33] and [36] use an Impedance controller with a set stiffness, while learning only the trajectories, [34]

and [32] do not use force control at all, and [35] and [31] uses information from a force/torque sensor on the robot to infer the stiffnesses. This means that none of these methods directly solve the proposed problem. While some of them solve a part of the problem, none of them even seem to learn stiffness from positional data.

Interactive Imitation Learning for Force Control

In this section the current methods that extend IIL to force control tasks are discussed. While a couple of methods were found that use Impedance control, or learn trajectories from force-based interactions, only one method was found that learns controller parameters for a force control scheme interactively.

In [30] the stiffness of an Impedance controlled robot arm is adapted, based on pHRI. This is done in two different modes, 'wiggling' the robot around its current position decreases the stiffness proportional to the amplitude of the wiggling. Increasing the pressure of the teachers grip on the robot arm increases the stiffness. This increase is measured by pressure sensors inside the robot arm. These choices were made, since these signals seem to be close to the signals that humans use to train other humans in stiffness-dependent tasks.

One of the main advantages of this method is that the robot reacts to the teacher input online, so the teacher can immediately experience the new stiffness. One major downside of this method is that it only adapts stiffness, and does not learn any trajectories, meaning that the teaching is very limited. Another downside is that the 'learned' gains only depend on the current interaction signal, meaning that this method cannot learn incrementally, since it does not use any memories of previous interactions.

2.4 Conclusion

In the current literature there are no methods that interactively learn all information needed for a force control scheme, from only positional corrections. There are methods in IL and IIL that learn either stiffnesses or positions, or both. These methods, however, never learn all of this information from physically applied corrections, measured only by the robots own proprioception sensors.

To create such a system, it would be best to combine Impedance Control, with DMPs and a novel learning algorithm, although the system should not be too dependant on the policy representation. Ideally, the representation could be changed afterwards, to fit the needs of a specific implementation. The choice for Impedance control is made, as this focuses on interaction safety, and can be fully defined by only a trajectory, and a stiffness trajectory. The choice for DMPs is made, since these have the best community support out of all the movement primitive approaches.

3 | Position And Stiffness Learning

This thesis presents a method for interactively learning variable Impedance control from only positional information. The approach presented in this thesis creates a logic system based on intention estimation. The method consists of a few key elements that shall be outlined in this chapter.

The general goal in creating this approach was to create an algorithm that learns variable Impedance control, from interactive corrections that are applied only in the positional domain. This task can be split up in multiple sub-tasks. A successful algorithm for this task needs to be able to represent an internal policy, extract corrections from user input, and use these corrections to update the internal policy. In this chapter the basic approach to these three sub-problems shall be discussed.

3.1 Policy representation

A policy representation for Variable Impedance Control needs to be able to represent both a positional policy, and an accompanying stiffness policy. For the positional policy, there are many solutions in the literature, while for the stiffness policy, the approaches are scarce.

In this thesis the choice was made to represent both the stiffnesses and the positions as six different policies, one for each of the 6 Cartesian DOFs, making it twelve one-dimensional policies total. Creating a complete representation is not the aim of this thesis, as it is aiming to present a set of update rules that should be compatible with most representations.

A good policy representation does not only output a single trajectory, but should also be able to generalize this trajectory to multiple different situations. A popular representation that can handle these problems is DMP, as discussed in the literature study in Section 2.2.2. In this thesis the six positional policies shall be represented as six separate DMPs, which are synchronized through the phase parameter. One of the features of DMPs is a guaranteed convergence to a predefined goal state, as the phase parameter goes to zero. To allow corrections in the later half of the trajectory, and to allow flexibility in the final position of the trajectory, the shared phase parameter of these DMPs is adapted to stay above 0.4, which negates this convergence guarantee.

The six stiffnesses policies shall be represented as a weights vector, which corresponds to the RBF features from the DMPs. A set of 50 basis functions was used for these representations. The advantage of this approach is that only a single set of basis functions is required. The downside is that the scaling properties of this representation are less clearly defined than the properties of DMPs. This is, however, not a point of investigation for this thesis, since it does not focus on the representation.

The update rules, presented in the next part, shall directly update the weights of the DMPs, as well as the weights for the stiffnesses, based on their features. A short study was done to validate this way of updating the trajectories, this study shows that directly increasing the weights of a DMP system gives a similar update to the represented trajectory. A summary of this study can be found in Appendix A. It would also be possible, if this method of updating is undesirable, to update the trajectories themselves, and re-fit the DMPs to the updated trajectory. This solution is recommended when, for instance, there is no direct access to the DMP weights, which could occur when using libraries or other external code.

3.2 Correction extraction

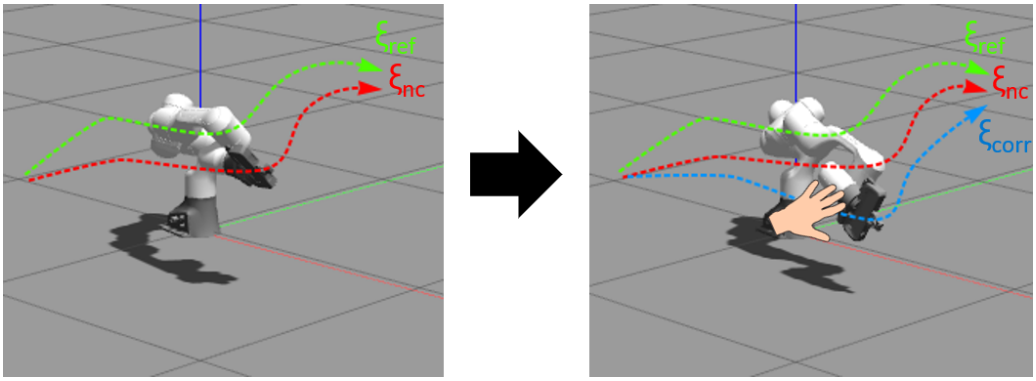


Figure 3.1: Figure that illustrates the difference between the reference trajectory ξ_{ref}, the executed trajectory ξ_{nc}, and the corrected trajectory ξ_{corr}.

The desired system should update the policy based on interactive corrections provided by a human expert. To be able to do this the corrections have to be extracted. This is, however, not as simple as it seems. Due to the inherent compliance that is present in Impedance control there could be a significant difference between the reference trajectory, and the trajectory that is executed, even when no corrections are applied, mostly due to ever-present modelling imperfections. To solve this problem, the trajectory is first executed without corrections, after which the trajectory is executed with human corrections. A visual representation of this way of extracting corrections can be found in Figure 3.1. This problem could also be somewhat resolved by looking at interaction forces, instead of positions, but this thesis focuses on learning from position-only data, to make its applicability as widespread as possible.

In this image ξ_{ref} is the reference trajectory, ξ_{nc} is the first trajectory execution, without corrections, and ξ_{corr} is the trajectory execution to which corrections are applied.

If the corrections are extracted this way, the assumption is made that all differences in the measured trajectory executions are due to human corrections. To validate this assumption, the repeatability of the trajectory execution with the Impedance controller had to be tested. Some tests that validate this repeatability, as well as the entire correction-extraction system are shown in Appendix B. In this appendix is concluded that, even though the controller is deterministic, there is some randomness in the simulation environment. That being said, it also concludes that the environment is deterministic enough to assume that all trajectory differences are the result of corrections. This appendix also finds that applying corrections to a single DOF through impulse-forces also affects some other DOFs. This could cause issues in extracting the intention of impulse-force corrections. Appendix B further shows that this issue is non-existent, when a PD-controller is used for the oracle, instead of feed forward impulse forces.

3.3 General algorithm structure

The way of extracting corrections is a limiting factor in the way the general algorithm can be set up. It makes the learning episodic and offline in nature, since a $\boldsymbol{\xi}_{nc}$ is non-trivial to define for a continuous task, and a pre-recorded $\boldsymbol{\xi}_{nc}$ would become invalid in an online learning scenario. A pseudo-code for the general structure of the proposed offline episodic learning algorithm can be found in Algorithm 1.

In this algorithm $\boldsymbol{w}_p$ are the weights for the nonlinear forcing function of the DMP system, $\boldsymbol{w}_k$ are the weights for the stiffness representation, $\boldsymbol{\xi}_{ref}$ is the reference trajectory, which is obtained from a rollout from the DMP system, $\boldsymbol{\xi}_{nc}$ is a recording of an execution of the current

Algorithm 1: General Algorithm Pseudo-code

Initialize DMPs with initial demonstration $\rightarrow \boldsymbol{w}_p$; `// Section 2.2.2`
Initialize stiffnesses with uniform low value $\rightarrow \boldsymbol{w}_k$; `// Section 3.1`
while *Behaviour not converged/satisfactory* **do**
  $\boldsymbol{\xi}_{ref} \leftarrow$ Reference trajectory for current policy, rollout from DMP system
  $\boldsymbol{\xi}_{nc} \leftarrow$ Record execution of current policy
  $\boldsymbol{\xi}_{corr} \leftarrow$ Record execution with interactive corrections ; `// Section 3.2`
  for *every timestep* t **do**
    $E_{a,t} = \xi_{corr,t} - \xi_{nc,t}$
    $E_{r,t} = \xi_{corr,t} - \xi_{ref,t}$
    Compute ΔK and ΔP according to the rules of the selected algorithm (PASTIL or HA-PASTIL), based on $E_{a,t}$ and $E_{r,t}$; `// Sections 3.5 and 3.6`
    $\boldsymbol{\Delta w}_K = \Delta K \boldsymbol{\phi}_K(t)$
    $\boldsymbol{\Delta w}_P = \Delta P \boldsymbol{\phi}_P(t)$
    $\boldsymbol{w}_k$+=$\boldsymbol{\Delta w}_K$
    $\boldsymbol{w}_p$+=$\boldsymbol{\Delta w}_P$
  end
end

policy, $\boldsymbol{\xi}_{corr}$ is a recording of a performed correction, and $E_{a,t}$ and $E_{r,t}$ are error parameters, that will be used in the update rules. ΔK and ΔP are intended trajectory updates that shall be the result of these rules. $\boldsymbol{\Delta w}_K$ and $\boldsymbol{\Delta w}_P$ are weight updates for weight-based representations, of which $\boldsymbol{\phi}_K(t)$ and $\boldsymbol{\phi}_P(t)$ are the corresponding features.

This algorithm follows the general structure of the described way of extracting the corrections. It performs two system executions per iteration, one without corrections, and one where the human teacher is allowed to apply corrections. From the two collected trajectories, along with the reference trajectory, some error parameters and corrections are extracted, which are then fed to the desired algorithm (PASTIL or HA-PASTIL, see Sections 3.5 and 3.6). The desired algorithm then returns two update vectors, one for the positional trajectory, and one for the stiffness trajectory. These update vectors are then translated into weight updates for the respective representations, which are then applied.

This form of weight updates stems from a general gradient descent approach. If the final representation, $r(t, \boldsymbol{w})$, is a linear combination of the weights and the features, as $\boldsymbol{w} \cdot \boldsymbol{\phi}(t)$, then the derivative of this representation, with respect to the weights is

$$\nabla r(t, \boldsymbol{w}) = \boldsymbol{\phi}(t)[11]. \tag{3.1}$$

The algorithm described in this pseudo-code is executed separately for all 6-Cartesian DOFs. This execution is synchronized between the DOFs using the DMP time parameter. A block diagram that further illustrates this algorithm can be found in Figure 3.2. This figure illustrates how all the parts work together to form a complete learning algorithm. In this way it also outlines the general loop structure.

3.4 Segmentation of the correction space

Now that a policy representation has been established, and the corrections can be extracted, it is time to look at a way of updating the policy based on the corrections. This is where the main contribution of this work truly begins.

To be able to learn from these corrections, an intention-estimation scheme was designed. This scheme tries to learn the stiffnesses by making an estimation of the intention behind the

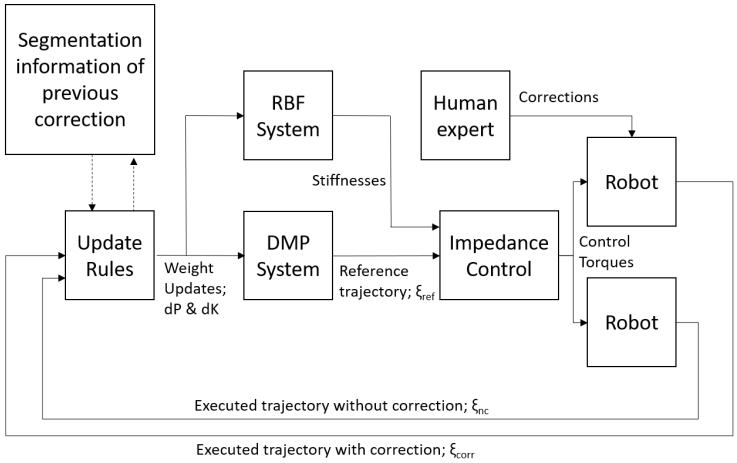


Figure 3.2: Block diagram of the general algorithm, where the RBF system represents the stiffnesses, and the DMP system represents the trajectories.

applied correction.

To estimate these intentions, the choice was made to segment the correction space, based on distance from he previous execution and the reference trajectory. “Ideal Behaviours" for any of the segments of the correction space were thought out, which inspired update rules. The segmentation is shown in Figure 3.3.

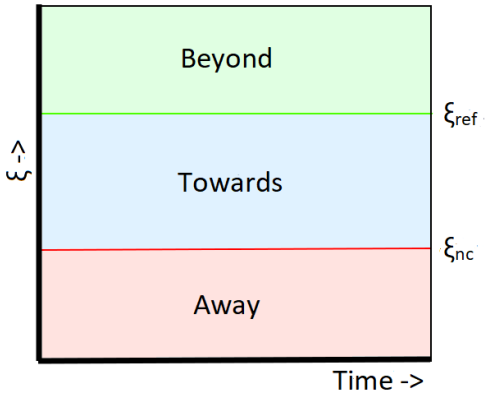


Figure 3.3: Segmentation of the correction space

Figure 3.4: Demonstration of the segmentation of the correction space.

In this image $\boldsymbol{\xi}$ represents the DOF for which the correction is applied, $\boldsymbol{\xi}_{ref}$ is the reference for the DOF, and $\boldsymbol{\xi}_{nc}$ is the execution without correction. The $\boldsymbol{\xi}$ parameters in this image correspond to the parameters in Figure 3.1. An even more visual representation is supplied in Figure 3.4. In this image examples are given for ξ_{ref}, ξ_{nc} and ξ_{corr}, to which the segmentation is applied.

If, as seen in the left part of Figure 3.4, the corrected execution is moved from $\boldsymbol{\xi}_{nc}$ in the direction of $\boldsymbol{\xi}_{ref}$, but further away, it is deemed to be *‘Beyond’*. If it is between $\boldsymbol{\xi}_{ref}$ and $\boldsymbol{\xi}_{nc}$, as seen in the middle section of Figure 3.4, it is deemed *‘Towards’*. If the correction is moved away from $\boldsymbol{\xi}_{ref}$, as seen in the right section of Figure 3.4, it is deemed *‘Away’*.

For a more mathematical definition of the segments, the segmentation parameter $\lambda_{seg,t}$ is introduced, according to

$$\lambda_{seg,t} = \frac{\xi_{corr,t} - \xi_{ref,i}}{\xi_{nc,t} - \xi_{ref,t}}. \tag{3.2}$$

As suggested by the sub-scripted t, this parameter is calculated at every timestep. $\lambda_{seg,t}$ can be used to determine the segment of a correction, at a certain timestep, by using

$$\begin{aligned} \lambda_{seg,t} < 0 &\rightarrow Beyond \\ 0 < \lambda_{seg,t} < 1 &\rightarrow Towards \\ \lambda_{seg,t} > 1 &\rightarrow Away \end{aligned} \tag{3.3}$$

These segments are introduced in an attempt to capture the difference in intention of different correction modes. Take, for example, a robot that has to lift an object with unknown mass to a constant height. In this situation, the $\boldsymbol{\xi}_{nc}$ will probably be lower than the $\boldsymbol{\xi}_{ref}$. If a human expert would want to reinforce the reference, a correction would be made *'Towards'* $\boldsymbol{\xi}_{ref}$, while if the expert would want to alter the reference, a correction would be made either above and *'Beyond'* the reference, or downwards, *'Away'* from the reference.

From this example is seen that, in this case, a difference in intention for correcting stiffnesses or positions, leads to a different correction space segment selected by the human expert.

3.5 PASTIL

If the aforementioned segmentation is used to estimate the intention of these corrections, and each of these segments is linked to a different set of update rules, a learning algorithm can be obtained. These update rules can be based on an estimation of the intention of a correction, as stated in the previous example. For instance, in the *'Towards'* region is assumed that the current reference is still the desired reference, therefore the stiffness probably needs to be increased to get to the reference. This effectively leads to the first, and most simple, version of the proposed algorithm: Position And Stiffness Teaching with Interactive Learning (PASTIL). This algorithm increases the stiffness when a *'Towards'* correction is made, increases the stiffness, and changes the reference trajectory, when a *'Beyond'* correction is made, and changes only the reference position when an *'Away'* correction is made. These updates are based on the magnitude of the different error parameters, as well as the feature activations of the internal RBF representation. The rules for this algorithm are shown in Table 3.1. In the 'Expected effects' column of this table, the estimated intention of the correction is shown. For instance, 'K:↑' means that the system estimates that the user wants to see the stiffness increased, and 'P:-' means that the system estimates that the current reference position should not be moved.

Segment	Expected Effects	Update Rule	Explanation
Beyond	K: ↑P:↑	$\Delta K = \alpha \cdot \lvert E_{a,t} \rvert$ $\Delta P = \beta \cdot E_{r,t}$	The correction is beyond the reference, so the current reference needs to be adjusted, while the stiffness also needs some adjustment, to allow the execution of a trajectory above the reference
Towards	K: ↑ P:-	$\Delta K = \alpha \cdot \lvert E_{a,t} \rvert$ $\Delta P = 0$	The reference seems correct, while the stiffnesses need to be adjusted.
Away	K: - P:↓	$\Delta K = 0$ $\Delta P = \beta \cdot E_{a,t}$	The reference seems to be incorrect, while nothing can be said about the stiffnesses

Table 3.1: Update rules for PASTIL

In this table K is the stiffness, P is position, ΔK is the intended update for the stiffnesses, ΔP is the intended update for the trajectories, $E_{r,t}$ is the difference between the correction trajectory and the reference trajectory, $E_{a,t}$ is the difference between the correction trajectory and the executed trajectory, α is the learning rate for the stiffnesses, and β is the learning rate for the positions. Both learning rates are hyperparameters that can be set.

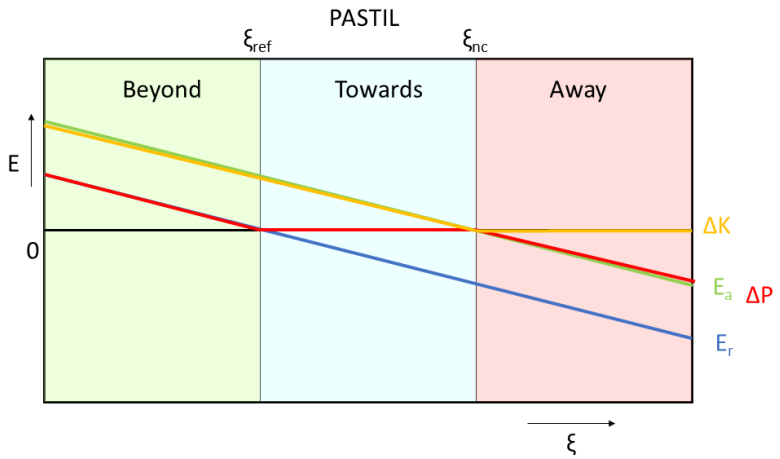


Figure 3.5: Visual representation of the rules for PASTIL

In Figure 3.5 a visual representation of the rules is shown. This representation gives more insight in how the rules connect to each other. From Table 3.1 and Figure 3.5 is seen that there is no rule that reduces the stiffness. The reason for that is that it is often non-trivial to estimate the intention based on a current correction only. One might argue to reduce the stiffness when corrections are very far away from the reference, or to reduce the overall stiffness by a set percentage each round to balance the occurring increases. These are, however, both ineffective solutions. Very far corrections are quite counter-intuitive, and might be hard to accomplish when the stiffness is already large. Reducing the overall stiffness might also have unwanted side-effects, as the same corrections shall have to be made over and over to keep the stiffness at the desired level.

3.6 HA-PASTIL

PASTIL does not account for temporal variations in the corrections. It does not know, or care, about any previous corrections. Information from previous corrections can be very important in estimating the intention of certain parts of a correction. For instance, if a correction is inconsistent with the previous correction, this could be a sign that this part of the trajectory is not important for the task, or that the teacher changed the objective of the task. In both cases it is undesirable to increase the stiffness in the current update round.

This is where History Aware PASTIL (HA-PASTIL) comes in. In this algorithm, segmentation information is stored about the previous correction for the every timestep. This information can then be used to further estimate the intention of the corrections, by checking if they are consistent with previous corrections. The rules for this algorithm are shown in Table 3.2.

Additionally to the symbols in Table 3.1, a γ is introduced. γ is a secondary learning parameter, which is set to $0.5 \cdot \beta$. The secondary learning rate is included as a way to make a compromise between the current trajectory and the correction trajectory. This is used in cases where the current correction is very inconsistent with the previous correction. In this case, it is likely that the current correction does not represent an 'optimal' path, but merely a suggestion of where the robot should go.

Previous Correction	Current Correction	Expected Behaviour	Update Rule	Explanation
Beyond	Beyond	K: ↑/-; P:↑	$\Delta K = \alpha \cdot \max(0,(\lvert\text{offset}\rvert - \lvert E_{r,t}\rvert))$ $\Delta P = \beta \cdot E_{r,t}$	The reference seems to be quite incorrect, and needs to be adjusted, whilst nothing can be said about the stiffness
Towards	Beyond	K: ↑/↓; P:↑	$\Delta K = \alpha \cdot (\text{offset} - \lvert E_{r,t}\rvert)$ $\Delta P = \beta \cdot E_{r,t}$	If the Correction is quite close to the reference, the stiffness should be increased, while, if the correction is far away from the reference, it should be decreased.
Away	Beyond	K: ↓; P:↑	$\Delta K = \alpha \cdot \max(-3,(2 \cdot \lvert\text{offset}\rvert - \lvert E_{r,t}\rvert))$ $\Delta P = \gamma \cdot E_{r,t}$	The Correction seems to be inconsistent with the previous correction, so the stiffness should be decreased, and the position should be set somewhere in the middle
None	Beyond	K: ↑/↓; P:↑	$\Delta K = \alpha \cdot (\text{offset} - \lvert E_{r,t}\rvert)$ $\Delta P = \gamma \cdot E_{r,t}$	If the Correction is quite close to the reference, the stiffness should be increased, while, if the correction is far away from the reference, it should be decreased. The position probably should not be changed too much, since it was apparently at the right position during the previous execution.
Uninitialized	Beyond	K: -; P:↑	$\Delta K = \alpha \cdot \max(0,\text{offset} - \lvert E_{r,t}\rvert)$ $\Delta P = \beta \cdot E_{r,t}$	The initial demonstration seems to be incorrect at this point, so the position should be changed, whilst nothing can be said about the stiffness
Beyond	Towards	K: ↑; P: -	$\Delta K = \alpha \cdot \lvert E_{a,t}\rvert$ $\Delta P = 0$	While in the previous correction the reference might have been wrong, in the current correction the reference is being reinforced, so the stiffness should be increased, while the reference should not be moved
Towards	Towards	K: ↑; P: -	$\Delta K = \alpha \cdot \lvert E_{a,t}\rvert$ $\Delta P = 0$	Both corrections are reinforcing the reference, so the stiffness should be increased, while the reference should not be changed
Away	Towards	K: ↑; P: -	$\Delta K = \alpha \cdot \lvert E_{a,t}\rvert$ $\Delta P = 0$	While in the previous correction the reference might have been wrong, in the current correction the reference is being reinforced, so the stiffness should be increased while the reference should not be moved
None	Towards	K: ↑; P: -	$\Delta K = \alpha \cdot \lvert E_{a,t}\rvert$ $\Delta P = 0$	Although the behavior was deemed satisfactory during the previous correction, an increase in stiffness seems to be required
Uninitialized	Towards	K: ↑; P: -	$\Delta K = \alpha \cdot \lvert E_{a,t}\rvert$ $\Delta P = 0$	The initial stiffness might not have been satisfactory, so it should be increased
Beyond	Away	K: ↓; P:↓	$\Delta K = -\alpha \cdot \lvert E_{a,t}\rvert$ $\Delta P = \gamma \cdot E_{a,t}$	The Correction seems to be inconsistent with the previous correction, so the stiffness should be decreased, and the position should be set somewhere in the middle
Towards	Away	K: ↓; P:↓	$\Delta K = -\alpha \cdot \lvert E_{a,t}\rvert$ $\Delta P = \gamma \cdot E_{a,t}$	The Correction seems to be inconsistent with the previous correction, so the stiffness should be decreased, and the position should be set somewhere in the middle
Away	Away	K: -; P:↓	$\Delta K = 0$ $\Delta P = \beta \cdot E_{a,t}$	The reference seems to be quite incorrect, and needs to be adjusted, whilst nothing can be said about the stiffness
None	Away	K: ↓; P:↑	$\Delta K = -\alpha \cdot \lvert E_{a,t}\rvert$ $\Delta P = \gamma \cdot E_{a,t}$	The Correction seems to be inconsistent with the previous correction, so the stiffness should be decreased, and the position should be set somewhere in the middle
Uninitialized	Away	K: -; P:↓	$\Delta K = 0$ $\Delta P = \beta \cdot E_{a,t}$	The initial reference should be adapted, while nothing can be said about the stiffness.
Any	None	K: -; P: -	$\Delta K = 0$ $\Delta P = 0$	If no correction is performed, nothing should be changed, no matter the history

Table 3.2: Table with the logic behind the suggested learning algorithm, where offset $= |E_{r,t} - E_{a,t}|$, α, β and γ are learning rates.

The *'Away'-'Beyond'* rule also contains a few additional hyperparameters, to prevent the stiffness loss from getting too large. More research is needed to see their effect, and to validate their necessity. These rules have been designed to increase stiffness if the system is repeatedly corrected to the same location, and to decrease the stiffness if the corrections are inconsistent and do not converge to an optimal policy. A more intuitive visual representation of some of the rules can be found in Figures 3.6 and 3.7. These dependency plots have also been made for the other correction cases, to further visualise the consistency, and the effects, of the rules. These figures, along with their explanations, can be found in Appendix D.

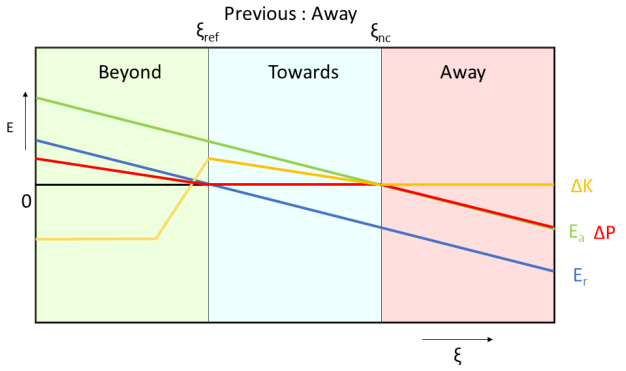


Figure 3.6: Visual representation of the rules for when the previous correction was *'Away'*

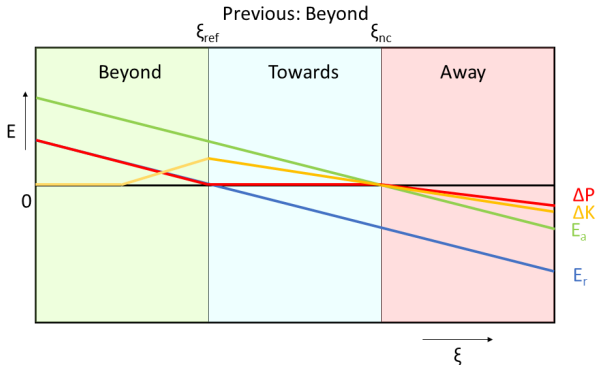


Figure 3.7: Visual representation of the rules for when the previous correction was *'Beyond'*

In these figures the location of the correction is placed on the x-axis, while the magnitude of different dependent variables is placed on the y-axis. These figures show how the rules connect with each other, and their transitions.

In Figure 3.6 the three rules, for the case that the previous correction was *'Away'*, are represented. These rules make an increase in stiffness when the new correction is *'Towards'*, since this means that the previous correction adjusted the reference to the correct position, and the reference has to be tracked more accurately.

If the correction is *'Beyond'*, the position is increased with half the normal learning rate, to allow finetuning. This is the basic overall strategy for conflicting positional corrections. When *'Beyond'*, the stiffness is only increased, if the new correction is very close to the reference, and decreased if the correction is further away from the reference. This rule also has a saturation value, to avoid losing too much stiffness. If the reference has to be adjusted a lot, it is probably not the intention to drop the stiffness to a very low value. This rule still contains some hyperparameters, that could be removed in a future version of the algorithm.

Figure 3.7 represents the three rules that control the policy updates in the case that the previous correction was *'Beyond'*. This plot shows that the stiffness is increased if the new correction is closer to the reference than the executed trajectory, not updated if the correction is further *'Beyond'*, and decreased if the new correction is further away. The position is increased if the correction is *'Beyond'*, and decreased with half the learning rate, if the correction is *'Away'*.

4 | Experimental Setup

In this chapter the experimental setup for this thesis is described. This setup has been designed to validate the performance of the proposed algorithms. Due to the COVID-crisis, and the novel nature of the algorithm, all experiments have been performed in a virtual environment, with virtual oracles. To test whether the algorithm does indeed correctly learn trajectories and stiffnesses, three tasks have been designed. Each task highlights specific features of the algorithm. These three tasks are outlined in this chapter, along with the baseline algorithms to which the algorithm's performance will be compared. The development of this method can be considered a success if the learning algorithm obtains a lower cost in the selected tasks than any state-of-the-art interactive position-only, or position and stiffness, algorithm. Tests shall also be done to see whether the learned stiffness is an improvement over a constant stiffness.

To experimentally test the performance of PASTIL and HA-PASTIL three experiments were designed: The Push task, the Lift task, and the Policy Reuse Push task. In the Push task a box has to be pushed off of a table as fast as possible, while it lands as close to the table as possible. In the Lift task, a box has to be lifted to a constant height, while the mass changes, and in the Policy Reuse Push task, the flexibility of the algorithms is tested by changing the objective of the task halfway through the learning process. In this chapter the task setup, the cost function, the oracle setup and the initial trajectory for these tasks shall be described in detail.

4.1 Environment

The validation of the algorithm shall be done in a virtual Gazebo environment. In this environment a Franka-Emika Panda robot is controlled using a Cartesian Impedance controller. Since a simulated environment with an Impedance-controlled robot was not freely available, a custom Impedance controller has been implemented and designed for this thesis. This Impedance controller supports stiffness values ranging from 10 to 1500. Below 10 the tracking is not suitable to perform tasks, while above 1500 the controller starts to introduce a lot of oscillations. In Appendix C a study has been done that shows that with these stiffness bounds the robot can always be corrected by an average human, in terms of maximal exerted force.

This controller does not perform path planning, since this was not trivial to implement. Creating such an environment also seemed out of the scope of the current research. The learning algorithm and the controller were implemented as ROS nodes, to make an eventual transfer to a real robot arm possible. The choice for gazebo was made because of the way it integrates with ROS and the community support. Also, there already was an existing ROS package that supplied some of the required functionality. More information about the simulators and environments that were considered can be found in Appendix E. More information about the chosen environment can be found in Appendix F.

All corrections for all experiments were made by manually tuned oracles. Quite some research has gone into designing a suitable interface to apply the corrections manually, or through a user study, but none such interface was found, and the idea was abandoned. The results of this study are presented in Appendix G.

4.2 Baseline Algorithms

Two algorithms have been used as baseline algorithms. Since there is no directly comparable method in the literature, some methods had to be adapted, in order to fit the learning task. The first baseline algorithm, representing the group of position-only-algorithms is an incremental learning algorithm, loosely based on the COACH algorithm from [25]. According to my literature study, COACH is one of the state-of-the-art algorithms for interactively learning position-only tasks. This algorithm incrementally learns positions, with an adaptive learning rate, based on the coherence of the corrections. This method had to be adapted to fit the situation of this thesis, since it is online in nature, while the teaching setting in this thesis is offline in nature. The representation is set to be for the position is the same as for PASTIL and HA-PASTIL, only the update rules were adjusted in the general algorithm.

The second baseline algorithm, representing the Imitation Learning (IL) algorithms, is the "mean-var" algorithm. This is an algorithm that collects all the demonstrations and corrections in one large database, and treats the corrections as additional demonstrations. It then bases the position profile off of the mean of the dataset, while the stiffnesses are based off of the variance of the dataset. This approach was created for the express purpose of this thesis, but is inspired by approaches like DAgger [29], and the approaches from [37] and [38].

4.3 Push Task

The first of the three validation tasks is the "Push task". In this task the robot has to push a box off the table as fast as possible, while the box has to land as close to the table as possible. When the box crosses a set line on the table, it increases its mass to a random value.

These objectives were chosen, as they are inherently contradictory, and need to be balanced for a good task score. To throw the box off of the table as fast as possible, it would be easy to learn a very stiff controller, with a reference very far from the edge of the table. This would, however, fling the box very far away, which is deemed as an undesirable outcome. If, on the other hand, the main goal is to only make the box land as close to the table as possible, it would be easy to move it very slowly.

The mass change was added to test how the learned policies react to stochastic perturbations, these kinds of perturbations are expected to occur quite often in non-standardized tasks, and need to be dealt with gracefully.

This task tests the ability of the system to be taught a policy that remains accurate in the face of stochastic perturbations. The policy also cannot have either saturated maximum, or saturated minimum stiffness, since this would result in poor task performance. This means that it tests whether the system is able to learn a middle-ground stiffness.

In the real world, this task would relate to most accurate manipulation tasks, with stochastic perturbations. The mass-change is implemented to emulate a friction change on the sliding surface, as if the box were to slide from one surface, onto another, more rough, surface.

The setup used for this task can be seen in Figure 4.1. A schematic drawing, along with some measurements is shown in Figure 4.2.

This section shall give further details about the task setup. First the environment will be described, after which the mass change shall be explained. Then the cost function and the oracle shall be outlined. Finally the initial demonstration, and the expectations shall be discussed.

4.3.1 World Description

In the gazebo world, the Panda robot, indicated by a grey circle in the schematic of Figure 4.2, is placed at the origin, facing in the positive X direction. All measurements in the gazebo

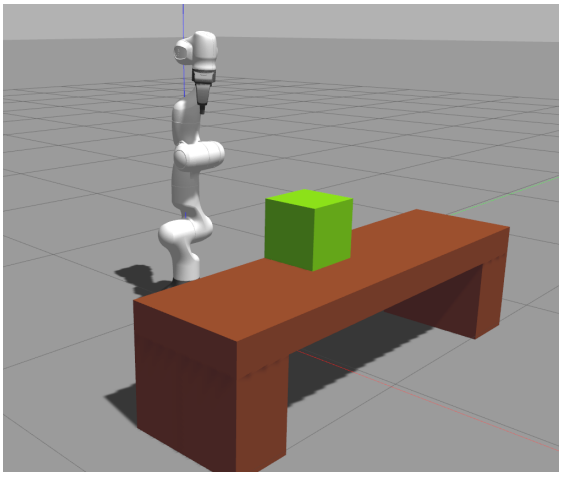


Figure 4.1: Environment setup for the Push task. The box has to be pushed off of the table, towards the right, while the mass changes.

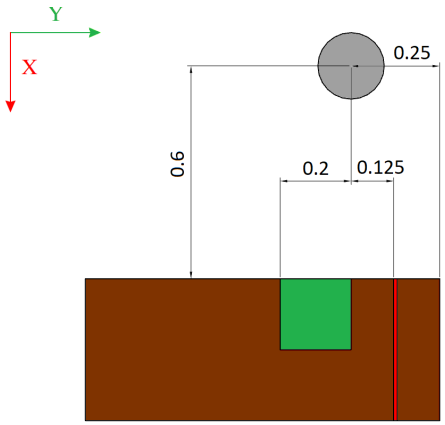


Figure 4.2: Schematic of the environment setup for the Push task. The box changes mass, when the center passes the red line.

world are set to be in meters. For this task, the robot is set to an initial position, where the end effector is on the negative Y side of the green box. The box has to be pushed towards the positive Y, until it reaches the end of the table and falls off. The box is 0.2 m by 0.2 m by 0.2 m, and is initially spawned with its center at (0.7,-0.1,0.5) m. The edge of the table is located at Y = 0.25, so the box has to be moved at least 35 cm before it can fall off.

4.3.2 Mass Change

The box changes its mass during this task. The mass starts as 1 kg, but when the center of the box crosses the red line, as seen in Figure 4.2, the mass gets set to a random value taken from a uniform distribution between 2 kg and 3 kg. This change demonstrates the ability of an algorithm to deal with stochastic disturbances, such as the random weight of the box, while also maintaining positional accuracy. The mass change was implemented to emulate a friction change, as in sliding a box over different materials. This emulation was necessary, since implementing an actual friction change in the gazebo environment did not seem feasible.

4.3.3 Cost Function

The cost function for this task consists of three parts, one part that represents the distance that the box landed from the table, one part that represents the time that the box took to get to the ground, and one part that represents the average learned stiffness. To obtain a good score, speed has to be balanced with how far the box is flung, while keeping the stiffnesses as low as possible. This cost function is defined by

$$\frac{W_0 c_y + W_1 c_t + W_2 c_s}{W_0 + W_1 + W_2}, \tag{4.1}$$

where the W parameters are weights, c_y is the cost for the distance from the table, c_t is the cost for the time, and c_s is the score for the stiffness.

The weights vector $[W_0, W_1, W_2]$, in the case of this task, is set to [2,1,1]. This setting was chosen to put more emphasis on the final Y-location. This part of the task performance was deemed the most important, since it represents the final state of the environment. The robot could, for instance, be placed in an industrial environment, where the box would have to fall onto a conveyor belt close to the table. It would be beneficial if it could push more boxes per hour, but if these boxes would miss the conveyor, the entire system is worthless.

The Y-location cost, c_y is determined by measuring the distance from the center of the box to the edge of the table, in meters. This distance is then normalized by dividing it by a 'worst case performance' of 1 m. This obtains a cost that ranges roughly from 0 to 1.

The time cost c_t is determined by measuring the time in seconds from the start of the episode, to the point where the center of the box falls below 0.15 m. For the time, a maximum and a minimum expected value have been determined, called t_{max} and t_{min}. These are then used to normalize the cost, according to

$$\frac{t - t_{min}}{t_{max} - t_{min}}. \tag{4.2}$$

In the case of this task t_{min} was set to empirically determined value of 2, while t_{max} was set to 5.2, the length of a full episode.

The final part of the score, c_s, is determined by taking the average of the full stiffness trajectories, in the X, Y and Z directions. These are all averaged into a single score, which is then normalized, by dividing it by the maximum allowed stiffness of 1500.

4.3.4 Oracle

For this task an oracle was set up that continually corrects the robot towards the same point, which is slightly above the edge of the table. The oracle performs these corrections by exerting external forces on the robot end-effector, in a similar way that the end user would do this. The oracle uses a PD controller to determine the magnitude of the corrections. Using the study on interaction forces done in Appendix C, the oracle has been tuned to exert maximum forces similar to what a human would be able to apply.

This oracle is expected to increase the performance on this task, since the PD-controller used is over-dampened. This means that it slowly approaches the desired point, instead of going there as fast as possible and flinging the box far away.

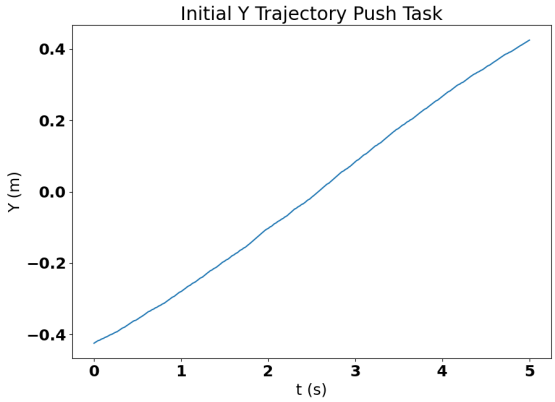


Figure 4.3: Initial Y trajectory for Push task.

4.3.5 Initial Trajectory and Expectations

The initial supplied reference trajectory is shown in Figure 4.3. Although this trajectory looks like it is a straight line, it actually is part of a $\frac{1}{8}^{th}$ circle around the origin. The other DOFs are configured to be part of the same circle, with the Z at an almost constant value. This trajectory was chosen, to avoid running into joint limits, while still making a movement.

The final learned trajectory of the algorithm is expected to reach a higher Y-value a bit faster, while the other axes should not change much. For this task the analysis shall be mostly focused on the Y-dimension, and, unless some very strange unexpected effects occur, the other dimensions will not be discussed.

4.4 Lift Task

The second task is the “Lift Task". In this task, the robot has to lift a box to a pre-set height, and move it in a $\frac{1}{8}^{th}$ circle on this height, while the mass changes. This task is implemented as a stiffness focused task. This task tests the ability of the system to learn stiffnesses, without affecting the positional trajectory.

4.4.1 World Description

The setup for this task is shown in Figure 4.4, while the initial trajectory for the Z-axis is shown in Figure 4.5.

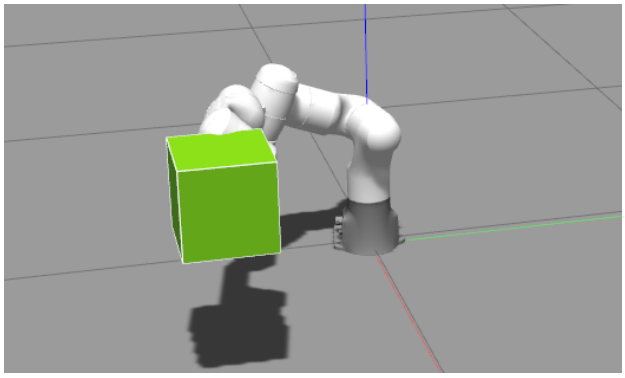


Figure 4.4: Environment setup for the Lift task. The box needs to be lifted to this constant height, while the mass changes.

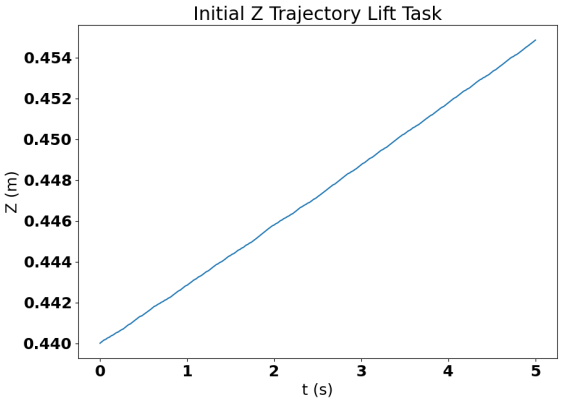


Figure 4.5: Initial trajectory for the Lift task.

From Figure 4.4 is seen that the world consists of a Panda robot in the origin, with a box attached to the end-effector link. In this world the robot shall execute a trajectory in the XY-plane, while maintaining a constant height in the Z-direction, at Z=0.45 m. This trajectory is not blocked, or influenced by other objects in the world.

4.4.2 Mass Change

In this task, the box also changes its mass at a set time, to a random value. The box starts out with a mass of 1 kg, and after 1.6 seconds the weight gets set to a value taken from a uniform distribution, between 2 kg and 3 kg. This mass change represents, for instance, a waiter holding a tray to which an object is added. The goal for this task is to keep the box as close to the set height as possible.

4.4.3 Cost Function

The cost function of this task consists is a weighted average of three parts, and is composed according to

$$\frac{W_0 c_{MSE} + W_1 c_{maxz} + W_2 c_s}{W_0 + W_1 + W_2}. \tag{4.3}$$

In this task all weights in $[W_0, W_1, W_2]$ are set to 1, since all objectives are directly related. Part one, c_{MSE} is adapted from the MSE of the executed Z-trajectory with the desired “optimal" Z-trajectory. This MSE cost is normalized by dividing by the empirically obtained value of 0.1, to obtain c_{MSE}.

The second part c_{maxz} is adapted from the maximal deviation from the path in Z-direction. c_{maxz} is determined by dividing this deviation by 0.45, the maximum possible downward de-

viation. These two parts represent the task performance, on a global (entire trajectory), and local (maximum deviation) level.

The last part is an average of the X-, Y- and Z-stiffness of the trajectory, a lower stiffness is desired throughout the trajectory for interaction safety. These three scores are normalized and combined in a weighted average. To minimise this cost, the stiffness has to be balanced with with task performance. For the interactive learning approaches, the assumption is made that the teacher does this implicitly.

Note that this cost function, with the exception of the stiffness part, only considers the Z-dimension. This choice was made, since the effects that this task is meant to demonstrate are expected to be the strongest on the Z-axis.

4.4.4 Oracle

For this task an oracle was set up as well. This oracle only corrects in the Z-dimension. It constantly corrects the robot to a height of 0.45 m, which is the height that was chosen for the cost function. The oracle uses a PD-controller to determine the magnitude of the corrections. The corrections are applied to the end effector as external forces. Corrections from this oracle are expected to increase to the task performance, because they constantly push the robot back to the correct height, without overshoot into the beyond region.

4.4.5 Initial trajectory and Expectations

The initial trajectory is shown in Figure 4.5. Note that the initial trajectory increases very slightly, instead of being constant. This has to do with the fact that the pydmps library, that was used for the DMP representation of the trajectory, does not handle straight trajectories well. It is also unable to handle a trajectory for which the end point is exactly the same as the start point.

In this task the learning of the correct stiffness should be the most important, since the robot has to deal with a stochastic perturbation, while keeping the box steady. The expectation is that the algorithm does not change the positional trajectory, while it learns an increased stiffness after the mass change, to account for the higher mass of the box.

4.5 Policy Reuse Push Task

The last task, the 'Policy Reuse Push task', is derived from the 'Push task'. In the environment of this task, as seen in Figure 4.6, two boxes are present. The initial goal of this task is the same as with the Push task, pushing the rightmost box off of the table, towards the right. After a few iterations where the Oracle reinforces this task, the oracle decides that it wants to push the leftmost box of the table towards the left instead. A successful completion of learning this task would show the flexibility of this incremental learning algorithm. It would prove that an end-user can change his/her mind about the goal of the task, without having to re-start the algorithm. This property is useful for two main reasons.

Firstly, it makes the algorithm robust against erroneous feedback. The effect of a wrong or mistaken correction will fade away over multiple corrections, instead of influencing the behaviour forever after.

Secondly, it can be used for transfer learning. If the robot has previously learned a certain task, and the new task is similar in some ways, the previously learned task can be used as a starting point, to decrease the amount of episodes needed in teaching.

4.5.1 World Description

The world of this task consists of the Panda robot, which is located at (0,0), with a 1 m wide table, placed 0.4 m away from it, in the positive x-direction. The edges of this table are located at 0.5 m, and -0.5 m in the Y-direction. On this table are two boxes, that will have to be pushed of during this task. These boxes are located 0.15 m from the origin in the positive and negative Y-direction. The size of the boxes is 0.2 by 0.2 by 0.2 m. A detailed overview, with some of the most important measurements can be found in Figure 4.7.

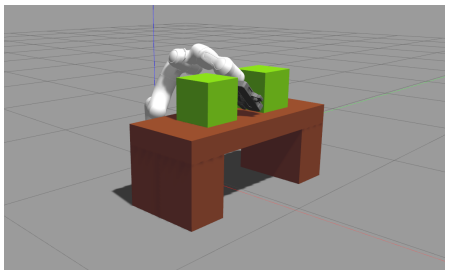


Figure 4.6: Environment setup for the Policy Reuse Push task. This task is similar to the Push task, but at a certain point the oracle changes its mind about which box has to be pushed off the table

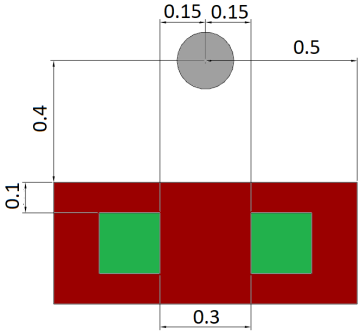


Figure 4.7: Overview of the environment of the Policy Reuse Push task, with some of the most important measurements.

4.5.2 Cost Function

The cost functions of these boxes is the same as with the box for the Push task, as shown in Section 4.3.3, consisting of a part, c_t that measures the time, and a part that measures the distance from the table, c_y, and a part for the stiffness c_s. The main difference is, that, this time around, there are two separate cost functions, instead of just one. One cost for the initial task, and another for the final task. For the initial task, the cost for the Y-location of the box is calculated with respect to the right side of the table, as seen in Figure 4.6, while for the final task the cost is calculated with respect to the side of the table that is on the left-hand side in the image. A penalty is also added for cases where the box did not leave the table. This penalty is constant in its magnitude of 0.5.

4.5.3 Oracle

The oracle for this task differs quite a lot from the oracles in the other tasks. For this task, an oracle was chosen that holds a ‘desired trajectory" and corrects the robot to this trajectory, using a PD-controller. This means that, in contrast to the other oracles, this oracle does not always correct to the same point, but has a reference that changes over time. This kind of oracle was chosen due to the increased complexity of the task. After a certain number of corrections, 5 in this case, the oracle changes the desired trajectory, by mirroring it around the Y-axis. This means that the starting point is still the same, since the Panda robot starts in Y = 0 m.

The initial oracle reference trajectory is the same as the initial demonstrations provided to the algorithm, but sped up 1.5 times. This trajectory was chosen, since it seems to capture the essence of the task quite well, and the speedup was added to slightly decrease the time component of the cost, with respect to the initial trajectory.

Due to this speedup, the oracle finishes the reference trajectory before the end of the episode. This is solved by setting the final point of the reference trajectory as a PD-controlled point attractor, when the oracle finishes the trajectory.

Images of the initial and final oracle reference trajectories are shown in Figures 4.8 and 4.9.

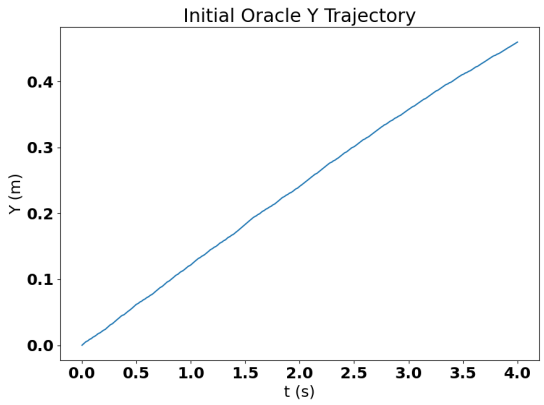


Figure 4.8: Initial oracle trajectory for the Policy Reuse Push task

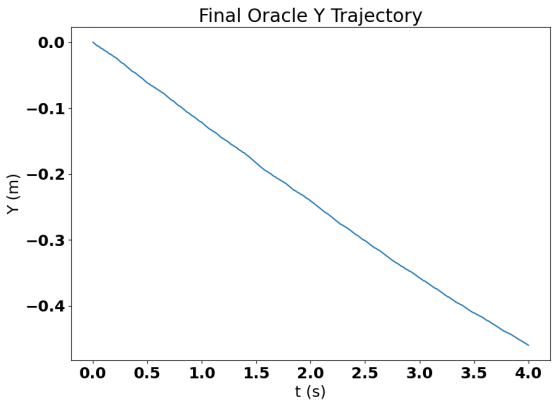


Figure 4.9: Final oracle trajectory for the Policy Reuse Push task

4.5.4 Initial trajectory and Expectations

Figure 4.10 shows the initial Y trajectory provided to the algorithms for the Policy Reuse Push task. This trajectory, just like in the previous tasks, is a part of a circle around the origin.

The expectation for this task is, that the algorithm learns a trajectory that goes towards the negative Y, about as fast as the final oracle trajectory. For the stiffness, the expectation is that the stiffness is higher when the robot is in contact with the box, while the stiffness remains low at points where there is no contact.

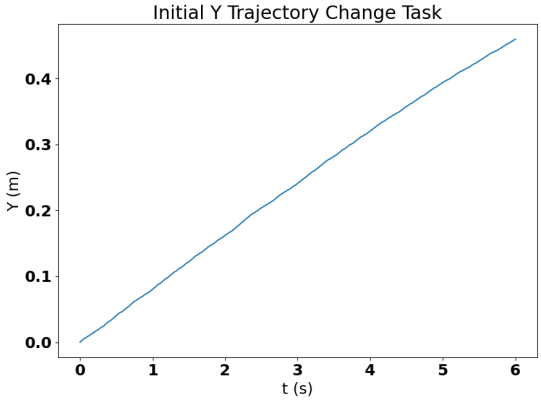


Figure 4.10: Initial trajectory supplied to the learning algorithms

5 | Results

In this chapter the results of the previously discussed test shall be shown and discussed. First up, the results of the Push task are presented. Secondly, the results for the Lift task are shown. Then the results of the Policy Reuse Push task are treated. Finally, some variations of the algorithms are tested on the Policy Reuse Push task, to further investigate the properties and the behaviour of the algorithm. All experiments were done using ROS melodic and Gazebo 9, on an Ubuntu 18.04 laptop with an Intel i7 7700HQ processor, an NVIDIA Quadro M1200 GPU, and 16 GB RAM.

5.1 Push task

The goal of the Push task is to learn to push a box off of a table, as fast as possible, while the box lands as close to the table as possible. To see how well the algorithm performed, the learned trajectory and stiffness shall be discussed, after which the resulting task-cost, and a comparison against baseline algorithms shall be shown. This test has been repeated 20 times, for statistical stability.

5.1.1 Analysis

In Figure 5.1 the learned reference is shown, this reference is an average of 20 learning episodes, and plotted with its variance. If this reference is compared to the initial reference, as seen in Figure 4.3, can be seen that the Y increases slightly faster, and the robot thus pushes the box off faster. It also shows that the planned Y-velocity between Y 0.1 and Y 0.3, the location near the edge of the table, is a lot lower than for the initial demonstration, which might be beneficial in making the box land closer to the table.

The learned stiffness profile is seen in Figure 5.2, this stiffness profile is also an average of 20 learning episodes, and plotted with its variance. From this figure is seen that the stiffness increases after the initial contact is made, peaks at around the time that the mass changes as expected, and drops after that. The time where the mass changes is shown in Figures 5.1 and 5.2 as the vertical red line. This behaviour is in line with the expected behaviour from Section 4.3.5. In this section the expectation was set that the algorithm would learn to push the box off of the table a little bit faster. This plot also shows that the maximum stiffness ($\sim$112) is still quite low, since the stiffnesses range from 0 to 1500.

In Figure 5.3 a breakdown is shown of the costs for the Push task, these costs are the average costs of 20 runs, plotted with their variance as errorbars. This figure shows that both the cost for the time and the cost for the Y-location of the box go down, and seem to converge after 4-5 correction rounds. This indicates that the algorithm successfully learns to improve performance on both these sub-goals of the task, without the stiffness rising dramatically.

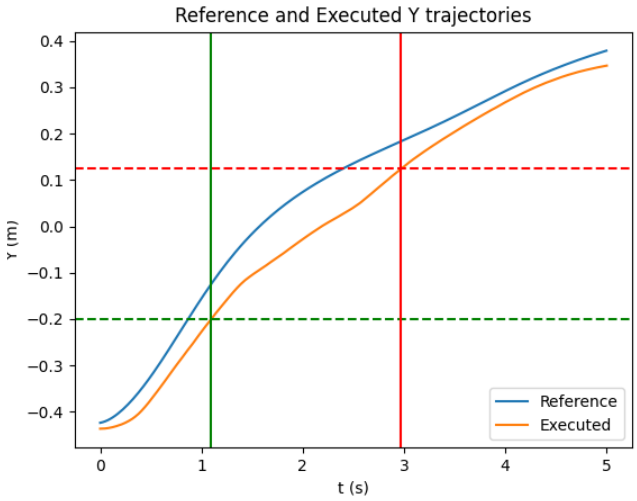


Figure 5.1: Final executed Y-trajectory, and learned reference, for the Push task. The vertical lines represent events that happened during the execution, where green is the initiation of contact, and red is the change of the mass. The green horizontal line represents the initial location of the box, the red one represents the location where the box changes weight.

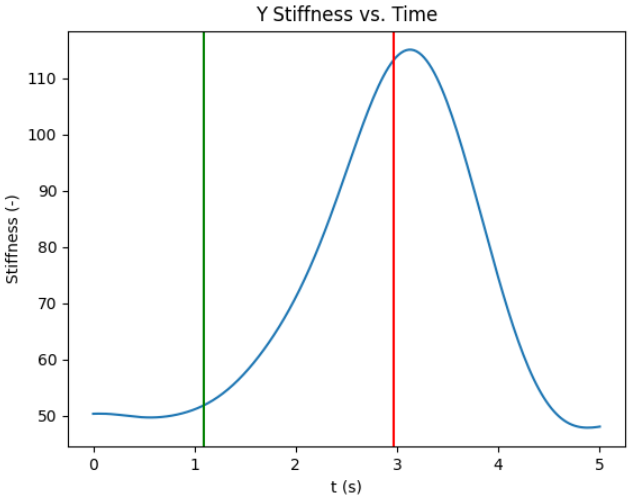


Figure 5.2: Learned Y-stiffness for the Push task. The vertical lines represent events that happened during the execution, where green is the initiation of contact, and red is the change of the mass.

Figure 5.3: Breakdown of the cost for HA-PASTIL on the Push task. The green part represents the time it takes for the box to be pushed off the table, while the yellow part represents the distance of the final position of the box to the table. The blue part represents the cost for the stiffness.

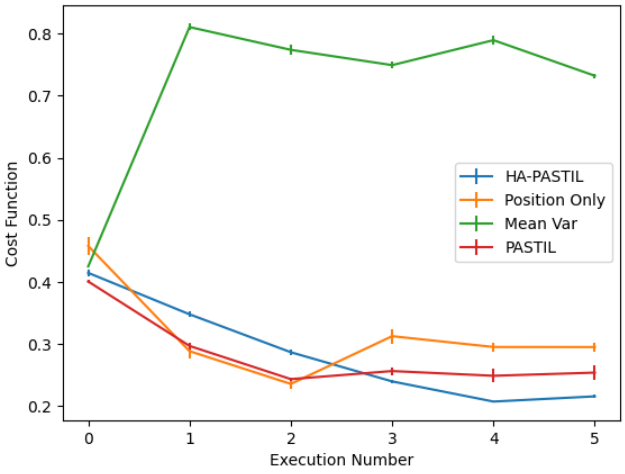


Figure 5.4: Comparison of the evolution of the algorithm performance for the Push task, over multiple correction episodes, plotted with the variance.

5.1.2 Comparison

While the results from the previous paragraph validate the behaviour of the algorithm to some extent, a comparison with the baseline algorithms discussed in Section 4.2 is required to further prove this algorithm's capabilities. A graph containing the costs for all algorithms, for different numbers of training episodes can be found in Figure 5.4, as an average of 20 runs. The errorbars in this plot represent the variance. This figure shows that the costs of the positional, PASTIL, and HA-PASTIL all go down, and converge within the five trials. From these algorithms HA-PASTIL obtains the lowest cost. The high cost for the mean-var algorithm is mostly due to the high stiffness that it learns, if stiffness would not be accounted for in the task performance, the performance would be in line with the other algorithms. A breakdown of all the costs of all the algorithms can be found in Appendix H.

Algorithm	Final Cost	Standard Deviation	Percentage with respect to HA-PASTIL	P-Value with respect to HA-PASTIL
HA-PASTIL	0.22846	3.987E-2	100%	-
PASTIL	0.25422	0.1142	111.3%	0.3471
Mean-Var	0.73229	6.149E-2	320.5%	< 0.0001
Position Only	0.29539	8.563E-2	129.3%	0.0030

Table 5.1: Results for the Push task, with variance

The final costs of all the algorithms, along with their variances, are displayed in Table 5.1. This table also shows the obtained scores as a percentage of the score obtained by HA-PASTIL, to be used for comparison purposes. From this table is seen that the P-value of the difference between PASTIL and HA-PASTIL is 0.3471. This number represents the chance to obtain these test results, while the actual expected results are the same. This means that the difference between HA-PASTIL and PASTIL is not statistically significant for this test, but mostly due to the high variance of PASTIL.

When the same calculation is done for HA-PASTIL, and the position-only algorithm, a P-value of 0.0030 is obtained, meaning that, with a 95% confidence bound, the difference is significant.

5.2 Lift Task

This section shall present the results for the Lift task, analyse the learned trajectories, and compare the behaviour to the baseline algorithms. The goal of the Lift task is to validate the performance of the stiffness learning system, since it is expected that the algorithm will not learn anything for the positional part. This test was repeated 20 times, for statistical stability.

5.2.1 Analysis

The Z-trajectory, as learned by HA-PASTIL, is plotted in Figure 5.5, along with the final executed trajectory. This graph shows that the learned trajectory is the same as the initial trajectory, as presented in Section 4.4.5. This is to be expected, since the corrections will always be towards the reference, thus only increasing the stiffness. The final executed trajectory shows a slight dip in the position, after the mass change. The deviance from the planned path, however, never exceeds 8 cm. The noise on the trajectory can be attributed to the controller, which starts to introduce some oscillations at higher stiffness levels. The stiffnesses show a significant increase from the base values. They are also higher after the weight change, than

before the weight change, as was expected. This is an indication that the learned stiffness profile might be a better profile than a uniform stiffness distribution. Note that this learned stiffness is a lot higher than the one learned for the Push task.

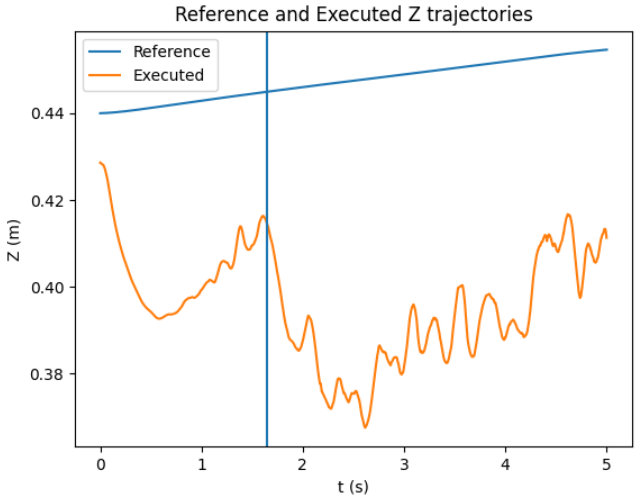


Figure 5.5: Learned Z-trajectory, and final executed trajectory, for the Lift task. The vertical line represents the change of mass.

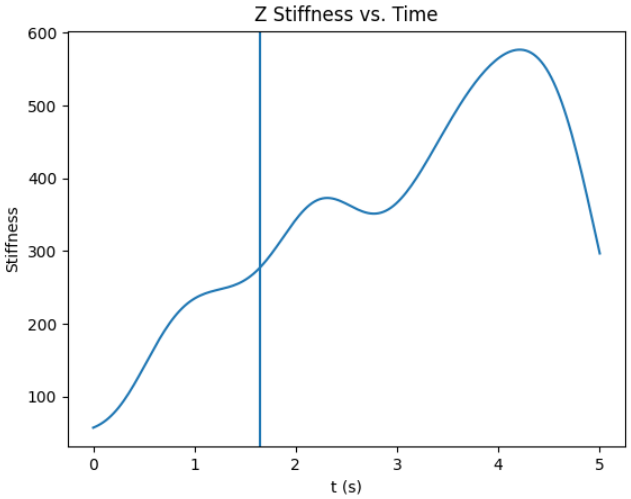


Figure 5.6: Learned Z-stiffness for the Lift task. The vertical line represents the change of mass.

To further analyse the learned behaviour of HA-PASTIL, a breakdown of the costs obtained during the learning of the tasks is shown in Figure 5.7. This figure shows that both the MSE cost, and the maximum deviation cost reduce in a smooth asymptotic way, while the stiffness increases slightly. This seems to indicate that the algorithm is able to capture all aspects of this task rather well.

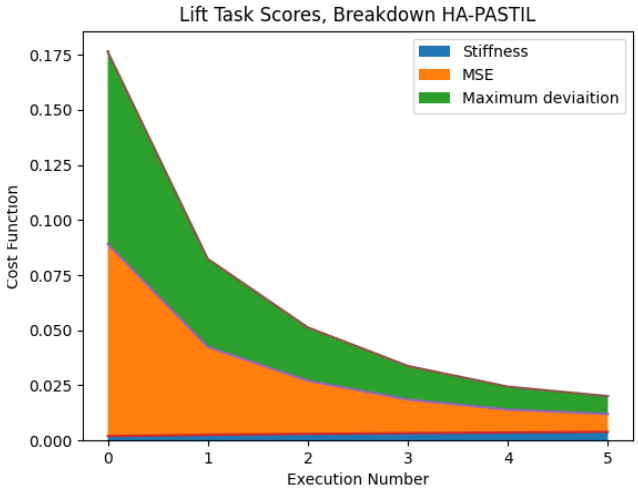


Figure 5.7: Breakdown of the costs for HA-PASTIL on the Lift task.

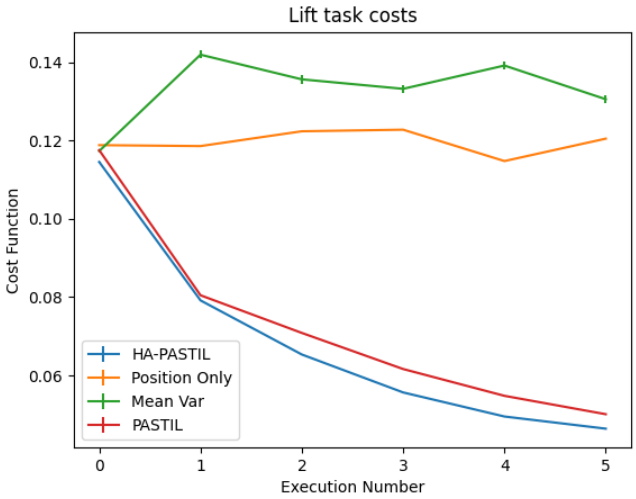


Figure 5.8: Comparison of the evolution of the algorithm performance for the Lift task, over multiple correction episodes, plotted with the variance.

5.2.2 Comparison

Algorithm	Final Cost	Standard Deviation	Percentage with respect to HA-PASTIL	P-Value with respect to HA-PASTIL
HA-PASTIL	0.046393	2.635E-3	100.00%	-
PASTIL	0.050087	3.554E-3	107.96%	0.7246
Mean-Var	0.130532	3.464E-2	281.36%	< 0.0001
Position Only	0.120460	9.271E-3	259.65%	< 0.0001

Table 5.2: Results for the Lift task, with variance

A comparison of the cost obtained by HA-PASTIL, with the costs obtained by the baseline algorithms is shown in Figure 5.8, plotted alog with the variance. This figure shows that, while the performance is quite similar, HA-PASTIL is able to outperform PASTIL by a small margin. The mean-var algorithm does not perform well on this task, mainly due to the high stiffness values it learns. The positional algorithm does not seem to learn anything that affects the performance. A further analysis of the behaviour of the baseline algorithms can be found in Appendix H.

A numerical comparison between the final costs of the algorithms is shown in Table 5.2. If a statistical analysis is done, for the difference between the results of HA-PASTIL and PASTIL, a P-value of 0.7246 is obtained. This means that the difference is not statistically significant, if a 95% confidence bound is desired. The P-value for the difference between the results of HA-PASTIL and the position only algorithm is lower than 0.0001, meaning that the difference between the results of these algorithms are significant.

5.3 Policy Reuse Push Task

In this section the results for the Policy Reuse Push task are shown, and shortly discussed. First an analysis of the learned trajectory and stiffness is presented, after which a breakdown of the scoring is shown. Finally the performance is tested against the baseline algorithms. Due to computational complexity, this test has only been repeated 10 times.

5.3.1 Analysis

Figure 5.9 shows the trajectory that was learned by HA-PASTIL, along with the final executed trajectory. The trajectory seen in this graph differs a lot from the initial trajectory, as presented in Section 4.5.4. The final executed trajectory corresponds quite well to the oracle trajectory presented in Section 4.5.3. This trajectory moves to the edge of the table quite fast, after which it remains stationary at that position. A plot of the learned stiffnesses is shown in Figure 5.10. In this figure the initiation of contact with the box is shown with a green vertical line, while the loss of the contact is shown with a red vertical line. From this plot is seen that the stiffness has its highest point, just after the initiation of the contact, and goes down after the contact is lost. This is roughly in line with the expectations from Section 4.5.4. There is, however, still a smaller peak in the stiffness, after the contact is lost. This peak might be here due to the non-sparse corrections of the oracle.

A breakdown of the costs assigned to the performance of HA-PASTIL can be found in Figures 5.11 and 5.12. In these images, the black vertical line represents the change in correction objective. From these images is seen that HA-PASTIL manages to capture the change in objective quite quickly. This is illustrated by the fact that after round 8, three rounds after the objective was changed, the penalty is no longer present in Figure 5.12.

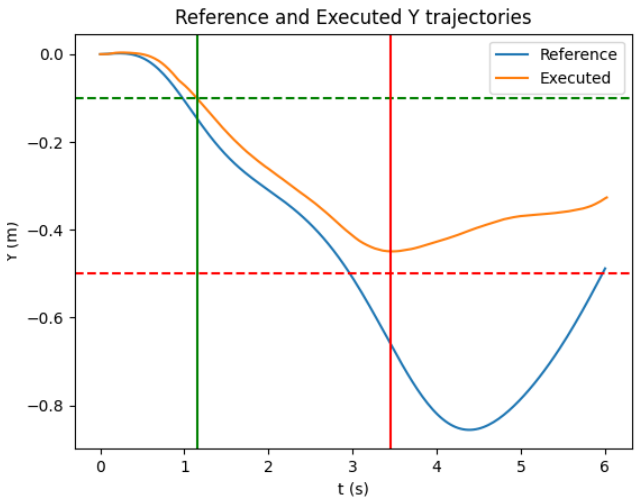


Figure 5.9: Learned reference, and final executed, Y-trajectory for the Policy Reuse Push task. The horizontal lines represent object in the environment, green is the initial location of the box, and red is the end of the table. The vertical lines represent the changes in contact, as measured during the final execution, where green is initiated contact and red is lost contact.

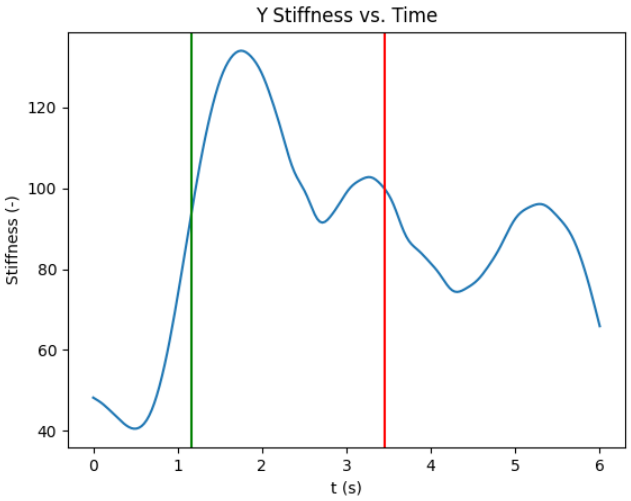


Figure 5.10: Learned Y-stiffness for the Policy Reuse Push task. The vertical lines represent the changes in contact, as measured during the final execution, where green is initiated contact and red is lost contact.

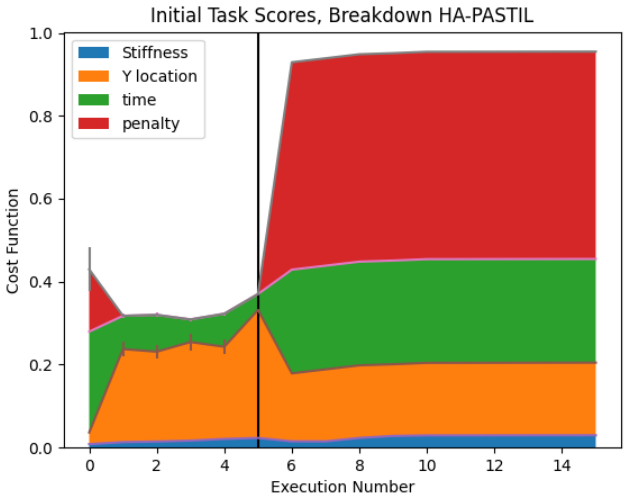


Figure 5.11: Breakdown of the costs of HA-PASTIL for the initial task of the Policy Reuse Push task

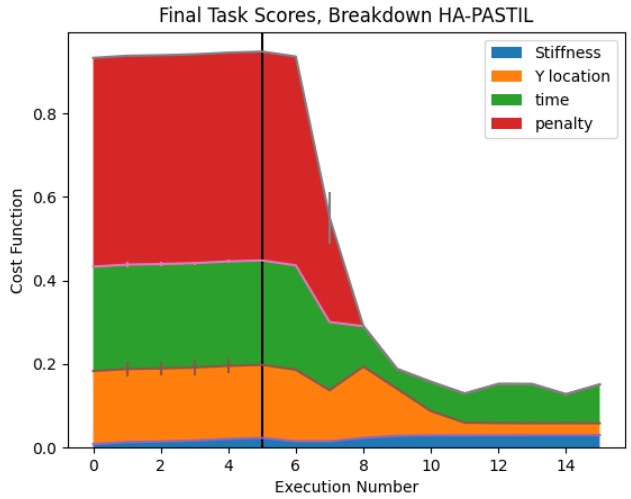


Figure 5.12: Breakdown of the costs of HA-PASTIL for the final task of the Policy Reuse Push task

5.3.2 Comparison

A comparison of HA-PASTIL against the baseline algorithms is presented in Figures 5.13 and 5.14. Figure 5.13 shows the costs for the initial task, while Figure 5.14 shows the costs for the final task. From these images is seen that both PASTIL and HA-PASTIL are able to reliably capture the change of the task, within a few correction rounds. From these two algorithms HA-PASTIL outperforms PASTIL in the last few corrections. While the position only-algorithm tries to capture the change in objective, it does not seem to be able to do so reliably, while the mean-var algorithm does not seem to capture it at all. Breakdowns of these cost graphs, and a further explanation of the effects can be found in Appendix H.

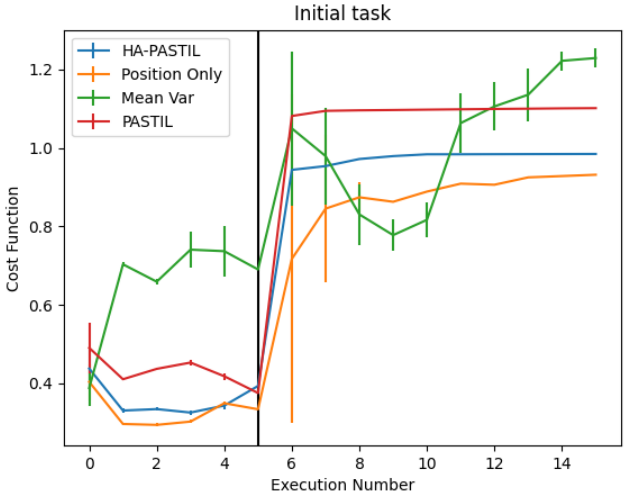


Figure 5.13: Evolution of the costs for all the algorithms for the initial task in the Policy Reuse Push task.

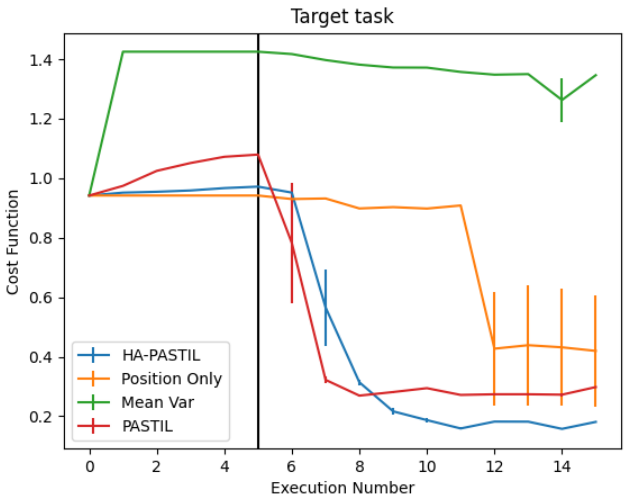


Figure 5.14: Evolution of the costs for all the algorithms for the final in the Policy Reuse Push task.

Algorithm	Final Cost	Standard Deviation	Percentage with respect to HA-PASTIL	P-Value with respect to HA-PASTIL
HA-PASTIL	0.151220	4.959E-2	100.00%	-
PASTIL	0.298018	5.062E-2	197.08%	< 0.0001
Mean-Var	1.346117	4.515E-2	890.17%	< 0.0001
Position Only	0.419553	0.4598	277.44%	0.0788

Table 5.3: Results for the final task of the Policy Reuse Push task, with variance

A numerical representation of the results of the Policy reuse Push task is presented in Table 5.3. The P-value for the difference between the results of HA-PASTIL and PASTIL is lower than 0.0001, meaning that the difference is significant.

5.3.3 Tests on variations of the algorithm

To further analyse the performance of HA-PASTIL, a short ablation study was done on the Policy Reuse Push task. This task was chosen for the ablation study, since it is the most complicated task in this thesis, and is expected to show the most significant results.

There are two parts to this ablation study, the first part shall initialize HA-PASTIL with a rather high stiffness, to see if it is able to recover from that. The second part shall investigate the performance of the algorithm, while the stiffness learning is fully disabled, to prove that the learned stiffnesses have a positive effect on the behaviour of the system. This part shall also test the execution of the final learned trajectory with different constant stiffnesses, to further analyse the added value of the learned stiffness profile. These tests can be considered successful, if the learned stiffness is proven to be more beneficial to the task performance than a constant stiffness.

Initialization with high stiffness

To test the robustness of the system and to see how well it recovers from wrongly learned policies, a study was done. In this study HA-PASTIL had to learn the Policy Reuse Push task, while being initialized with a relatively high stiffness. For this stiffness a value of 1000 was chosen, instead of 50. This means that the algorithm had to unlearn both the initial trajectory in the Y-dimension and the high-initialized stiffness profiles, to test whether lowering the stiffness is as easy as increasing the stiffness. The learned trajectory can be found in Figure 5.15, while the learned stiffnesses can be found in Figure 5.16. The goal of this test is to see if the algorithm is able to recover from high stiffness levels. This test can be considered to be successful, if the stiffness is reduced to an acceptable level, while the other parts of the task-cost are comparable to those of the low-stiffness-initialisation scenario.

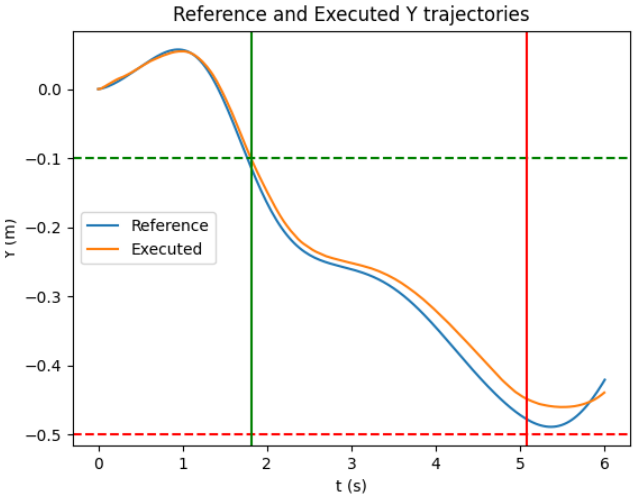
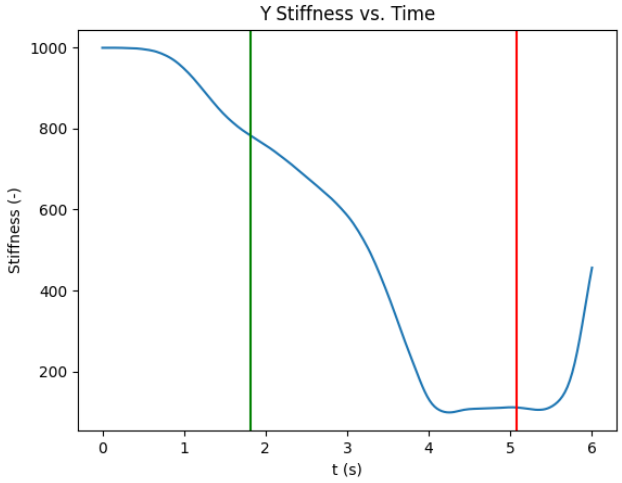


Figure 5.15: Learned trajectory, for the Policy Reuse Push task, with high initial stiffness.



4

Figure 5.16: Learned stiffness, for the Policy Reuse Push task, with high initial stiffness.

When these trajectories and stiffnesses are compared to the trajectories and stiffnesses learned by HA-PASTIL with the recommended low-stiffness initialization, as seen in Figures 5.9 and 5.10, a rather large difference can be observed. Although the algorithm did manage to lower the stiffnesses in the second half of the task, the stiffness remains at the initial value of 1000 during the first part of the task. This is most likely caused by the very small corrections applied at this time. If the corrections are smaller that a pre-set threshold, the algorithm assumes that the policy is correct, and it does not apply any changes. The small corrections

at this time are caused by a combination of the high stiffness, the small difference between the two tasks, and the force based oracle. Due to the small difference between the initial trajectory and the desired trajectory, the oracle will try to correct the robot with a small force, which is quite ineffective against the stiff robot.

Also note that the trajectory still starts off by moving in the positive Y direction, this might be caused by the high stiffness in this part, combined with the relatively small correction.

For the evaluation of the performance of the high stiffness initialisation, a breakdown of the final task costs can be found in Figure 5.17, while a comparison of the final costs with the low stiffness initialised HA-PASTIL can be found in Figure 5.18.

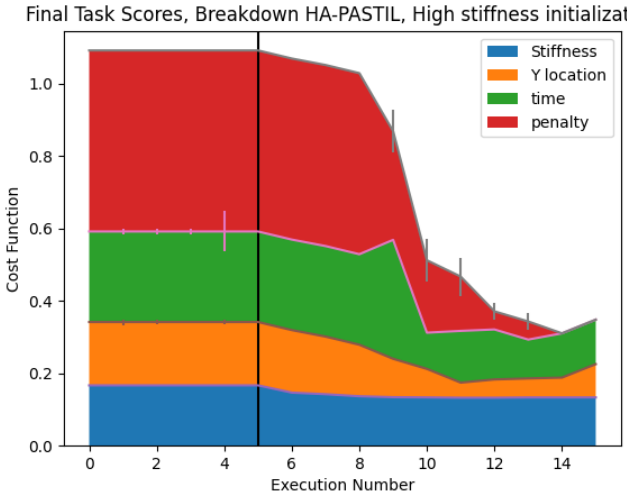


Figure 5.17: Final task cost breakdown, for the Policy Reuse Push task, with high initial stiffness.

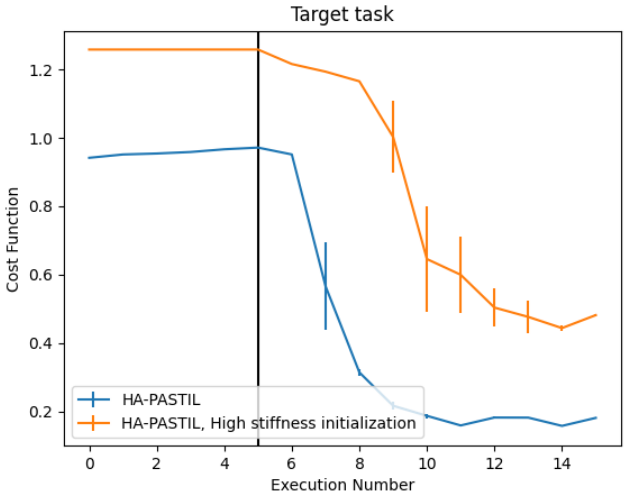


Figure 5.18: Final task cost comparison, for the Police Reuse Push task, with high initial stiffness.

From these two figures can be seen that the high -stiffness initialised algorithm learns slower, and also retains some of the penalty, until almost the last trial, meaning that it needs all 15 trials to learn to consistently push the box off of the table. Due to the higher stiffnesses, the total cost is also consistently higher, even at the end. If the stiffness-cost is removed from the total cost, however, the final performance is comparable between the scenarios. This means that, in a less extreme case, the algorithm is not fully robust against wrongly applied corrections, both for the stiffnesses, and for the positions.

Some of these effects might also be explained by the way the corrections are applied. An application on a real robot, with an actual human applying the corrections may be needed to further investigate whether this is an issue with the algorithm, or an issue with the oracle.

Effects of learned stiffness

In order to review the significance of the learned stiffnesses, a few tests were done. Firstly a test was done with HA-PASTIL, where the learning of the stiffnesses was disabled, so the stiffnesses were fixed to the initial value of 100. None of the other task parameters were changed for this test. This test can be considered successful if the task performance with stiffness learning is better than the task performance without stiffness learning.

The resulting trajectories can be found in Figure 5.19. In this plot the green vertical line represents the time at which contact is made with the box, the green horizontal line shows the initial location of the box, and the red horizontal line shows the location of the edge of the table. Note that the learned reference trajectory is quite similar in shape as the one learned, while the stiffness learning was active, which is seen in Figure 5.9. The final executed trajectory,

however, differs quite a bit, due to the effect of a lower stiffness. Also not the absence of the vertical red line. This line is not shown in the figure, because there is not a reliable point at which the contact is lost. The final executed trajectory does not make it to the end of the table.

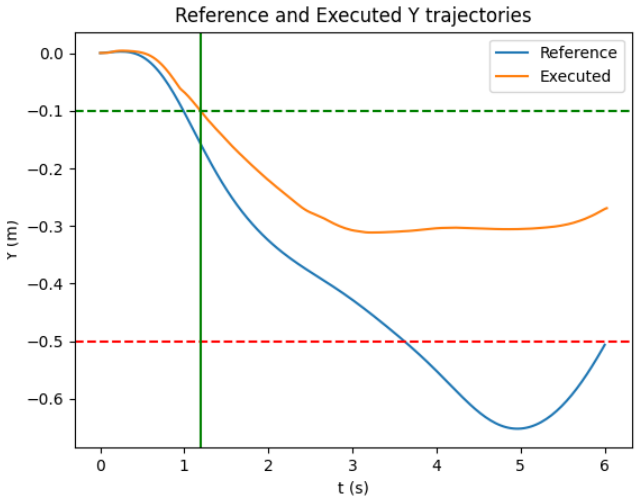


Figure 5.19: Learned trajectory, for the Policy Reuse Push task, without stiffness learning.

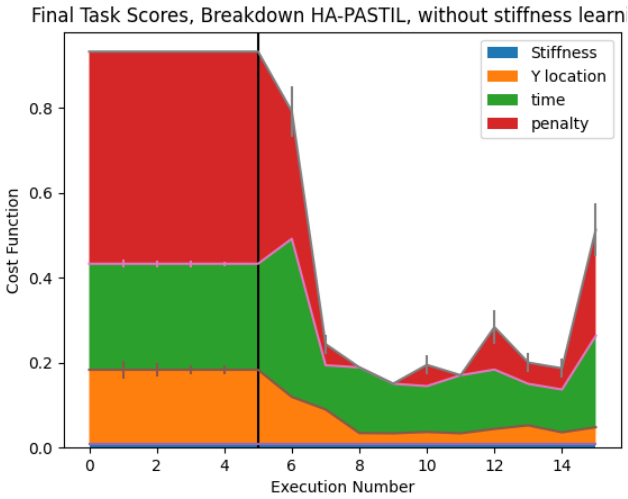


Figure 5.20: Cost Breakdown, for the Policy Reuse Push task, without stiffness learning.

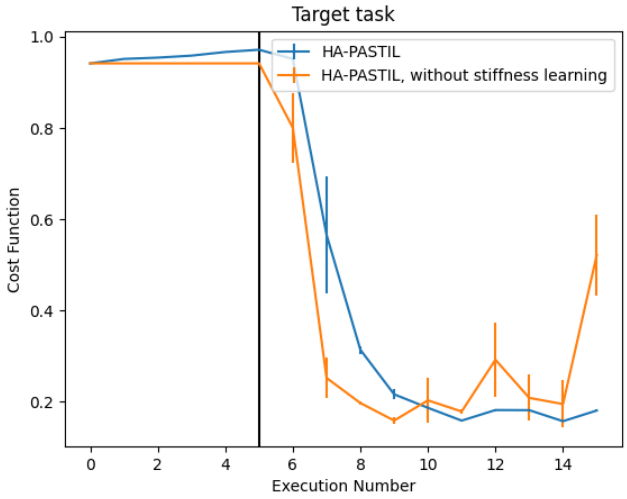


Figure 5.21: Cost Comparison, for the Policy Reuse Push task, without stiffness learning.

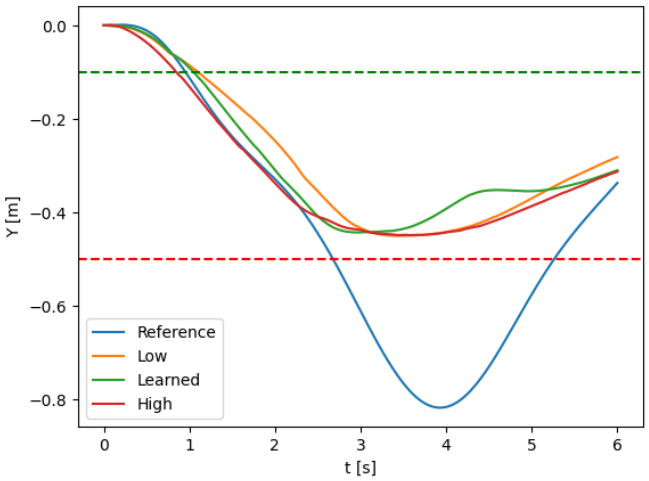


Figure 5.22: Executed trajectory for different stiffnesses.

To evaluate the effect of the stiffness on the cost, the costs for the final task are presented, the scores for the initial task are not shown for this experiment, since these are expected to show smaller differences. A breakdown of these task costs is shown in Figure 5.20, while a comparison with unmodified HA-PASTIL is found in Figure 5.21, in these figures the vertical black line represents the change in teacher policy. From the breakdown figure is seen, from the presence of a penalty in the last trial, that without stiffness learning, the algorithm is not able to learn to consistently push the box off of the table, within the given 15 trials. From the comparison graph is seen that, without stiffness learning, and thus with a lower overall stiffness, the algorithm learns faster. This is in line with the effect observed in the trial with high initial stiffness, where an algorithm with a higher stiffness learned slower. This effect is, in part, due

to the force based corrections from the oracle. The conclusion of this trial is that the algorithm performs better with stiffness learning, than without, as seen from the task cost graphs.

To analyse the effects of the learned stiffnesses even further, another trial was conducted. In this trial the final learned trajectory from regular HA-PASTIL was taken, and combined with multiple different constant stiffness trajectories. This is different from the previous trial, since in this trial, the positional profile was learned with the stiffness learning enabled, to enable the effects of jointly learning the stiffness and the position. These newly obtained policies were then executed, and the executed trajectories were observed, along with the task costs. For the new stiffness values, a high value, and a low value were chosen. The high value was set to 1500, while the low value was set to 50. The resulting executed trajectories can be found in Figure 5.22.

Comparing the trajectories in this figure, the loss of contact with the box seems to lie at around the 3 second mark for all trajectories. This is roughly when the Y location goes beyond Y = 0.45 m, and the mass center of the box goes over the edge of the table. Until this point, the trajectory with the learned stiffnesses is closest to the high-stiffness trajectory, while the stiffnesses are closer to the low stiffness. This means that the learned stiffnesses can push the box off of the table slightly faster than the low stiffnesses, but not by a large margin. A comparison, and breakdown of the costs can be found in Table 5.4.

	Low	Learned	High
Cost	0.15436 ± 1.48E-13	0.15418 ± 1.49E-13	0.28189 ± 5.7E-9
Y-Cost	0.02826 ± 1.48E-13	0.02861 ± 1.49E-13	0.02818 ± 5.7E-9
T-Cost	0.12410 ± 0.0	0.11071 ± 0.0	0.10671 ± 0.0
Penalty	0.0 ± 0.0	0.0 ± 0.0	0.01665 ± 0.0
Stiffness Cost	0.002 ± 0.0	0.0148703 ± 0.0	0.147 ± 0.0

Table 5.4: Breakdown of the costs of the different stiffnesses, average of 10 trials

From this table is seen that, the leaned stiffness obtains the lowest cost, albeit by a small margin. The main advantage over the low stiffness is in the time part of the cost function, while the main advantage over the high stiffness is in the stiffness part. Due to the extremely small variances, the P-value of the difference between the low and learned stiffness is below 0.0001, meaning that the difference is statistically significant. A sidenote here is that, even though it is taken into account in the P-value, the sample size for this test was 10, which is rather low. This might mean that the assumption of the test data being normally distributed does not hold.

5.4 Conclusion

In this chapter the results of the previously discussed tests have been shown. The conclusions for the Push and Lift Task are that the trajectories and stiffnesses learned by HA-PASTIL seem to make sense for the task, and that HA-PASTIL outperforms both baseline algorithms, and PASTIL. While the difference between HA-PASTIL ad the baseine algorithms is significant, the difference between HA-PASTIL and PASTIL is not significant for these two tasks.

For the Policy Reuse Change Task the conclusion is that both PASTIL and HA-PASTIL are able to successfully unlearn previously reinforced trajectories. In this task there was a statistically significant improvement in the performance of HA-PASTIL over PASTIL.

Some studies on the Policy Reuse Change Task have also been done with some variants of HA-PASTIL. From these test can be concluded that the algorithm is not able to fully unlearn stiffnesses that have been increased too far, which could be an area of further investigation. These trials also conclude that the stiffness learning is beneficial to the task-performance, and that the learned stiffness is an improvement over a static-low and a static-high stiffness.

6 Discussion

In an attempt to aid the future of household robotics, a method has been developed specifically for interactively learning Impedance control applications. The goal of this method is to be able to learn simple force based tasks, through a limited number of correction episodes, from a human supervisor, without explicit use of a force/torque sensor.

The method presented in this thesis has been tested against two baseline algorithms in three different example tasks. Since no comparable algorithms can be found in the literature, these baseline algorithms had to be created specifically for the purpose of this research. The results, as presented in Chapter 5, indicate that HA-PASTIL is able to decrease the cost of all tasks, and to obtain lower costs than the baseline algorithms, albeit that only the Policy Reuse Push task showed a significant difference between PASTIL and HA-PASTIL. This difference is very important, since this task demonstrates the ability to recover from wrongly learned policies, and the intended end-users shall supposedly be suboptimal in the generation of corrections. This alone, however, does not prove the effectiveness of the algorithm, since the effectiveness of the baseline algorithms was never proven.

To further validate the performance HA-PASTIL a variation study was done, which proves that HA-PASTIL is, to a certain extent, able to recover from wrongly learned policies. The study also shows that the stiffnesses, as learned by HA-PASTIL, are more beneficial to task performance, than a uniform stiffness.

These results combined together indicate that HA-PASTIL is able to successfully learn some robotic interaction tasks interactively, while being robust against some wrong corrections. This kind of learning has a lot of possible applications, where non-expert end users can easily, and cheaply, program all kinds of robotics. This could, for instance, be used for a meal-preparation robot, that has to perform several different impedance-based tasks.

6.1 Possible further research

While this work offers a beginning into a new way of interactive imitation learning, a lot still has to be done to properly validate its findings, and to further improve the algorithm.

One such thing is to search for a solution for the problem found in the high-stiffness-initialisation scenario. In this scenario, the algorithm was unable to fully recover from the faulty initial policy. This might have to do with the way the corrections are applied, but it definitely needs further investigation.

In this study, all experiments have been done in a gazebo simulation, where the simulation setup, and the robot controllers, were less than ideal. It would be beneficial for the validation of the algorithm to do some experiments on a real robot, to show that it also works in such a setting. This would also make it possible to do a user study. All experimental data generated in this work made use of oracles to mimic human corrections. This choice had to be made, due to the limited simulation environment. A lot of virtual correction interfaces have been tried and implemented, but they were deemed not intuitive enough to accurately apply corrections. An overview of these interfaces can be found in Appendix G. While the data from the oracles validates the performance of the algorithm to a certain extend, it currently is hard to differentiate between effects caused by the oracle implementation, and effects caused by the algorithm. A user study would negate some of these negative effects.

In this work, the algorithm was partly validated, by comparing the algorithm to non-validated, self-compiled, baseline algorithms, since there did not seem to be any validated

alternatives in the literature. If the algorithm were to be tested in a real-robot setting, actual human performance could be used as a validation baseline. It would be useful to see if the algorithm is able to surpass human behaviour, or of it is not able to come close to it.

Also, the current way of extracting corrections is a limiting factor in the way the algorithm can be set up. It makes the learning episodic and offline in nature, since a $\boldsymbol{\xi}_{nc}$ is non-trivial to define for a continuous task, and a pre-recorded $\boldsymbol{\xi}_{nc}$ would become invalid if the policy is updated online. In the future some research can be done to extend the algorithm to continuous, or online, learning tasks.

Another point of attention for future research would be path planning. The current setup sends the learned paths directly to the Cartesian Impedance controller, without checking if the path is kinematically feasible. This can often cause problems in high-DOF robot applications. For this thesis, the tasks were carefully chosen to not run into joint limits, or move into other problematic areas. This greatly limits the applicability of the algorithm.

A further point of investigation could be the number of hyperparameters. The HA-PASTIL algorithm currently has three “avoidable" hyperparameters. Both learning rates are probably useful to retain, for tuning purposes. The slope and saturation hyperparameters from the away-beyond rule, on the other hand, could probably be replaced for some automatically scaled metric. It would also be a good idea to investigate the way that the current hyperparameters are set. Do they need to be set per application, or can a setting be found that delivers satisfactory performance for a multitude of different tasks?

Something that should also be investigated is the scale-ability and the generalization properties of the learned trajectories and the learned stiffnesses. The trajectories should be scale-able, since they are able to use all the features of the DMP framework. The stiffnesses, on the other hand, are represented by an RBF representation, that does not hold any scaling/generalization properties. A study is needed to see of the stiffnesses should scale with the trajectory, and how they should scale.

One last point of attention could be to investigate the resource efficiency, compared to some other algorithms. While the algorithm might be relatively memory efficient, since it does not store a dataset of part corrections, it might not be as computationally efficient as it could be. Some further research could be done to see if it is necessary to improve the efficiency, and how it could be done.

7 | Conclusion

Now that all the experiments have been done, and the results have been discussed, it is time to revisit the research question, and draw a conclusion. The initial research question was "*How can a system be designed that interactively learns the stiffness gains of an Impedance controller, along with the reference trajectory, from only position feedback?*". Based on the experiments in this thesis, the answer to this question is: "*Such a system can be designed by combining pHRI with intention estimation*". In this thesis, such a system has been proposed, tested, and validated.

The main contribution of this thesis is the introduction of a novel interactive imitation learning system, designed specifically for force control tasks. The proposed system uses intention estimation to select a set of update rules, with which it updates both a positional trajectory, and a controller stiffness trajectory.

This approach has been tested in three different classes of tasks. In the first task the algorithm had to jointly learn a positional, and a stiffness trajectory, which had to work together to complete the task.

In the second task, the algorithm had to learn a stiffness profile, while explicitly not changing the reference. This task was introduced to show that the algorithm does not 'overlearn' in parts of a trajectory where nothing needs to be learned.

In the third task, the algorithm had to transfer the policy of a learned task, to a different task. This task was introduced to show the flexibility of the incremental nature of the algorithm.

The performance of the proposed algorithm was, for all tests, at least 30% better than the baseline algorithms, in the task performance score. This shows that the algorithm is able to capture these three types of tasks better than the baseline algorithms.

To further analyse the behaviour of the system an ablation study was done on the proposed algorithm. This study shows, most importantly, that the stiffnesses the system learns are beneficial to the task performance. This, together with the performance improvement over the baseline, shows that the algorithm might allow robotic systems to be instructed by users, instead of experts. This lets robotics be applied at lower cost and with less expertise needed to program.

The ablation study also shows that the system is somewhat able to recover from a very poorly initialized stiffness trajectory, albeit not fully. This is something that needs further investigation in future research.

The discussion then shows some more points of improvement for the system, and some other areas for future research. Among other things, it stresses the importance of a trial on a real robot, with real human interaction.

This system is, all in all, still somewhat of a crude start, and a lot more work needs to be done to make it useful in trivial applications, but it might someday contribute to a future of household robotics.

References

[1] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55(June 2017):248–266, 2018.

[2] Bruno Siciliano and Khatib Oussama. *Springer handbook of robotics*. Number 06. Springer, 2016.

[3] Guanglong Du, Mingxuan Chen, Caibing Liu, Bo Zhang, and Ping Zhang. Human – Robot Interaction. 65(12):9571–9581, 2018.

[4] Harish Ravichandar, Athanasios S. Polydoros, Sonia Chernova, and Aude Billard. Recent Advances in Robot Learning from Demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):297–330, 2020.

[5] Luigi Villani. Encyclopedia of Systems and Control. *Encyclopedia of Systems and Control*, (Whitney 1977):1–10, 2019.

[6] J Kenneth Salisbury. Active stiffness control of a manipulator in cartesian coordiates. *Proceedings of the IEEE Conference on Decision and Control*, 1:95–100, 1980.

[7] S Chiaverini and L Sciavicco. Force/Postion control of manipulators in task space with dominance in force. *IFAC Proceedings Volumes*, 21(16):137–143, 1986.

[8] Neville Hogan. Impedance control: An approach to manipulation. In *Proceedings of the American Control Conference*, volume 1, pages 304–313. IEEE, 1984.

[9] M T Mason. Compliance and force control for computer controlled actuators. *IEEE Transactions on Systems, Man, and Cybernetics*, 1(6):418–432, 1979.

[10] Alin Albu-Schäffer, Christian Ott, Udo Frese, and Gerd Hirzinger. Cartesian impedance control of redundant robots: Recent results with the DLR-Light-Weight-Arms. *Proceedings - IEEE International Conference on Robotics and Automation*, 3:3704–3709, 2003.

[11] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, second edition, 2017.

[12] J. Park and I. W. Sandberg. Universal Approximation Using Radial-Basis-Function Networks. *Neural Computation*, 3(2):246–257, 1991.

[13] João P.S. Rosa, Daniel J.D. Guerra, Nuno C.G. Horta, Ricardo M.F. Martins, and Nuno C.C. Lourenço. *Overview of Artificial Neural Networks*. 2008.

[14] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning attractor landscapes for learning motor primitives. *Advances in Neural Information Processing Systems*, 2003.

[15] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Probabilistic movement primitives. *Advances in Neural Information Processing Systems*, pages 1–9, 2013.

[16] Sylvain Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1):1–29, 2016.

[17] Simon Manschitz, Michael Gienger, Jens Kober, and Jan Peters. Mixture of Attractors: A Novel Movement Primitive Representation for Learning Motor Skills from Demonstrations. *IEEE Robotics and Automation Letters*, 3(2):926–933, 2018.

[18] Stefan Schaal. Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics. *Adaptive Motion of Animals and Machines*, pages 261–280, 2006.

[19] 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.

[20] AJ Ijspeert, J Nakanishi, and S Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. *Proceedings 2002 IEEE International Conference on Robotics and Automation*, 2:1398–1403, 2002.

[21] Dae-Hyung Park, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. *2008 8th IEEE-RAS International Conference on Humanoid Robots*, pages 469–474, 2008.

[22] Knox W.B., Stone P., and Breazeal C. Training a Robot via Human Feedback: A Case Study. In Herrmann G., Pearson M.J., Lenz A., Bremner P., Spiers A., and Leonards U., editors, *Social Robotics*, pages 460–470. Springer, 2013.

[23] Guangliang Li, Randy Gomez, Keisuke Nakamura, and Bo He. Human-Centered Reinforcement Learning: A Survey. *IEEE Transactions on Human-Machine Systems*, 49(4):337–349, 2019.

[24] W. Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The TAMER framework. *K-CAP'09 - Proceedings of the 5th International Conference on Knowledge Capture*, pages 9–16, 2009.

[25] Carlos Celemin and Javier Ruiz-Del-Solar. Interactive learning of continuous actions from corrective advice communicated by humans. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9513:16–27, 2015.

[26] James Macglashan, Mark K. Ho, Robert Loftin, Bei Bel Peng, Guan Wang, David L. Roberts, Matthew E. Taylor, and Michael L. Littman. Interactive learning from policy-dependent human feedback. *34th International Conference on Machine Learning, ICML 2017*, 5:3557–3566, 2017.

[27] Robert Loftin, Bei Peng, James MacGlashan, Michael L. Littman, Matthew E. Taylor, Jeff Huang, and David L. Roberts. Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning. *Autonomous Agents and Multi-Agent Systems*, 30(1):30–59, 2016.

[28] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L. Isbell, and Andrea Thomaz. Policy shaping: Integrating human feedback with Reinforcement Learning. *Advances in Neural Information Processing Systems*, 2013.

[29] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research*, 15:627–635, 2011.

[30] Klas Kronander and Aude Billard. Learning Compliant Manipulation through Kinesthetic and Tactile Human-Robot Interaction. *IEEE Transactions on Haptics*, 7(3):367–380, 2014.

[31] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input. *Advanced Robotics*, 25(5):581–603, 2011.

[32] Andrej Gams, Tadej Petrič, Bojan Nemec, and Aleš Ude. Learning and adaptation of periodic motion primitives based on force feedback and human coaching interaction. *IEEE-RAS International Conference on Humanoid Robots*, 2015-Febru:166–171, 2015.

[33] Andrea Bajcsy, Marcia K. O'Malley, Anca D. Dragan, and Dylan P. Losey. Learning Robot Objectives from Physical Human Interaction. (CoRL):1–10, 2017.

[34] Dylan P. Losey and Marcia K. O'Malley. Including uncertainty when learning from human corrections. *arXiv*, (CoRL), 2018.

[35] Mattia Racca, Joni Pajarinen, Alberto Montebelli, and Ville Kyrki. Learning in-contact control strategies from demonstration. *IEEE International Conference on Intelligent Robots and Systems*, 2016-Novem(October):688–695, 2016.

[36] Andrea Bajcsy, Dylan P. Losey, Marcia K. O'Malley, and Anca D. Dragan. Learning from Physical Human Corrections, One Feature at a Time. *ACM/IEEE International Conference on Human-Robot Interaction*, pages 141–149, 2018.

[37] Sylvain Calinon, Irene Sardellitti, and Darwin G. Caldwell. Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies. *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, (May 2014):249–254, 2010.

[38] Alex X. Lee, Henry Lu, Abhishek Gupta, Sergey Levine, and Pieter Abbeel. Learning force-based manipulation of deformable objects from multiple demonstrations. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June(June):177–184, 2015.

[39] K H E Kroemer. Horizontal Static Forces Exerted By Men Standing in Common Working Positions on Surfaces of Various T R Actions-Including Coefficients of Friction Between Various Floor and Shoe Materials. Technical report, AEROSPACE MEDICAL RESEARCH LABORATORY, 1971.

[40] Coppelia Robotics. https://www.coppeliarobotics.com/.

[41] Open Source Robotics Foundation. https://www.openrobotics.org/.

[42] MathWorks. https://nl.mathworks.com/solutions/robotics.html.

[43] Carlo Pinciroli. ARGoS, https://www.argos-sim.info/authors.php.

[44] Carlo Pinciroli. https://carlo.pinciroli.net/.

[45] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June(June):4397–4404, 2015.

[46] Serena Ivaldi, Vincent Padois, and Francesco Nori. Tools for dynamics simulation of robots: a survey based on user feedback. Technical report, 2014.

[47] Jan Fras B and Kaspar Althoefer. *Soft Pneumatic Prosthetic Hand Jan*, volume 8069. Springer, 2014.

Appendices

A | Analysis of DMP weight increments

To validate the way the positional trajectory is incremented, some small tests were done to analyse the behaviour of direct increments in the DMP weights on the final rollout trajectory. For this test different corrections were applied to two different trajectories, to see how the corrections affect the trajectory. The first trajectory is a relatively simple one, while the second trajectory is more complicated.

A.1 Simple Trajectory

To see what the effect is of directly updating the DMP weights, on the rollout trajectory, a simple trajectory was chosen, for which a single weight update was done, in the same way that it would be done in the final algorithm. The results of this experiment are shown in Figures A.1 and A.2. From these figures is seen that a simple weight update generates a similar reaction

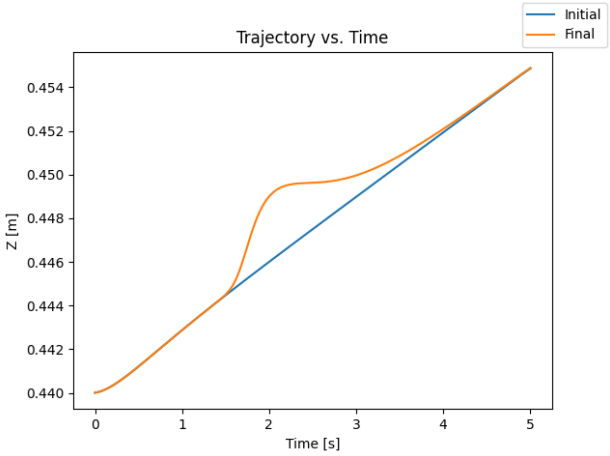


Figure A.1: Trajectory vs. time for a DMP, to show the difference that a weight update makes.

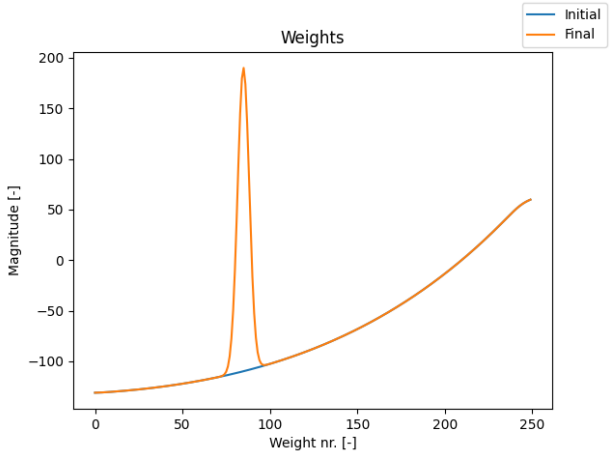


Figure A.2: DMP weights. This image shows the weight update that was applied.

on this trajectory. This shows that this might be a good way to adjust the rollout trajectory of a DMP system, though more experiments need to be done.

A.2 More realistic scenario

In the next scenario a trajectory was taken from one of the tasks, to see what the effect of these kinds of corrections is on a more complicated trajectory. The results of this experiment an be found in Figures A.3, A.4, and A.5.

From these figures, especially when looking at the difference plot in Figure A.5, is seen that the final trajectory still behaves exactly as anticipated, and that it should be possible to build a learning algorithm around this updating method.

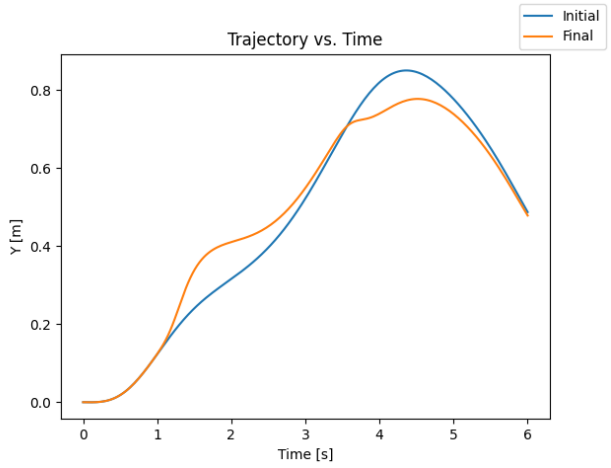


Figure A.3: Trajectory vs. time for a DMP, to show the difference that a weight update makes.

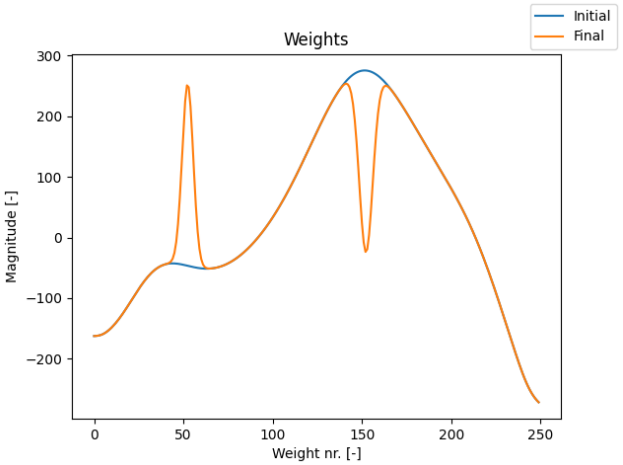


Figure A.4: DMP weights. This image shows the weight update that was applied.

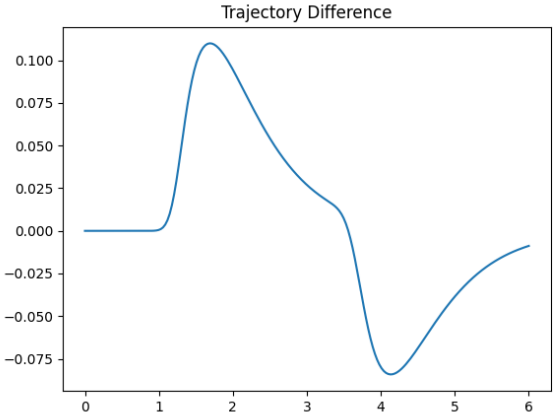


Figure A.5: Difference between the updated and the initial trajectory.

B | Controller Validation

For this appendix some tests were done that validate some of the assumptions that the algorithm makes about the controller. These tests look mainly into the assumption of repeatability, and the effect that corrections have on the repeatability. This shall be done for the Cartesian Impedance controller used in this research, to see if it is usable in the desired setting

B.1 Cartesian Controller

The tests done in this section are for the Cartesian impedance controller. This section tests the repeatability of movements, and investigate the way corrections are extracted.

B.1.1 Repeatability

In the algorithm, the corrections are extracting by executing a policy on the robot twice, once with corrections, and once without corrections, and subtracting the measured trajectories. To be able to do this, the algorithm makes the assumption that multiple attempts to execute the same policy lead to similar executed trajectories. To experimentally validate this assumption the robot was commanded to execute the same trajectory twenty times in a row, without corrections. The variance of these twenty trajectories, for all six Cartesian DOFs can be found in Figure B.1.

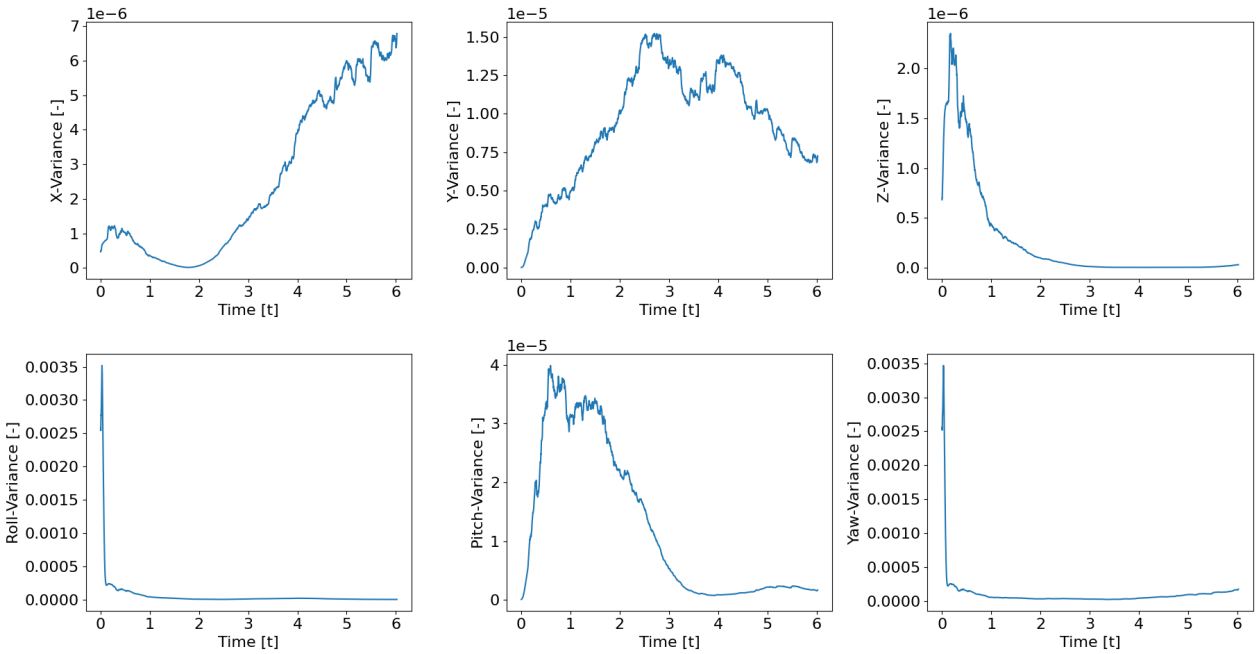


Figure B.1: Variance for 20 repetition of the same trajectories, for all six Cartesian DOFs

from this figure is seen that the variances in the trajectory are multiple orders of magnitude smaller than the trajectory, which has commended values of around 0.5. This gives us a rather large signal to noise ratio for extracting the corrections, especially for the translational DOFs. Note that the controller did not send a command for the rotational DOFs due to environment limitations.

B.1.2 Corrections

To validate the way corrections are made and extracted, a simple trajectory was executed, where a single correction was made in the X-axis. This correction consisted of an force of 500 N, applied on the end-effector, in the X-direction, during 0.05 seconds. The trajectory without correction was then subtracted from the trajectory with correction, and the result was plotted in Figure B.2.

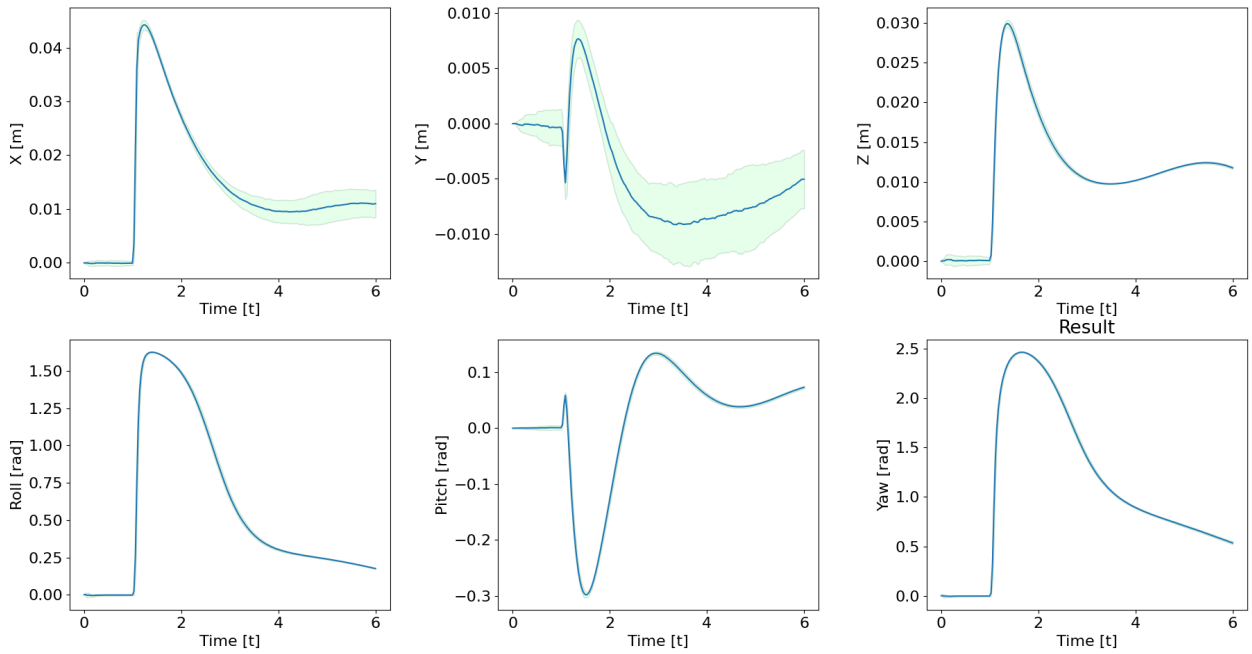


Figure B.2: Extracted corrections for all six Cartesian DOFs.

From this figure is seen that applying corrections is not as straight forward as one would expect, since a simple correction on the X-axis seems to have a lasting effect on all axes. This means that making oracles for these kinds of problems is going to be quite problematic, but that is not an issue specific this algorithm, since it assumes that the corrected trajectories are generated by humans, instead of single impulse-forces.

B.2 Oracle

To further prove that this effect will not be a problem for the experiments done in this thesis, a small validation study was done for the PD-controller oracle that is planned to be used in these experiments. For this experiment, the controller was tasked with executing the same trajectory, as in the previous experiments, but the oracle was instructed to consistently correct towards the point [0.6;0;0.45]. For this test, the reference trajectory, the non-corrected trajectory, and the corrected trajectory are shown in Figure B.3.

From this figure is seen that the corrected trajectories are all consistently closer to the "target point" of [0.6;0;0.45], than the uncorrected trajectories are. This leads to the conclusions that the previously discovered problem does not affect the PD-controlled oracle.

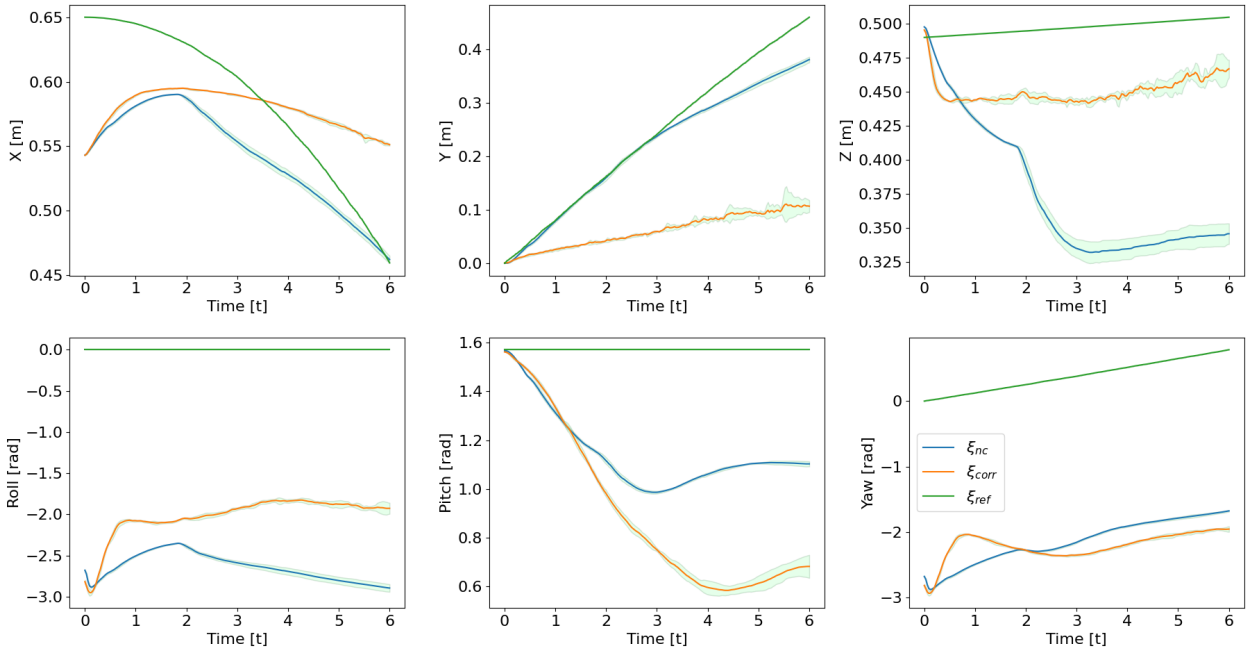


Figure B.3: Test of the oracle system, comparison of reference trajectory, and trajectories with and without corrections.

C | Interaction Forces

To further study the effects of the stiffness on the robot behaviour, and to ensure the safety of the system, a study was done into the magnitude of the interaction forces in the teaching process.

The goal of this experiment is to obtain a maximum for the stiffness value, for which a human can still apply corrections. The maximal forward force that a human can apply while free standing, to an object at shoulder height, while standing on a floor with a friction coëfficient of 0.6, is around 297 N [39]. This value is treated as an absolute maximum for the correction force. Ideally, the interaction force should not exceed half of this value, for safety reasons.

To find the maximum stiffness, the robot was set to a single-point trajectory, in a neutral position, that is comparable the positions it will be in during task execution. In this position a force was applied, and the resulting deviation was measured. This was done for multiple different stiffnesses, and two different poses.

C.1 Test Setup

Images of the two chosen positions can be found in Figures C.1 and C.2. These poses were chosen, since they represent the poses that the Panda will take on during the different tasks. While in these poses an interaction force will be applied to the end effector in the Z-direction,

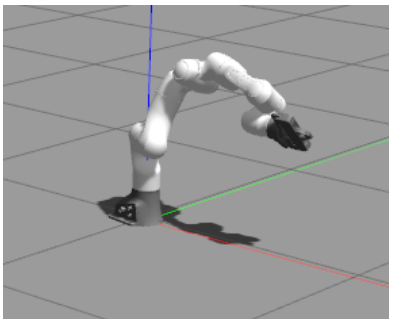


Figure C.1: First chosen pose

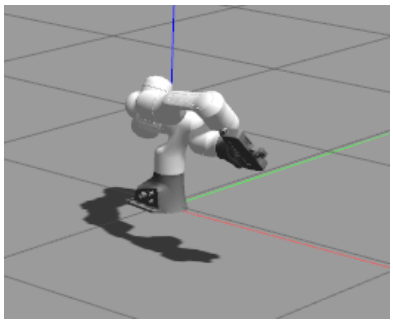


Figure C.2: Second chosen pose

for three seconds, while the maximal deviation is measured. This experiment is then repeated with different forces and different stiffnesses. The entire experiment is then repeated five times for statistical stability, even though the entire experiment should be fully deterministic.

For controller stability, the stiffness cannot exceed 1500, so the test range for the stiffnesses will be [50-2000], while the test range for the forces will be [1-500] N.

The results of these experiments will determine the maximum saturation value for the stiffnesses.

C.2 Results

The results of these test can be found in Figures C.1 and C.2. From these images is seen how the magnitude of a correction relates to the force and the stiffness. Note that the maximum displacement is around 0.4, which corresponds to the robot hitting the floor. This is the case for the aforementioned value of 297 N, for all stiffness values.

These plots show that the second configuration is less stiff than the first configuration, meaning that with, with the same stiffness, a larger displacement is created.

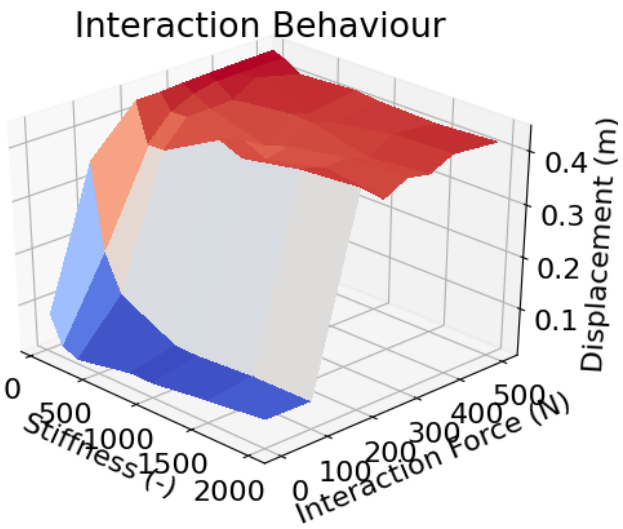


Figure C.3: First chosen pose

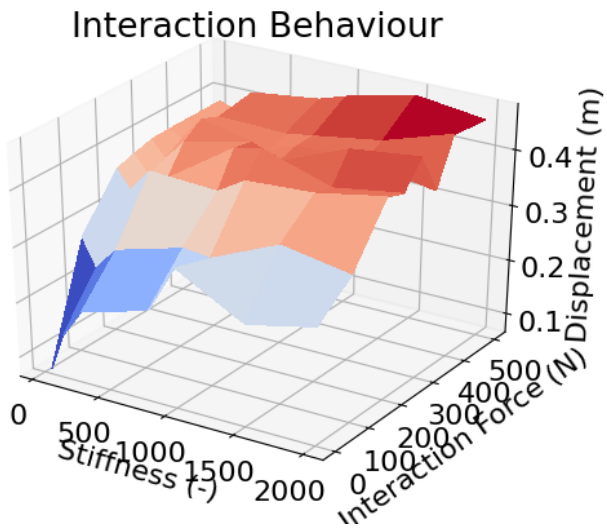


Figure C.4: Second chosen pose

For both situations however, it is clear that even the highest stiffness can be push against the ground, with a reasonable amount of force. This means that the full range of stiffnesses can be used.

C.3 Conclusions

This study shows that the full range of available stiffnesses from the controller [50-1500] can be used, without the interaction forces getting larger than a human can handle.

D | Rule Analysis

In this appendix some of the analysis images of HA-PASTIL are shown, that did not fit, or did not add enough value to the main text. These images illustrate the connection and the transition between the rules, and show that there are no discontinuities in the dependency of the weight updates on the correction.

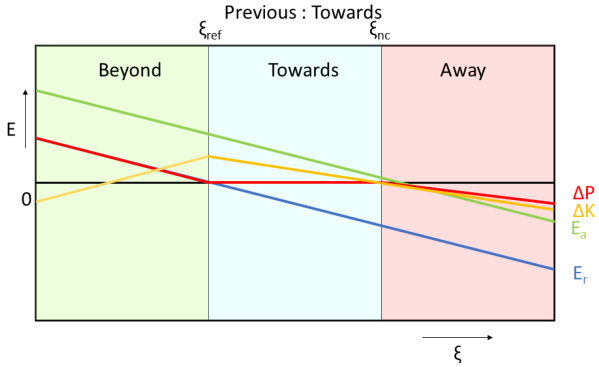


Figure D.1: Visual representation of the rules for when the last correction was towards

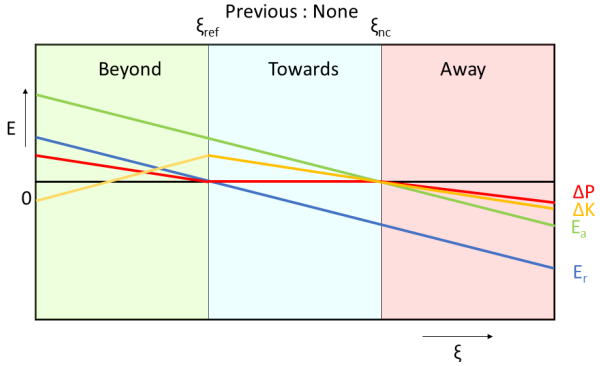


Figure D.2: Visual representation of the rules for when the last correction was None

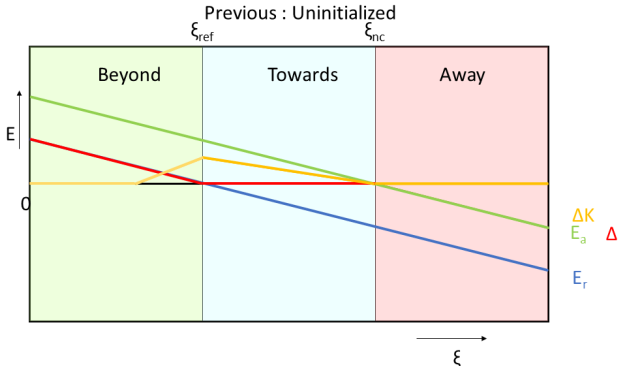


Figure D.3: Visual representation of the rules for when the last correction was Uninitialized

D.1 Towards

Figure D.1 shows the three rules, in the case that the last correction was towards. If the last correction was towards, and the user does not agree with the current policy, the expected correction is either towards again, or slightly beyond. If the new correction is in line with this expected correction, the stiffness is increased, otherwise the stiffness is decreased. If the new correction is beyond, the position will be updated with the normal learning rate, since it is expected that the reference might have to be moved that way, while if the new correction is away, the user is being inconsistent, and the position is only updated with half the normal learning rate, to find a bit of a compromise between the last two corrections.

D.2 None

Figure D.2 shows the three rules, in case the previous correction was to not give a correction. The rules for this are quite similar to the rules if the last correction was towards. If a previous correction was not given, the user either agreed with the previous performance, or has not much interest in this particular point of the trajectory. for this reason, the stiffness must be reduced, except in the case that the new correction is closer to the reference than the executed trajectory, in which case it should be increased. The position is updated with only half the learning rate, for both beyond and away, to use this a sort of finetuning mode.

D.3 Uninitialized

Figure D.3 shows the set of rules used in the very first update, these rules are for the case that the system has not been initialized yet. These rules increase the stiffness if the corrected trajectory is closer to the reference than the executed trajectory, and do not adjust it otherwise. The position is updated with the full learning rate, for both beyond and away, since the expectation is that the initial trajectory is suboptimal, and has to be adjusted quite a bit.

E | Environment choice

In this chapter a choice shall be made for the simulator environment. This shall mostly consist of comparing different simulators and different robot arm implementations.

E.1 Simulators

In this section a comparison shall be made between multiple physics simulators, for robotic applications.

E.1.1 CoppeliaSim

CoppeliaSim, previously known as V-REP, is a propriatary robotics simulator, developed by Coppelia Robotics[40]. While it is propriatary software, a free eduactional license is available, making CoppeliaSim an option to consider in this research. CoppeliaSim is a complex, heavy weight simulator, that is very feature packed. The most notable included features are the ability of the user to interact with the simulation during runtime, and mesh-manipulation.

E.1.2 Gazebo

Gazebo is an open-source robotics simulator that is developed by Open Robotics and is distributed as a part of the Robotics Operating System (ROS)[41]. The Gazebo simulator is a bit simpeler than CoppeliaSim, lacking some of its features. The major benefit of Gazebo is the large community and the existence of a lot of third-party implementations of robots with their controllers. This also means that there is a large forum, where questions can be asked, and can be answered. Another major benefit is that, since it is part of the ROS ecosystem, usage of ROS is deeply supported.

E.1.3 MuJoCo

MuJoco is a propriatary physics engine, developed by Roboti LLC, that, just like CoppeliaSim, offers a free educational license, allowing it to be used in this research. MuJoCo is mostly a physics engine, but also comes with a simulator, or with Unity integration, which has a lot of features.

E.1.4 MATLAB/Simulink

MATLAB is a proprietary tool suite developed by MathWorks[42]. Among lots of different tools, used in different fields, it also has a built-in robotics simulation environment. While MATLAB does not offer free educational licenses, the University offers MATLAB licenses to its students, making it an option to consider for this research.

E.1.5 ARGoS

ARGoS is an open-source robotics simulator, developed by Carlo Pinciroli, for the Swarmanoid project[43]. ARGoS claims to be the fastest general-purpose robot simulator in the literature[44]. ARGoS is mostly aimed at use for swarm robotics, for that purpose it trades complexity for performance. It does currently not support the importation of 3D meshes into the simulation, making it hard to simulate exitisting robots.

E.2 Conclusions

Some studies were found that compare these simulators[45] [46] [47], but none of them were recent enough to fully base the conclusion on.

The chosen environment is Gazebo with the package from `https://github.com/justagist/panda_simulator`, since it is the most complete Panda simulation, and it provides the Jacobian and the coriolis compensation from libfranka. For this environment it seemed to be the most doable to code a custom impedance controller, without too much delay for the thesis project.

F | Description of the Chosen Environment

In this section the chosen environment shall be outlined. This section shall be a short summary of the documentation of the used software packages. These include the used programming language, the used libraries, ROS, the used ROS packages and Moveit!.

F.1 Programming Languages

The programmming languages used in this research shall be Python 3 and C++, since these are the ones compatible with ROS and the ones that the TU Delft has courses on. The focus shall be on using Python, since it is the quickest to build these kind of research projects in, but for an eventual industrial application C++ might be a better choice, due to its superior performance.

F.1.1 Libraries

To limit the scope of this research, some library functions shall be used. These shall be outlined in this section

Basic Libraries

Some basic libraries are used in this research, such as numpy, scipy, math, time, sys, matplotlib and csv. Since these are used in almost every project, it does not seem necessary to write an analysis of these libraries.

dtw-python

This library is used to perform dynamic time warping on the collected trajectories. This is required for demonstrations, since some demonstrations might be performed faster than others, and the start and the end of the recordings might not line up perfectly.

pydmps

This library is used for all the DMP representations in this research. This library was chosen over other libraries, due to the ease of install, clear documentation and the understandable API.

F.2 ROS

ROS is an opensource robot middleware, developed by Open Robotics, that allows communication between multiple programs on multiple different devices, created in multiple different languages. Tme main concempt behind ROS is that these programs, called 'nodes' publish their information onto 'topics' where other nodes can find this information and process it. In this research ROS shall be used as the main way of communicating with the simulation, this choice was made, because ROS can use almost the same code to control an actual robot arm, as it does for controlling the simulation.

F.2.1 panda_simulator

This ROS package provides the Robot description and the Gazebo world file, for the Gazebo simulation, as well as ROS integration, Moveit! integration, and position, velocity and effort controllers for the Franka Emika Panda Robot. It also supplies most of the elements needed to build more complex controllers, such as the end-efector Jacobian, the Joint Mass Matrix, a gravity compensation vector and a coriolis compensation vector.

Topics

In this section the most important topics of the panda_simulator package and their possible usage shall be discussed. There are a lot more topics provided by this package, but these are the most relevant ones for this research.
/panda_simulator/motion_controller/arm/joint_commands
This is the topic that the controller commands have to be published to. This topic allows three different control modes, position control, velocity control and effort control. It also allow the specification of desired positions, velocities, accelerations and control efforts. The effort control mode can possibly be used to build more complex controllers onto.
/panda_simulator/custom_franka_state_controller/joint_states
On this topic the current Joint States are published as sensor_msgs/JointState messages. In this message type the joint positions, velocities, accelerations and control efforts are listed, along with the joint names. These messages can be used to record the joint states for kinaesthetic teaching.
/panda_simulator/custom_franka_state_controller/robot_state
On this topic information about the current robot state is published, these messages contain a lot of information, but most importantly the external forces on the end-effector, the gravity compensation, the coriolis compensation and the Jacobian and mass matrices. This information is can mostly be used to build controllers.

Services

The most important service provided by the panda_simulator package is the dynamic reconfigure for the PID gains for all controllers for all joints. This is spread out over a lot of different services, but can also be adjusted with rqt_reconfigure. This can be used to improve the tracking and decrease the overshoot of the controllers.

F.3 Gazebo

The Gazebo simulator itself, as described in the previous chapter, also has some of its functionalities available through the use of topics and services, the ones used in this research shall be outlined in this section.

F.3.1 Topics

The most important topic provided by the gazebo simulator is the /gazebo/link_states topic. This topic provides the current Cartesian location and orientation of all links of the robot. This should be mostly used for plotting and to check results, since the information will not be available for the control of an actual robot arm.

F.3.2 Services

The gazebo simulator provides some services that could be of use for the purposes of this research, the most important ones are listed below:

/gazebo/spawn_sdf_model
This service can be used to spawn objects in the Gazebo environment. These objects have to be specified in the SDF format, which allows the specification of geometry and the most important physical properties, such as mass, inertia and surface friction coefficients. While the SDF format has some great properties, it seemed to be lacking is some other areas, such as parameterization. For this reason it would be great to use the xacro package, which is only compatible with the URDF format. For this reason a custom xacro to SDF parser was built, using the sed command.
/gazebo/delete_model
This service can be used to delete models from the Gazebo environment. This could be used to reset the environment after manipulating an object.
/gazebo/apply_body_wrench
This service can be used to apply a Cartesian wrench (6-vector of forces and torques) to a robot link, for a certain duration. This can be used as a proxy for kinaesthetic teaching/correcting, since physical interaction with a simulated robot is not possible.

F.4 Moveit!

Moveit! is a software package that provides functionality for, for instance, kinematics, motion/path planning, collision checking, 3D perception and robot interaction. While most of its features shall not be used in this research, the ability to control the robot in Cartesian space is great for supplying initial task demonstrations to a learning algorithm.

F.4.1 Topics

The most important topic that Moveit! publishes to is the /move_group/status topic. This topic publishes some simple status messages of the Moveit! application, such as whether it is idle or moving the robot. This information is mostly used to synchronize timings for recording movements.

F.4.2 Services

Moveit! also supplies two important services, the first is /compute_fk, which can be used to transform Joint-space coordinates to Task-space coordinates in real-time. This can be used to replace the info of the /gazebo/link_states topic for use in Cartesian controllers. The other service supplied be Gazebo is /compute_ik, which translates Task-space coordinates to Joint-space coordinates. This might, for instace, be used to relate Task-space corrections to a Joint-Space Controller.

F.4.3 API Functions

Besides the aforementioned Topics and Services, Moveit! also has an API that allows communication with the robot as well as its planning environment. While this API has a lot of functions, the most important ones are MoveGroupCommander.compute_cartesian_path(), MoveGroupCommander.execute(), MoveGroupCommander.set_pose_target(), MoveGroupCommander.go() and PlanningSceneInterface.add_box(). MoveGroupCommander.compute_cartesian_path() is used to generate a Joint-Space trajectory from a few Cartesian waypoints and MoveGroupCommander.execute() is used to make the robot execute this trajectory. PlanningSceneInterface.add_box() is used to add certain regions to the environment that the no part of the robot is allowed to move into. These functions can be used in conjunction to do teaching by remote control. MoveGroupCommander.set_pose_target() plans a path to a Cartesian Pose goal and MoveGroupCommander.go() is used to execute this plan. This combination can be used to set the end-effector to an initial Pose, before starting any teaching.

G | Correction Interfaces

In this appendix an overview can be found of the correction interfaces that were designed during the process of this thesis, as well as their advantages and disadvantages. All these interface were abandoned in the end, since an oracle was implemented instead. During the thesis, a lot of time was invested in the creation of these interfaces, and they might be useful for further research.

G.1 Raw forces

The first correction interface was very simple. It allowed the user to directly apply pre-specified forces to the end-effector, in all 6 Cartesian DOFs, using the keyboard. There was no visual feedback, other than the robot moving in the desired direction. This interface had as downside that the forces were pre-specified, and that made it very hard to dynamically correct the robot in a meaningful way.

G.2 Movable boxes

For this interface a box was placed in the gazebo environment, that could be moved in XYZ-space by a user, with the keyboard. One key on the keyboard could then attach a virtual spring/damper system between the box and the end-effector, on which the box also would change colour, to indicate that it is in the 'active' state. This had as advantage that it was easier to correct to a single point, but it had as disadvantage that the box could not be moved fast/accurate enough to give the desired corrections. A visual interpretation of this interface can be found in Tables G.1 and G.2.

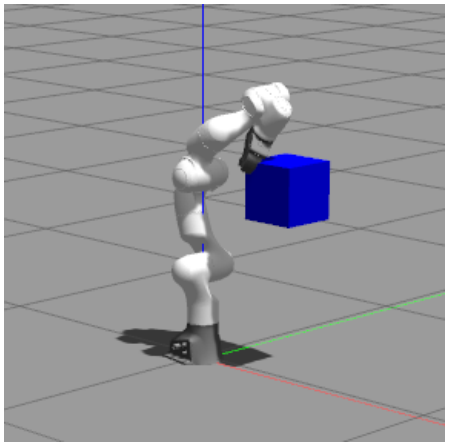


Figure G.1: Correction interface, the blue color indicates that the box is actively pulling the end-effector

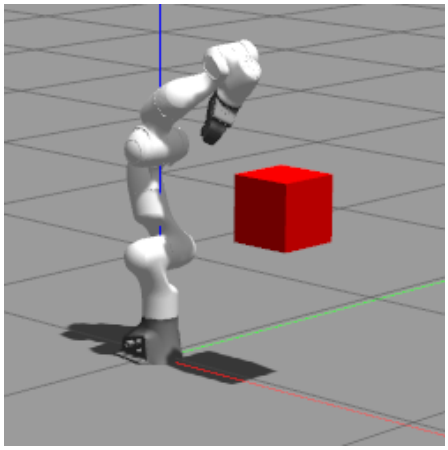


Figure G.2: Correction interface, the red color indicates that the box is not pulling the end-effector

G.3 Movable arrows

The last implemented human correction interface added floating arrows to the gazebo environment, that would float around the end-effector, as seen in Figure G.3. When one of these arrows would be activated, by pressing and holding a key on the keyboard, this arrow would change colour, and start floating away from the end effector, while exerting a force proportional

to its distance, meaning that the longer the key was pressed, the larger the exerted force. The arrows would also turn gray and transparent at times when giving a correction was not appropriate. In this state they would also be unresponsive to key presses. While this environment was supposed to combine the best of both previous interfaces, with a nice visual representation, it turned out that, even with this interface, it was difficult to apply high-quality corrections. This was mostly due to the fact that it still was not possible to apply quick corrections to a desired trajectory, in the way that one would do when correcting a physical robot.

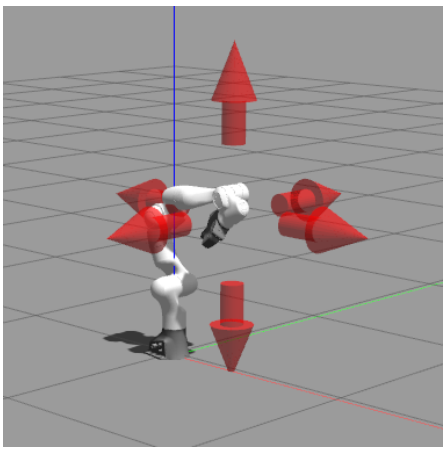


Figure G.3: Image of the movable arrows correction interface

G.4 Oracle

In the end it seemed best to use an oracle. Both, for the repeatability, and since a user study would not be possible in current times. The oracles were set up to perform corrections using a PD-controller, but only in the three translational DOFs. This decision was made, since the rotational DOFs do not add much for the designed tasks.

G.5 Stiffness visualisation interface

To make the correction process even more intuitive, an interface was developed to communicate the stiffnesses to the user, to help the user estimate the interaction forces and the expected robot behaviour. This interface was developed in a phase of the research where the entire system worked in joint-space instead of Cartesian space, due to this, an interface could be designed where a slightly translucent sphere was placed on each of the robot joints. These spheres would be green if the corresponding joint was fully compliant, and would gradually turn to red, as the stiffness increased. An image of this interface can be found in Figures G.4 and G.5. Since the Cartesian system was designed to be used with an Oracle, and did not have to communicate stiffnesses to an and user, no new interface was designed for stiffness communication. From these images is seen how the visualisation interface communicates the stiffnesses to the user. Figure G.5 shows the range of different colors between compliant and stiff.

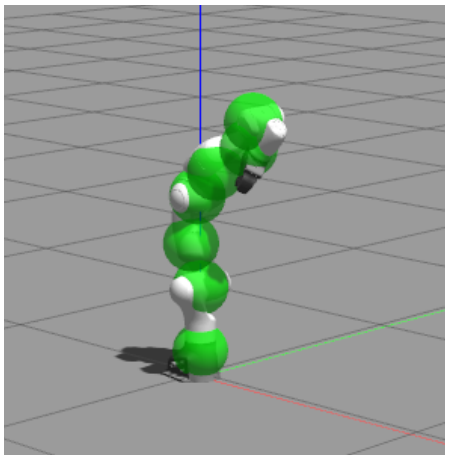


Figure G.4: Impedance visualization interface, all stiffnesses are set to 0

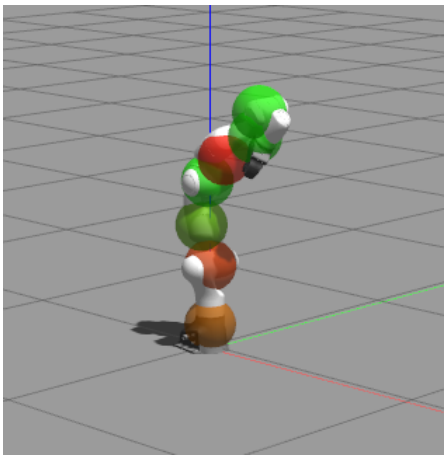


Figure G.5: Impedance visualization interface, all stiffnesses are set to random values

H | Cost Breakdown

In this appendix the breakdown of the cost function for the baseline algorithms is shown. This appendix also shortly discusses some of the effects found in these graphs.

H.1 Push Task

Figures H.1, H.2 and H.3 show the breakdowns of the cost function for the baseline algorithms for the push tasks. From these figures is seen that both the positional algorithm and PASTIL improve upon the score, while the mean-var algorithm increases the stiffness too much, to be able to improve the score. An intensive study was done into improving the behaviour of the mean var algorithm, to see if different parameter settings would lead to more optimal behaviour, but this did not seem to be the case.

Figure H.1: Breakdown of the cost for PASTIL

Figure H.2: Breakdown of the cost for the mean var algorithm

Figure H.3: Breakdown of the cost for the position-only algorithm

H.2 Lift Task

A breakdown of the scores obtained by the baseline algorithms can be found in figures H.4, H.5, and H.6. These figures show in more detail how the cost is built up. This information can be used to analyse some of the weak points of the algorithms.

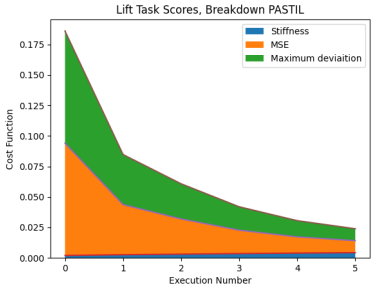


Figure H.4: Breakdown of the cost for PASTIL.

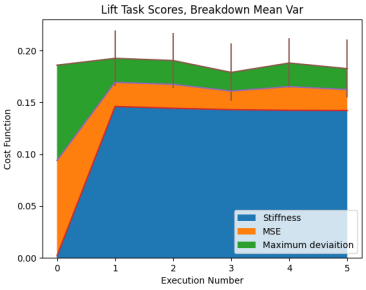


Figure H.5: Breakdown of the cost for the mean var algorithm.

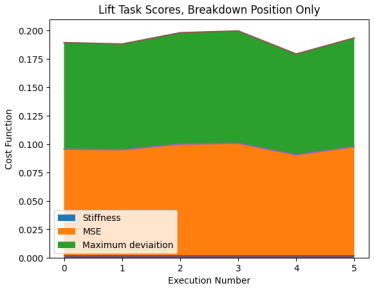


Figure H.6: Breakdown of the cost for the position-only algorithm.

From the cost breakdown for PASTIL is seen, that is closely resembles the breakdown of HA-PASTIL. This is to be expected, since both algorithms were designed for a similar purpose, and with a similar mindset. The breakdown for the mean-var algorithm shows that, while the task performance seems to improve, the cost does not decrease, do to the high stiffnesses being

learned by the algorithm. A cost comparison for all algorithms, without accounting for the stiffness can be found in Figure H.7. This comparison shows that without accounting for the stiffnesses, the mean-var algorithm has a decreasing cost, but it still is unable to compete with PASTIL and HA-PASTIL. Finally, the position-only algorithm does not seem to learn anything at all for this task. This hypothesis is backed up be Figure H.8. From this figure is seen that the "Learned" trajectory is the same as the initial trajectory, as specified in Section 4.4.5. Many different parameter settings were tried to improve the behaviour of this algorithm, but a better setting was not found. It might be beneficial for future research to try to find better baseline algorithms.

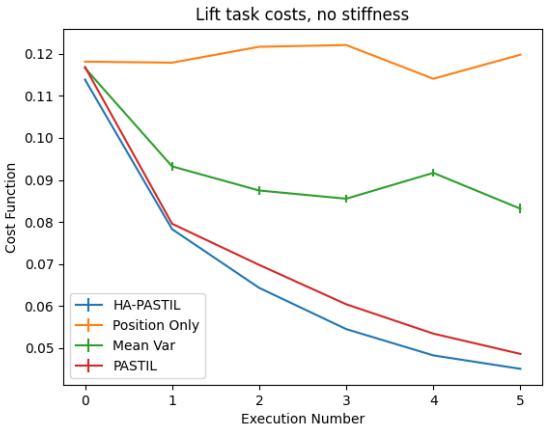


Figure H.7: Comparison of all algorithms, without accounting for the stiffness.

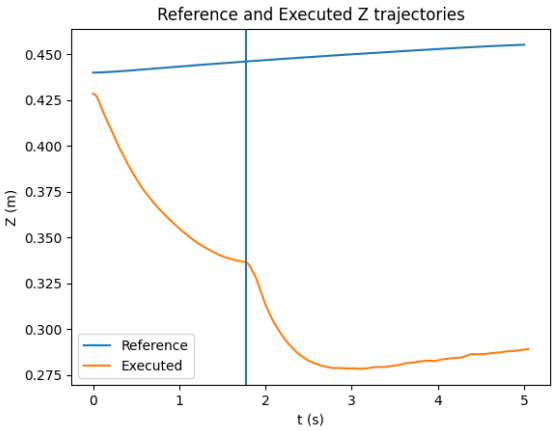


Figure H.8: Learned vs. Executed trajectory for the position-only algorithm.

H.3 Policy Reuse Push Task

Figures H.9, H.10, and H.11 show the breakdown of the cost functions for the baseline algorithms for the first task of policy reuse push task. From these images is seen that both PASTIL, and the positional algorithm show a behaviour that is consistent with that of HA-PASTIL, in this first task. The mean-var algorithm, on the the other hand, takes a lot longer unlearn the initial behaviour. Also note the relatively high costs for the stiffness for both PASTIL and the mean-var algorithm, compared to HA-PASTIL, which learns to be a lot more compliant.

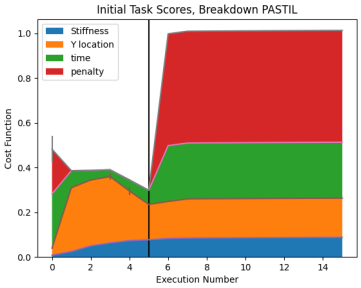


Figure H.9: Breakdown of the cost for PASTIL. The black line represents the change in correction strategy.

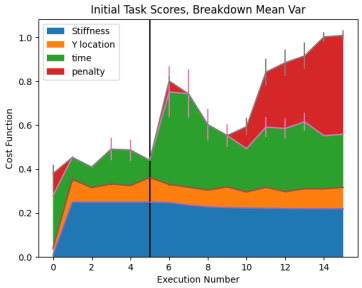


Figure H.10: Breakdown of the cost for the mean var algorithm. The black line represents the change in correction strategy.

Figure H.11: Breakdown of the cost for the position-only algorithm. The black line represents the change in correction strategy.

Figures H.12, H.13, and H.14 show the cost breakdown for the baseline algorithms, for

the final task of the policy reuse push task. From these images is seen that, from these three algorithms, only PASTIL learns to consistently push the box off of the table. While the position only algorithm ends up with a score that is significantly lower than the initial score, the presence of the penalty in the end, indicates that there were still some cases in which the box was not pushed off of the table. From these figures is also seen that the mean-var algorithm does not learn to push the box off of the table at all. This effect can best be explained by looking at the scores obtained during the corrected episodes. these scores can be found in Figure H.15. These scores were obtained in episodes where the oracle was active, and these represent the actual applied corrections. As seen from this image, the oracle is not able to push the box off of the table, when the mean-var algorithm is executing its policy. This can partly be explained by the high stiffnesses. This could, however, not explain the entire effect, since, as Appendix C showed, the robot is, at maximum stiffness, still compliant enough to perform rather large corrections. The effect might also be explained by the slow learning of the mean-var algorithm. Since the mean is taken of the entire dataset of corrections, every new correction has a smaller influence on the total trajectory, to a point where it almost cannot learn anymore.

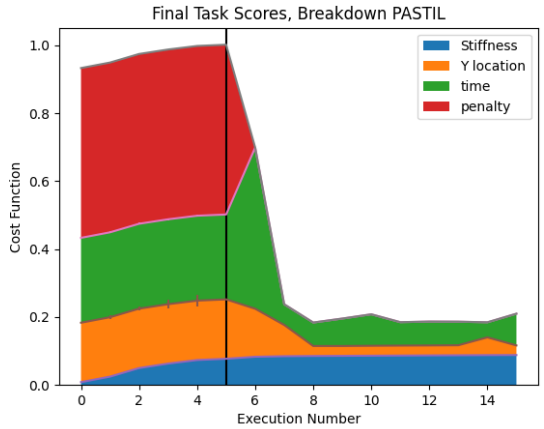


Figure H.12: Breakdown of the cost for PASTIL. The black line represents the change in correction strategy.

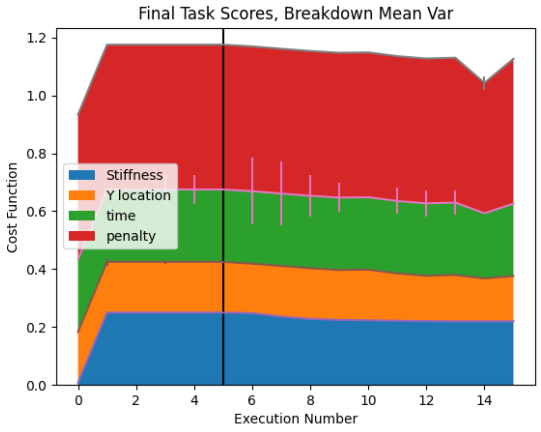


Figure H.13: Breakdown of the cost for the mean var algorithm. The black line represents the change in correction strategy.

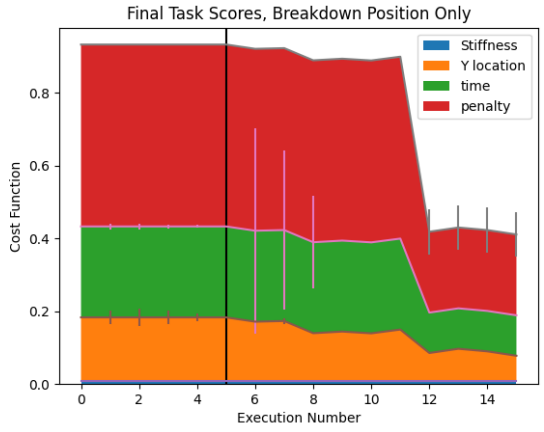


Figure H.14: Breakdown of the cost for the position-only algorithm. The black line represents the change in correction strategy.

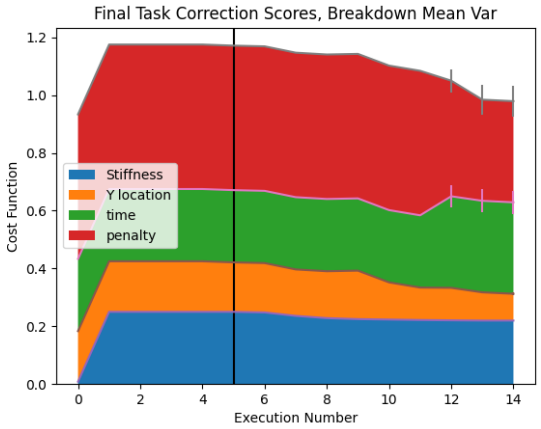


Figure H.15: Breakdown of the correction scores for the mean-var algorithm.