Joint Learning of Human and Robots

Gemeinsames Lernen von Menschen und Robotern
Bachelor-Thesis von Zlatko Nikolaev Kolev aus Sofia, Bulgarien
Tag der Einreichung:

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Marco Ewerton
3. Gutachten: Oleg Arenz, Dr. Joni Pajarinen

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Joint Learning of Human and Robots
Gemeinsames Lernen von Menschen und Robotern

Vorgelegte Bachelor-Thesis von Zlatko Nikolaev Kolev aus Sofia, Bulgarien

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Marco Ewerton
3. Gutachten: Oleg Arenz, Dr. Joni Pajarinen

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 30. August 2018

(Zlatko Nikolaev Kolev)

Abstract

Object disentangling and motion planning are interesting fields in robotics. They require e.g. a robot arm to be aware of obstacles in the way. Furthermore, the robot arm has to know how to grasp an object properly. Typically, it is hard for humans to provide proper demonstrations for untrivial disentangling tasks, when an imitation learning approach is chosen. Therefore, the use of reinforcement learning algorithms is required in order to infer an optimal execution trajectory. This bachelor thesis mainly focuses on implementing and testing the algorithm called *Relevance Weighted Policy Optimization* (RWPO) which is described in section 5 of the paper "Assisting Movement Training and Execution with Visual and Haptic Feedback" by Ewerton et al. [1]. RWPO is a reinforcement learning algorithm. In this thesis, different aspects of the algorithm are discussed and some of them are adjusted. Among the discussed details are the punishment of too long or too jerky trajectories and learning in up to 7 dimensions for multiple via points.
This work shows two application scenarios of RWPO. In the first scenario, RWPO is used to teach a robot arm how to disentangle objects in the robot simulation environment gazebo. The second experiment is similar to the piano movers problem. It is conducted in the simulation environment rviz and involves using the Haption Virtuose 6D device. In this experiment, a bar is teleoperated from a starting point to an end point in task space through a small window in a wall.

Acknowledgments

I would like to thank Marco Ewerton, Oleg Arenz, Joni Pajarinen, Jan Peters, Jeremy Tschirner, Melvin Laux and Michael Eiden.
Thanks to Marco for being my main supervisor. He is always friendly, helped me a lot during the thesis and showed a lot of patience. Thanks for staying in university at the weekend and guiding me through the experiments and writing.
Thanks to Oleg who granted me access to the grasping testbed project on GitLab and gave Marco the idea which led to using PCCs.
Thanks to Joni who showed me the real iiwas robot and the experiments, told me about Melvin and Jeremy and stayed with me in the lab at the weekend.
Thanks to Jan for his cool lectures Robot Learning and Statistical Machine Learning and for giving me the chance to write this cool thesis.
Thanks to Jeremy for helping me install the grasping testbed package and for sharing his useful ros nodes with me during the gazebo experiments.
Thanks to Melvin for allowing me to use his simulated s- and u-shape during my gazebo experiments.
Thanks to Michael for creating a picture which shows all the different positions of the bar during a Haption experiment.

Contents

Figures

List of Figures

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
CHOMP	Covariant Hamiltonian Optimization for Motion Planning
EM	Expectation Maximization
GMM	Gaussian Mixture Model
GMR	Gaussian Mixture Regression
GPR	Gaussian Process Regression
HSC	Haptic Shared Control
i.i.d.	Independent and Identically Distributed
LQR	Linear-Quadratic Regulator
LWL	Locally Weighted Learning
LWPLS	Locally Weighted Partial Least Squares
LWPR	Locally Weighted Projection Regression
LWR	Locally Weighted Regression
MLE	Maximum Likelihood Estimate
MP	Movement Primitive
NDT	Non-destructive testing
PCC	Pearson's Correlation Coefficient
PLS	Partial Least Squares
ProMP	Probabilistic Movement Primitive
ROS	Robot Operating System
ROV	Remotely Operated Vehicle

rviz	ROS Visualization
RWPO	Relevance Weighted Policy Optimization
RWR	Reward-weighted Regression
SSC	State Shared Control
STOMP	Stochastic Trajectory Optimization for Motion Planning
TP-GMM	Task-parameterized Gaussian Mixture Model

List of Symbols

Notation	**Description**
$\mathbf{c_i}$	center of the i-th basis function
$\boldsymbol{\epsilon}$	Gaussian noise
$\mathbf{h}$	width of a basis function
$\boldsymbol{\lambda}$	regression parameter
$\boldsymbol{\mu_w}$	mean of weight vectors w
$\boldsymbol{\Psi}$	basis function matrix
$\mathbf{R}$	reward for a given policy
$\mathbf{R_{\rho,o}}$	reward function for a parameter vector and a given objective
$\boldsymbol{\rho}$	parameter vector
$\boldsymbol{\sigma}$	standard deviation
$\boldsymbol{\Sigma_w^{f_o}}$	weighted covariance matrix
$\boldsymbol{\Sigma_w}$	variance of weight vectors w
$\boldsymbol{\tau}$	approximation of a trajectory
$\mathbf{w}$	weight vector
$\mathbf{z_t}$	phase variable

List of Operators

Notation	**Description**	**Operator**
b_i^G	Gaussian basis function	$b_i^G(\bullet)$
O	upper bound on the growth rate of a function	$O(\bullet)$
|	conditional distribution	$p(\bullet\|\bullet)$
exp	e^x	$\exp(\bullet)$

N	normal distribution	$N(\bullet)$
ψ_i	normalized Gaussian basis function	$\psi_i(\bullet)$
f_o	relevance function for objective o evaluated for policy parameter n	$f_o(\bullet)$

1 Introduction

1.1 Motivation

Teleoperation of a robot arm can be challenging for a human because the human cannot precisely estimate the positions of objects or find out what force to apply in a certain direction in order to reach a given goal. There are some methods to assist humans in teleoperation, which are stated in Section 1.2 Related Work. In the method stated by Ewerton et al. [1], the robot assists the human by using a probabilistic distribution of trajectories. This distribution is based on initial human demonstrations and optimized based on certain performance criteria.

The driving idea of this thesis is that humans and robots can work together in order to accomplish a given task. In an object disentangling scenario, a human can provide initial demonstrations. The robot will then optimize those in order to reach an object of interest, while avoiding collisions with other objects along the way. The human can then teleoperate the robot arm and let it guide him along the learnt trajectories. The human can himself apply a force at some point of the trajectory if the trajectory is not optimal at that given point. For example, the human can take over the grasping task. The movement can then be recorded, learnt and optimized by the robot.

The reinforcement learning method introduced by Ewerton et. al helps a robot learn a distribution over trajectories and optimize it based on given objectives. After that, a human can teleoperate the robot and let himself get guided along the trajectory distribution. This method is called RWPO.
As RWPO was implemented only for three via points (start, center and end), this paper aims to investigate whether RWPO can be used in a general setting, with an arbitrary number of via points. Furthermore, this paper investigates the applicability of the algorithm on object disentangling and teleoperation, using different experiments in simulated environments.

1.2 Related Work

This paper investigates the scalability and applicability of RWPO in the domains of teleoperation, motion planning and object disentangling. Therefore, this section gives an introduction into three related topics: Imitation learning, teleoperation and motion planning.
RWPO uses Reward-weighted Regression (RWR). This is a regression method in reinforcement learning which weights inputs according to their immediate reward [2]. This method is explained in the last part of this section.

1.2.1 Imitation Learning in Robotics

Imitation learning is used in robotics in order to learn how to execute a specific task based on demonstrations provided by a human expert. In [1], imitation learning is used to learn an initial probability distribution over possible trajectories. A ProMP is computed using demonstrations provided by humans. The following part shows two applications of imitation learning and also gives a short introduction to ProMPs. The computation of ProMPs is going to be explained in detail in the next section.

Superhuman Performance of Surgical Tasks by Robots Using Iterative Learning from Human-Guided Demonstrations

Van den Berg et al. [3] propose an approach with the aim to allow robotic surgical assistants to execute specific trajectories with superhuman performance in terms of speed and smoothness. They evaluate this approach on two different tasks using the *Berkeley Surgical Robots*: a figure eight trajectory and a two-handed knot-tie. The tie is a suturing sub-task which is a part of many surgical procedures. [3]
The approach works as follows. In the first step, a set of trajectories is recorded using human demonstrations. These are then analyzed to extract a smooth reference trajectory. In this step, a *Kalman smoother* is used to produce (Gaussian) distributions of the states along the reference trajectory. The Kalman smoother is used as part of an Expectation Maximization (EM) algorithm, which iteratively derives the distributions of the reference trajectory given the current model

parameters. The model parameters are then updated by maximizing their likelihood given the current distributions of the reference trajectory. In order to time-align the demonstration trajectories with the current reference trajectory, a time-warping step is also included in the EM-algorithm. [3]
The learnt reference trajectory is executed at gradually increasing speeds using *iterative learning control*. It works as follows: First, the initial reference trajectory is sped up and executed via a Linear-Quadratic Regulator (LQR) controller. After that, the trajectory is adjusted based on the observed error and then again executed by the LQR controller. The process is repeated until convergence. Then, the execution speed of the newest reference trajectory is slightly increased, and the process repeats. The execution speed is continuously increased as long as the quality of the execution remains above a pre-defined threshold. [3]
Experiments show that the approach stated in the paper enables fast learning of trajectories, smoother trajectories than the human-guided ones, and an execution speedup of the factor 7 to 10. [3]

RWPO also learns a distribution over trajectories. However, it is a reinforcement learning algorithm which can optimize the trajectories w.r.t. given objectives, like via points along the trajectory. Van den Berg et al. [3] focus on rather simple tasks which do not need any variation or optimization. Another difference is that the trajectories learnt by RWPO are not meant to get sped up since they are used to provide force feedback to a user.

Learning for Control from Multiple Demonstrations

Another application of imitation learning is teaching aerobatics to an autonomous helicopter. The approach proposed by Coates et al. extracts a trajectory by using a small set of sub-optimal demonstrations. These are used by the algorithm in order to learn a local model suitable for control along a trajectory for each aerobatics task. The work also has strong similarities with recent work on inverse reinforcement learning, which extracts a reward function rather than a trajectory from the expert demonstrations. [4]
Coates et al. [4] propose a generative model that describes the expert demonstrations as noisy observations of the intended target trajectory. Each demonstration can be warped along the time axis. An EM algorithm is used for the learning of the target trajectory. It includes an extended Kalman smoother and a dynamic programming algorithm to perform the E-step in order to derive the intended target trajectory and a time-alignment of all the demonstrations. Furthermore, the algorithm incorporates prior knowledge to improve the quality of the learned trajectory. In other words, the demonstration data is first time aligned and then used to learn a sequence of trajectory-specific models, while prior knowledge is incorporated in the process. [4]
The results of the work include the first autonomous tic-tocs, loops and hurricane, better performance on previously performed aerobatic maneuvers (such as in-place flips and rolls), and a complete airshow. The airshow itself requires autonomous transitions between these and various other maneuvers. [4]

In [4], expert demonstrations are used to learn the aerobatics task. The main idea is similar to the idea of [3]: to learn how to execute a task based on demonstrations. RWPO on the other hand aims at optimizing imperfect demonstrations using reinforcement learning. In [4], the demonstrations have to be of certain quality, because otherwise, the helicopter would not be able to fly at all. RWPO allows initial demonstrations which do not necessarily have a lot in common with the desired final trajectory.

Probabilistic Movement Primitives

Movement Primitives (MPs) are a widely spread approach in imitation learning. A ProMP is a statistic representation of an MP. The RWPO algorithm in [1] uses ProMPs in order to learn an initial trajectory distribution using the human demonstrations. ProMPs are explained in detail in section 2: Foundations.

MPs are an approach for representing modular and reusable trajectories for robot movement. They provide a compact representation of continuous and high dimensional robot movements. ProMPs are a general probabilistic framework for representing and learning MPs. A ProMP is a distribution over trajectories. Using concepts from statistics brings many advantages. For example, adaption of a movement to a new target can be realized by conditioning on the target's positions or velocities. Moreover, two elementary behaviors can be used in parallel. This can be accomplished by a product of two independent trajectory probability distributions. A trajectory distribution can also encode the variance of the movement. Therefore, a ProMP can often directly encode optimal behavior in stochastic systems. Finally, a probabilistic framework can be used to model the covariance between trajectories of different degrees of freedom. That allows the joints of a robot to be coupled. ProMPs solve a main problem of MPs, namely the difficulty of extracting a policy for controlling the robot from a trajectory distribution. ProMPs incorporate many advantages from well-known previous movement primitive representations, such as phase variables that enable temporal rescaling of movements, and the ability to represent both rhythmic and stroke based movements. However, the increased flexibility and advantageous properties of the representation require multiple demonstrations to learn the primitives, unlike past approaches, which can clone movements from a single demonstration. [5]

RWPO and Imitation learning

RWPO learns an initial ProMP from given demonstrations. Since these lie somewhere in the task or joint space, they provide an initial environment for further exploration by the algorithm. The ProMP learnt from demonstrations is constantly optimized w.r.t the current objectives (via points). In other words, the policy search in RWPO is initialized with a ProMP based on demonstrated trajectories. RWPO is a reinforcement learning algorithm which aims at optimizing this ProMP w.r.t. certain objectives.

1.2.2 Teleoperation

In [1], teleoperation is used in an experiment where the users have to teleoperate a robot in a simulated 3D environment along a given trajectory. The trajectory is first learnt using the RWPO algorithm. As this paper investigates the possibilities of RWPO, the following part shows related work on teleoperation.

A Survey on Teleoperation

Teleoperation means "doing work at a distance". "Distance" can refer to a physical distance, where the operator is separated from the robot by a large distance, but it can also refer to a change in scale. For example, a surgeon may use micromanipulator in order to operate on a microscopic level. In teleoperation, a human called master controls a remote robot (slave). The master is also called *operator* and the slave is called *teleoperator*. Traditionally, teleoperation has been used in applications where normal onboard manual operation or control cannot be used because it would be too hazardous, expensive or inconvenient. A few examples are the handling of nuclear materials, control of small models and space and underwater exploration. These tasks are rather dangerous, expensive or impossible (e.g. when handling microscopic models). [6]
When teleoperating, telepresence is needed to let the operator feel that he is present at the site of the teleoperator. This is helpful for improving the quality of the task execution. Typical ways to simulate telepresence are e.g. cameras that follow the operator's head movements, stereovision, sound feedback, force feedback and tactile sensing. [6]
Force feedback means that the force generated by the teleoperator, usually a manipulator, is fed back to the operator in order to generate a real response in gripping and manipulation tasks. In manipulation tasks, force feedback is essential for a good telepresence. It can also be used in virtual environments. [6]
Tactile feedback is based on tactile information. It refers to the sense of contact with the object, mediated by the responses of low-threshold mechanoreceptors innervating the skin within and around the contact region.
Both force and tactile feedback can be united under the term *haptic feedback*. In teleoperation, the main difference between a haptic interface and a force feedback interface is the touch point. In force feedback, the muscular (kinesthetic) sensors give the response to the operator. In the haptic feedback, the tactile skin sensors have the main role. In haptic interfaces, the tactile sensing of the robot manipulator is fed back to the fingers of the operator. [6]

Experiments in a 3D environment investigating the RWPO algorithm include providing haptic feedback to a user. When learning complex movements in a 3D environment, haptic feedback may give the human information about how to adapt his movements. This information can be difficult to extract only from visual feedback. In the experiment conducted by Ewerton et al., haptic feedback is given to the users based on a probability distribution over trajectories resulting from the RWPO algorithm. [1]

Learning Assistive Teleoperation Behaviors from Demonstration

The initial handling of an emergency situation often involves using Remotely Operated Vehicles (ROVs), since dynamic interaction with the environment is required. At the same time, ROV tasks are common to such scenarios and are often recurrent.
Havoutis and Calinon [7] use a probabilistic approach to learn a task behavior model from data. The learnt model is a task-parameterized Gaussian mixture model. Such a model can be used to assist an operator performing the same task in multiple missions over and over again. This approach can capture behaviors (constraints) that are present in the training

data. The model can be combined with the operator's input online. Havoutis and Calinon demonstrate their approach on a teleoperation mock-up using a two-armed Baxter robot. Their model can learn task-specific behaviors, which are combined online with the operator's input in an assistive teleoperation manner. This approach aims at reducing the time and effort required to perform teleoperation tasks that are common in ROV missions in the context of security, maintenance and rescue robotics. [7]
Non-destructive testing (NDT) of underwater infrastructure is typically a time-consuming task for both human divers and ROVs. It involves "scanning" a number of surfaces that are of interest while keeping the NDT probe at a certain angle against the surface and moving at a certain speed. The end-effector state of a teleoperated arm performing the NDT scanning task is usually constrained. The constraint is to keep the probe at a certain angle against the surface, while the position of the probe can take a range of values. The difference in the signals (position and orientation of the scanning tool for the NDT task) can be exploited and encoded in a behavior used to assist the teleoperator in missions that require NDT scanning. [7]
The authors propose a Learning from Demonstrations (LfD) approach. The reproduction of a reference movement or an average skill behavior can be formalized as a regression problem. This can be done by using a Task-parameterized Gaussian Mixture Model (TP-GMM) representation of the demonstrated samples. For reproducing the motion or behavior, Gaussian Mixture Regression (GMR) is used. In robotics, GMR offers a simple solution to handle encoding, recognition, prediction and reproduction in robot learning. GMR relies on basic properties of normal distributions such as linear transformation and Gaussian conditioning. It provides a probabilistic representation of movements or policies. The learnt model can compute the next actions online with a computation time that is independent of the number of data points used to train the system. [7]
In contrast to other regression methods such as Locally Weighted Regression (LWR), Locally Weighted Projection Regression (LWPR), or Gaussian Process Regression (GPR), GMR does not model the regression function directly, but models the joint probability density function of the data. It then derives the regression function from the joint density model. This can be an advantage in robot applications where the input and output components are only specified at the last step of the process, if input information is not always given, or if retrieving the whole set of outputs is not needed. Density estimation can be performed in an offline phase with the TP-GMM estimation step. This speeds up the online regression process. [7]
During operation, the learnt reference position is tracked using an infinite-horizon formulation of a Linear Quadratic Regulator (LQR). Within this setting, the estimated mean of the previous step can be used as a reference, and the covariance as a weighting of the control cost. This is already a formulation of a controller for the teleoperated robot and allows the robot to act according to the learned task behavior. [7]
The task behavior models can be learned from data of previous missions as such tasks are often recurrent. The approach of Havoutis and Calinon is general and can be used for various ROV tasks, e.g. drilling, using a screwdriver, pushing buttons, turning valves, cutting using a tool, etc. [7]

Havoutis and Calinon [7] learn recurrent behaviour based on human demonstrations. The learnt behaviour is then used to assist a human on a teleoperation task. The authors use Gaussian mixture models and GMR for the learning. On the other side, RWPO uses other probabilistic methods. These are ProMPs. Furthermore, RWPO is a reinforcement learning method which can optimize trajectories w.r.t. given objectives. Therefore, RWPO is more dynamic and a robot which has learnt through RWPO is more autonomous. It is able to guide a human through a task. In [7], the task is split between the human and the robot: The human takes the hard part and the robot takes the simple recurrent task. In RWPO, human and robot work together towards a solution for a hard task.

Programming by Demonstration for Shared Control With an Application in Teleoperation

Shared control strategies can improve task performance in teleoperation. In such systems, a human operator is guided or corrected by the robot. The amount of correction or guidance that is provided is called the level of automation. It is normally time-consuming to manually decide on the level of automation. Zeestraten et al. present an approach to program this automated system by demonstration. The approach determines the level of automation online, by combining the confidence of the robot and teleoperator. The authors provide implementations using haptic shared control and state shared control and evaluate these in a user study. [8]
Depending on the implementation, virtual fixtures can either be seen as a form of semi-autonomous control or shared control. In their work, Zeestraten et al. [8] distinguish the two by the way human and robot control a system. A system is defined *semi-autonomous* when the control of the state variables is separated. A task that requires the control of both position and orientation of the robot end-effector is performed semi-autonomously when the human controls the position manually, while the orientation is controlled by the assistive system. In *shared control*, state variables are jointly controlled by the human and assistive system. The weighting of the control inputs determines the level of automation. [8]

The control intentions of human and robot can be combined at different levels. The majority of virtual fixtures can be seen as a form of Haptic Shared Control (HSC). In HSC, the intentions are mixed at the operator interface through haptic interaction. The haptic communication between operator and assistive system makes the human operator fully aware of the intentions and output of the control system. Alternatively, the intentions of the operator and assistive system can be mixed after the interface, at the state level. In this case, states given by the operator and the assistive system are fused through a weighted combination. This form of shared control is called State Shared Control (SSC). SSC does not require physical interaction with the operator. [8]
The paper of Zeestraten [8] is similar to Havoutis et al. [7]. It advances in two directions: First, the approach relies on the *Riemannian framework* introduced in [9]. This allows the consideration of generic rotation in 3D-space. Second, approaches on learning both SSC and HSC strategies are presented, while Havoutis et al. [7] only consider SSC. [8]
The methods in the work of Zeestraten et al. [8] involve control and statistical encoding of end-effector poses: elements of a *Riemannian manifold*. For the Riemannian Gaussian, there already exist Maximum Likelihood Estimate (MLE), product and conditioning operations. The authors take advantage of these properties in order to estimate the parameters of the Gaussian from demonstration data (MLE), combine statistical information from different sources (product), and infer the system state (conditioning). The result of these operations is approximated by a Riemannian Gaussian. Furthermore, complex distributions can be estimated by using a mixture model. The parameters of this Gaussian Mixture Model (GMM) can be estimated using EM. Statistical inference of GMM is done efficiently using *GMR* (GMR). Zeestraten et al. use an LQR to generate haptic feedback on the end-effector orientation. It provides a way to control gains of a state feedback controller by defining a state and control cost matrix. [8]
Zeestraten et al. [8] present a system that relates the level of automation to the confidence of its inputs. This is achieved by taking into account the confidence level when combining the inputs of the shared control system. When both human and assistance are equally confident, their inputs are weighted equally. When one of the inputs has higher confidence, it gets weighted stronger. The assistive system can change at each time step. As a result, the level of automation is continuously re-evaluated. [8]
Zeestraten et al. [8] have tested their approach in a real-world scenario. The experimental task is part of the maintenance procedure of a collimator, a device used to parallelize particle beams in a particle accelerator like e.g. the *Large Hadron Collider* (LHC) at CERN in Switzerland. The LHC can contain up to 152 collimators, which are located in radioactive areas. The maintenance procedure involves the removal and replacement of a protection cover. Both HSC and SSC can be trained on demonstrations and used in reproduction, providing a varying level of autonomy. [8]
An interesting result is that the subjects teleoperation performance (task execution time) did not improve, although the subjects indicated they preferred the learned shared control strategies. This is opposite to previous work on shared control. The authors state a potential source that could have caused this contradiction. They explain that the task for evaluating the shared controllers was found difficult by the subjects. The procedure in the task is simplified by a proper orientation of the end-effector, as provided by the shared controllers. However, additional fine manipulation is required to remove or place the cap. The cap can slip within the hand, if it collides with the environment. This makes replacement of the cap more difficult, as the orientation desired by the model mismatches the one required for replacement. This issue can be overcome by modifying the environment, but such modifications make application less realistic. [8]

Zeestraten et al. [8] use EM and GMR for the learning task, while RWPO relies on ProMPs and RWR. Another difference is that in [8], different control strategies are shown. Ewerton et al. [1] only use force feedback. Their control strategy can be seen as HSC. A performance difference is that RWPO is able to optimize given trajectories, while the approach of Zeestraten et al. [8] can not optimize the trajectories dynamically, e.g. w.r.t. new objectives. That is also why the grasping task was found hard by the human users [8], who had problems with the orientation of the cap during experiments.

RWPO and Teleoperation

The RWPO algorithm learns a ProMP from demonstrations and optimizes it with respect to the given objectives. If the ProMP is learnt in joint space, it can be used to sample reference trajectories over joint angles based on a Gaussian probability distribution. In a setting where a user teleoperates a robot arm, a distance between the current position of the robot and the mean of the ProMP in the current position can be computed. This distance can be used to provide haptic feedback to the user, guiding him towards the mean of the trajectory distribution. The mean signalizes an optimal trajectory. A simplified experiment is described in detail in [1]. Regarding this, RWPO can be used for teleoperation by providing haptic feedback to improve the task performance of a human user.

1.2.3 Motion Planning

RWPO is a general algorithm for optimizing demonstrations w.r.t. different objectives. For instance, it can be interpreted as a motion planning algorithm, if the demonstrations are trajectories in joint or task space and they need to be optimized w.r.t. criteria like distance to obstacles or trajectory length and jerkiness. This subsection introduces two well-known motion planning algorithms: Covariant Hamiltonian Optimization for Motion Planning (CHOMP) and Stochastic Trajectory Optimization for Motion Planning (STOMP) and points out differences and common aspects regarding RWPO.

CHOMP: Gradient Optimization Techniques for Efficient Motion Planning

CHOMP is a method for continuous path refinement. It uses covariant gradient techniques to improve the quality of sampled trajectories. CHOMP can be used as a standalone motion planner in many real-world planning scenarios. Its effectiveness is demonstrated by Ratliff et al. in manipulation planning for a 6-DOF robotic arm and in trajectory generation for a quadruped robot. [10]
The authors use geometrical relations and ideas from differential geometry in order to solve three different tasks. First, they measure quantities of trajectories like total length and jerkiness. These are used to ensure smoothness of the trajectories. Second, measurements of obstacle costs are taken into account in order to avoid unwanted collisions of the robot with obstacles in the environment. Finally, the same geometrical considerations used to update trajectories are used when correcting occurring joint limit violations. [10]
The goal of Ratliff et al. is to find a smooth and collision-free trajectory through the configuration space between two given points. The authors model the cost of a trajectory using an *obstacle term* and a *prior term*. The obstacle term measures the cost of being near obstacles, while the prior term measures quantities such as smoothness and acceleration. [10]
The update rule is a special case of a more general rule known as *covariant gradient descent* [11]. This said, CHOMP is a policy search method which uses gradient descent methods. CHOMP considered covariant, because the update of a trajectory is a function only of the trajectory itself, and not the particular representation used. The CHOMP update rule is the solution of an optimization problem that maximizes the decrease in the objective function subject to making only a small change in the average acceleration of the resulting trajectory and not by making a small change in the parameters that define the trajectory in a particular representation. [10]
The algorithm solves a large breadth of planning problems, but it is prone to local minima for more difficult problems when time limitations exist. [10]

Both CHOMP and RWPO are policy optimization methods. However, RWPO does not use gradient descent. It optimizes a ProMP based on reward-weighted regression. Relevance coefficients are an important part of RWPO, as they help the algorithm sample efficiently. Both CHOMP and RWPO are able to optimize quantitative properties of trajectories like overall length and jerkiness. A difference between CHOMP and RWPO is that CHOMP works on the direct representation of trajectories in order to optimize them in such a way that collisions are avoided. On the other hand, RWPO specializes on so-called objectives. These are general and can for example be via points along a trajectory. These via points can be for example positions which pass near objects but do not collide with them. RWPO optimizes trajectories w.r.t. those via points. In other words, RWPO optimizes a ProMP, which is a statistical representation of a trajectory distribution, while CHOMP optimizes a single trajectory.

STOMP: Stochastic Trajectory Optimization for Motion Planning

STOMP is a motion planning approach using a stochastic trajectory optimization framework. It works as follows: First, noisy trajectories of a fixed duration are generated to explore the task or joint space. These are then combined to produce an updated trajectory. The initial trajectories are optimized based on a cost function which punishes collisions with obstacles and jerkiness of the trajectories. The stochastic nature of STOMP allows it to overcome local minima, which are a problem of gradient-based methods like CHOMP. [12]
STOMP introduces a stochastic representation of trajectories based on noisy parameter vectors. This allows using a derivative-free stochastic optimization method which is able to optimize arbitrary costs which are non-differentiable. The optimization is done by computing a stochastic gradient estimate. The update rule of STOMP is similar to an expectation-maximization algorithm. The mean of the trajectory distribution is updated to optimize the distribution of costs from the previous iteration. The performance of STOMP thus varies based on the initial trajectories. [12]
The STOMP approach was demonstrated both in simulation and real-world experiments with the Willow Garage PR2 mobile manipulation robot. Two types of scenarios were tested: *Unconstrained* and *Constrained*. The unconstrained scenario used a cost function which consisted only of the obstacle and smoothness costs. The constrained scenario introduced

a constraint cost term to keep the gripper upright at all times. Both scenarios showed better results than CHOMP, as STOMP does not have the problems of CHOMP regarding local minima. [12]

RWPO and STOMP have a lot of similarities: They use stochastic methods and avoid computing a gradient directly. However, RWPO goes further and formalizes the trajectory distribution as a ProMP. Moreover, RWPO does not depend on gradients at all, while STOMP computes a gradient estimate. Another similarity are the initial trajectories. These have to "explore" the task or joint space to ensure that the trajectory distribution can be optimized for every possible via point or objective in terms of task or joint space. STOMP also depends on noisy trajectories which explore the task or joint space. A difference between RWPO and STOMP is that STOMP optimizes the whole trajectory since the cost functions are applied globally. RWPO on the other hand optimizes the trajectories w.r.t. objectives, e.g. via points. This means that RWPO is potentially more flexible because a user can for example define own via points which he sees as important. The trajectories will then be optimized w.r.t. those via points. Another feature of RWPO are the relevance coefficients, which help the algorithm sample smartly and therefore reduce the computation time and effort.

1.2.4 Locally Weighted Learning and RWR

The approach presented by Ewerton et al. [1] uses RWR in order to find the correlation between the different policy parameters and objectives. A specific relevance function is used for each objective. The relevance functions are then used to optimize the overall policy with respect to each objective. The following part introduces related work on RWR and locally weighted learning.

Locally Weighted Learning

Locally Weighted Learning (LWL) is a class of *function approximation* techniques. It does predictions by using approximated local models around specific points of interest. LWL is an alternative to *global function approximation*. [13] Locally linear models are a statistical compromise among the possible local polynomials that can be fit to data. The key problem in LWL is to determine the region of validity of a local model. The region of validity of a local model is also called a *receptive field*, and can be computed from a Gaussian kernel. [14]
In order to learn local models, each data point receives a weighting factor which expresses the influence of the data point on the prediction. In general, data points which are close to the current query point receive a higher weight than those which are further away. LWL is also called *lazy learning* because the processing of the training data is shifted until a query point needs to be processed. This approach makes LWL an accurate function approximation method which allows adding new training points easily. [13]
There are two major categories of LWL algorithms. The first one consists of *memory-based LWL methods*. These keep all training data in memory. The second category consists of *purely incremental LWL methods* that do not need to remember any data explicitly. [13]
LWR is a classic approach to solving the function approximation problem locally. It is also called *Memory-based Learning* because all training data is kept in memory in order to calculate the prediction. The advantages of LWR are the high accuracy due to the local model and few open parameters. Extensions of LWR can include a ridge regression formulation to increase stability, an outlier removal technique for higher accuracy and the arrangement of the training data in k-d trees for lower computational costs. [13]
Schaal et al. show how the computational complexity of LWR can be reduced and numerical problems avoided by using Partial Least Squares (PLS). PLS is a dimensionality reduction method. The essence of PLS is to fit linear models through a hierarchy of univariate regressions along selected projections in input space. The projections are chosen according to the correlation of input and output data, and the algorithm assures that subsequent projections are orthogonal in input space. The authors derive a Locally Weighted Partial Least Squares (LWPLS) algorithm. [14]
With that, there are still two points of concern regarding LWR and LWPLS. If the learning system receives a large, possibly never-ending stream of input data, as typical in online robot learning, memory requirements to store all data as well as the computational cost become too large. Under these circumstances, a non-memory-based version of LWL is desirable such that each incoming data point is incrementally incorporated in the learning system and lookup speed becomes accelerated. Schaal et al. therefore propose incremental updates of the parameters of the linear models by using recursive least squares techniques and derive LWPR. [14]
LWPR is a purely incremental method. Instead of throwing away the model after each prediction like in LWR, LWPR keeps each model in memory for further predictions. It uses multiple locally weighted linear models which are combined for approximating non-linear functions. Adding new data points requires only an update to the existing models or the creation of a new model if there is no trustworthy model available. As the models are updated incrementally, the training data does not have to get stored in memory anymore. Furthermore, LWPR is using an online version of the PLS method

to handle redundant and irrelevant input data. The goal of PLS is to reduce the dimensionality locally to find optimal local projections and eliminate subspaces of the input space that minimally correlate with the output. [13]
LWPR is well suited for tasks on high-dimensional data, redundant input dimensions and continuous data streams. It combines high prediction accuracy, low computational costs and dimensionality reduction. Schaal et al. provide an implementation of the proposed algorithms on high dimensional learning problems and demonstrate their applicability in robot learning examples, including the learning of devil-sticking, pole-balancing of a humanoid robot arm, and inverse-dynamics learning for a seven DOF robot. [14]

Using Reward-weighted Regression for Reinforcement Learning of Task Space Control

Many robot control problems of practical importance, including task or operational space control, can be reformulated as *immediate reward reinforcement learning problems*. However, these algorithms are either slow, do not scale for higher-dimensional problems, or require executing random search, which can be dangerous for a physical system. Therefore, they are not feasible for learning control online. [2]
Peters et al. [2] reduce the problem of learning with immediate rewards to a RWR problem and obtain an algorithm which is suited for high degree-of-freedom robots. They use linear regression techniques and methods employing EM-style algorithms and apply a transformation on the rewards for faster convergence. [2]
The authors use this approach in order to learn multiple local controllers. These are then linearised piecewise by using a linear forward predictor model, which was learnt using LWPR. The model is then used to build a globally consistent solution to an operational space control task. [2]

LWL, RWR and RWPO

RWPO optimizes an initial ProMP w.r.t. the desired objectives. [1] These can be via points along a trajectory in order to accomplish an object disentangling task. If the ProMP is learnt in joint space, it can be optimized in such a way that the resulting probability distribution over joint trajectories respects task- and robot-specific constraints like collisions with objects or self-collisions. Given a learnt relevance function, the algorithm can optimize a policy w.r.t. to this relevance function. By assuming a Gaussian distribution over policy parameters, the algorithm can weight the original variances according to the relevance function, sample from that distribution and compute the rewards associated with the current parameters. This reward can then be optimized w.r.t. the mean and variance of the parameter distribution using RWR. In other words, RWPO can be used to learn an optimal ProMP over joint trajectories. By sampling from that ProMP, the joint angles along the sampled trajectory can be used as input for an n-DOF robot arm. The robot arm can then execute trajectories in order to accomplish a specific task, e.g. object disentangling.
RWPO learns which policy parameters are relevant to optimizing a trajectory w.r.t. a given objective. This idea resembles locally weighted learning, as the algorithm has to find out the relevances for each objective separately. Furthermore, RWPO uses RWR in order to optimize the ProMP. However, RWPO assumes a Gaussian distribution over the policy parameters. Therefore, it does not need an EM implementation for mixture models in order to estimate the parameters of the probability distribution. These are rather initialized based on the demonstrations. In [1], the distribution over the relevance parameters is also assumed Gaussian and gets optimized by using RWR in order to estimate the relevance functions.

1.3 Remainder of the Paper

The remainder of this paper is structured as follows: First, the theory behind the RWPO algorithm is described in detail, followed by an explanation and justification of the implementation details used in the experiments. After that, the gazebo and rviz experiments and their results are shown and evaluated. In the end, a conclusion is drawn regarding the scalability and applicability of RWPO in the domains of object disentangling, motion planning and teleoperation.

2 Foundations

2.1 ProMPs

A ProMP is a probabilistic representation for a set of trajectories. A ProMP consists of a mean μ_w and variance Σ_w of weight vectors. These weight vectors are computed by Ridge Regression based on the given trajectories from demonstrations. [15]
For stroke-based movements, Paraschos et al. propose Gaussian basis functions $b_i^G(z_t) = exp(-(z_t - c_i)^2(2h)^{-1})$, such that each of those is activated for given points on the current trajectory when multiplied by a corresponding weight. The parameter h is the width of the basis and c_i the center for the i-th basis function. The phase variable z_t decouples the trajectory from the time signal. The basis functions are normalized, s.t.

$$\psi_i(z_t) = \frac{b_i(z_t)}{\sum_{j=1}^{N} b_j(z_t)}. \tag{2.1}$$

This is done in order to obtain a constant summed activation and improve the regression's performance. The sum of all weighted basis functions represents a given trajectory, i.e. a trajectory can be approximated by

$$\tau = \Psi w + \epsilon, \tag{2.2}$$

where ϵ is a zero-mean i.i.d. Gaussian noise and

$$\Psi = \begin{pmatrix} \psi_1(1) & \psi_2(1) & ... & \psi_N(1) \\ \psi_1(2) & \psi_2(2) & ... & \psi_N(2) \\ ... & ... & ... & ... \\ \psi_1(T) & \psi_2(T) & ... & \psi_N(T) \end{pmatrix} \tag{2.3}$$

contains the normalized Gaussian basis functions evaluated at each time step. [15]

Algorithm 1 shows how a ProMP can be trained to learn a distribution over given trajectories.

```
[1]Data: A set of N trajectories with position observations $Y_i, i = 1, ..., N$ at time $t_i$
Input: Number of basis functions K, basis function width h, regression parameter $\lambda$.
Result: The mean $\mu_w$ and covariance $\Sigma_w$ of $p(w) \sim N(w|\mu_w, \Sigma_w)$.
foreach trajectory i do
    Compute phase: $z_i = t_i / t_i^{end}$;
    Generate basis: $\Psi_t = f(z_i, K, b)$, Equation 2.1;
    Compute the weight vector $w_i$ for trajectory i: $w_i = (\Psi_t^T \Psi_t + \lambda I)^{-1} \Psi_t^T \tau_i$.;
Fit a Gaussian over the weight vectors $w_i$;
$\mu_w = \frac{1}{N} \sum_{i=1}^{N} w_i$;
$\Sigma_w = \sum_{i=1}^{N} (w_i - \mu_w)(w_i - \mu_w)^T$;
return $\mu_w$, $\Sigma_w$
```

Algorithm 1: Learning Stroke-Based Movements

2.2 Relevance Weighted Policy Optimization

RWPO is a reinforcement learning algorithm. It determines the relevance of each policy parameter to each subproblem and uses this information to guide the sampling procedure in policy search. This method makes use of RWR [2].

[1] Alexandros Paraschos, Christian Daniel, Jan Peters and Gerhard Neumann *Using Probabilistic Movement Primitives in Robotics, p.7-9* 2018

The basic idea of RWPO is to first find out how much each policy parameter influences each objective. This information is then used to optimize the policy with respect to the objectives. The approach to answer how much each policy parameter influences each objective consists of learning a relevance function for each objective. A relevance function can for example be represented by a weighted sum of basis functions. In the framework proposed by Ewerton et al. [1], learning a relevance function with respect to a certain objective means finding a vector that leads to a high variability in the value of that objective and a low variability in the values of other objectives. These vectors are called relevance parameters and can be learnt by using RWR. RWR is an iterative algorithm that finds the best Gaussian distribution over parameters of interest in order to maximize the expected reward, given samples from the Gaussian distribution of the previous iteration. After learning a relevance function for each objective, RWPO uses this information to optimize a policy with respect to each objective. For this purpose, RWR is used again to maximize the expected reward with respect to the mean and variance of the policy parameters. In other words, the previous mean and variance of irrelevant policy parameters are preserved, while the mean and variance of relevant policy parameters are updated. [1]

Assume following task: A robot arm has to move to an object, grasp it and then place it somewhere else. An optimal policy for this task would be a trajectory which starts at a given starting position, passes close to the object's position and ends at the desired end position where the object has to be placed. This problem can be decomposed into three subproblems w.r.t. which a certain policy parameter is relevant and others are not. RWPO is a policy search method which identifies the relevance of each policy parameter to each subproblem in order to learn an optimal global policy. The basic idea is to derive the correlation between each policy parameter and each objective. This information is used to optimize the policy with respect to the objectives. In the case of the object placement task, the policy parameters are the elements of the weight vector w as in Equation 2.2. [1]

2.2.1 Relevance Functions

The original RWPO algorithm consists of two parts. First, the *relevance functions* have to be learnt, i.e. the algorithm has to derive a function for each objective, which represents how a given policy parameter influences the current objective. Ewerton et al. [1] first introduced weighted sums of sigmoid basis functions. In their framework, learning a relevance function with respect to a certain objective is equal to finding a vector ρ that leads to a high variability in the value of that objective and to a low variability in the values of other objectives. First, a Gaussian distribution $N(\mu_\rho, \Sigma_\rho)$ over the parameter vectors ρ is initialized with a certain mean and covariance. Then, parameter vectors ρ are sampled from this distribution and the relevance function f_o is computed for each sample. [1]
Ewerton et al. assume a Gaussian probability distribution $N(\mu_w, \Sigma_w)$ over the policy parameters w. The mean and the covariance matrix can be computed from an initial set of demonstrations or determined by the user. The algorithm weights the original variances with a relevance coefficient for each objective and for every sampled vector ρ, obtaining

$$\Sigma_w^{f_o} = \begin{pmatrix} \sigma_{w_1}^2 f_o(1) & 0 & \dots & 0 \\ 0 & \sigma_{w_2}^2 f_o(2) & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \sigma_{w_N}^2 f_o(N) \end{pmatrix}, \tag{2.4}$$

where $f_o(n)$ denotes the relevance function for objective o evaluated for policy parameter n and $\sigma_{w_n}^2$ is the variance in the diagonal of Σ_w. By doing that, a larger range of values is sampled for the policy parameters w_n that are more relevant to the objective o and a smaller range of values is sampled for the policy parameters that are less relevant to this objective. [1]
Each sampled vector w determines a trajectory with a certain distance to each via point. These distances can be used to compute a reward for each w. The reward function is

$$R_{\rho,o} = exp(\beta_{relevance}(\sigma_o - \sum_{i \neq o} \sigma_i)), \tag{2.5}$$

where σ_o is the standard deviation of the distance values for the current objective and σ_i is the standard deviation of the distances between the current objective and a via point i. To learn the relevance parameters ρ and thereby optimize the reward, RWPO uses RWR. RWR solves an optimization problem over the parameters of the distribution $N(\mu_\rho, \Sigma_\rho)$. The result is a relevance function for each objective o. [1]
Figure 2.1 shows the relevance functions learnt for a 3D problem with 3 via points. Each via point is represented by a relevance function. This relevance function gets activated for a certain range of weight indices. The learning had 30

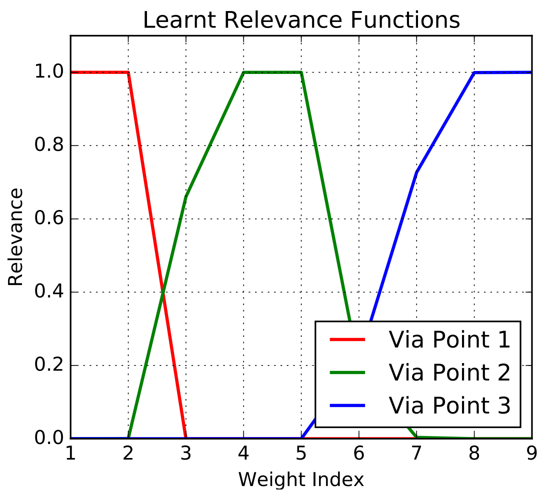


Figure 2.1.: Learnt Relevance Functions for 3 Via Points on a 3D Problem

iterations and 60 trajectories were sampled in each iteration. The learning took 11.24 seconds (Note: See A Appendix for the specifications of the computer used for the experiments).
The previous paragraph explained how the relevance functions can be computed according to [1]. However, different problems occur if the relevance functions are computed this way. These problems are stated in the next chapter. The next chapter also introduces another way to compute the relevances, namely by using *Pearson's Correlation Coefficient*.

2.2.2 Policy Optimization

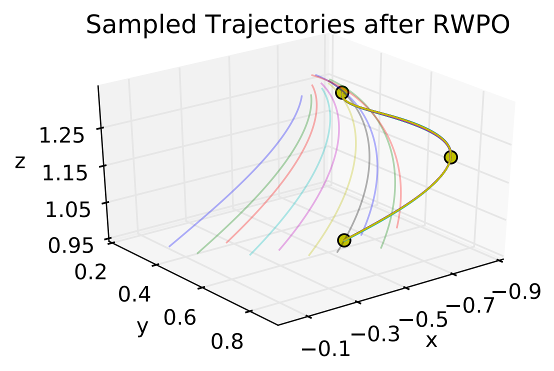
In the second part of the algorithm, the learnt relevance functions are used to optimize a policy w.r.t. each objective. Now that the relevance functions f_o^* are learnt, the algorithm can sample from the distribution $N(\mu_w, \Sigma_w^{f_o^*})$. After that, RWR is used again in order to maximize the reward

$$R = exp(-\beta_{policy} d_{objective}) \tag{2.6}$$

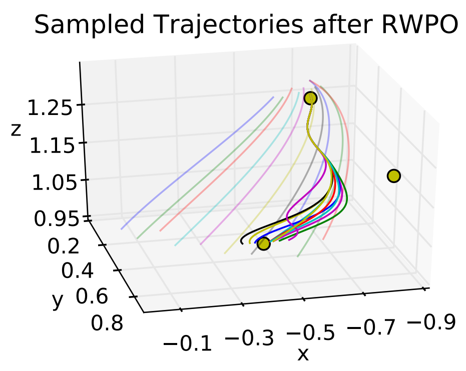
w.r.t. μ_w and $\Sigma_w^{f_o^*}$, where $d_{objective}$ is the distance between the current objective and the position of a sampled trajectory, where the trajectory should pass through the current objective. In each iteration, the previous variance of the parameters that are less relevant to the objective o is kept , while the variance of the parameters that are more relevant to this objective is updated. [1]
Figure 2.2(a) shows 10 sampled trajectories after optimizing the ProMP with RWPO w.r.t. the given via points. The transparent trajectories show the initial demonstrations. The policy optimization took 2.02 seconds, having 60 iterations and sampling 120 trajectories in each iteration. The collected rewards together with the distances to the different via points are shown in Figure 2.3. The importance of the relevance functions is illustrated in Figure 2.2(a). Relevance functions help the algorithm sample smartly. When using common RWR (i.e., the relevances are all set to 1), the algorithm does not know how to sample in order to minimize the distances for every via point, which leads to a high variance regarding the optimized trajectories. This is shown in Figure 2.2(b).

2.2.3 Existing Experiments with RWPO

Section 6 of [1] introduces an experiment with RWPO. In this user experiment, users had to teleoperate a small simulated cube attach to the end-effector of the Haption Virtuose 6D device. The device is shown in Figure 2.4. [1]
The Haption Virtuose 6D can provide force feedback to the user by simulating an object attached to its end-effector. The system can be interpreted as a mass-spring-damper system. The Virtuose API can compute stiffness and damping coefficients that guarantee the stability of the system based on the mass and inertia of the virtual object. The intensity of the force produced by this system can be rescaled by a certain factor. Force feedback can be provided to a user along a learnt trajectory. If the standard deviation at a certain part of the distribution is high, the haptic device becomes compliant in that region, while it becomes stiff given a low standard deviation. The virtual object always lies along the mean trajectory of the distribution. [1]
In the experiment, the users were instructed to begin at the position marked by the yellow sphere, pass through the

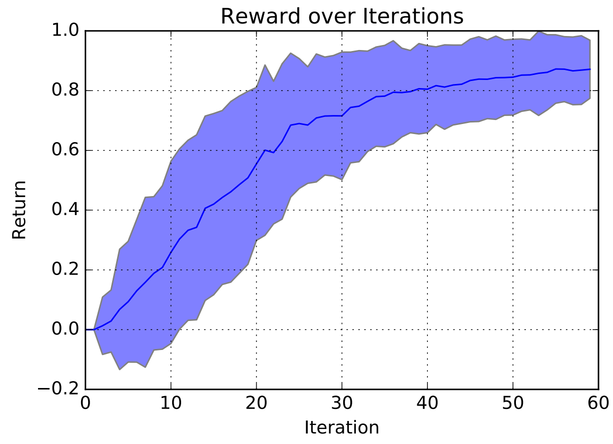


(a) Sampled Trajectories after RWPO

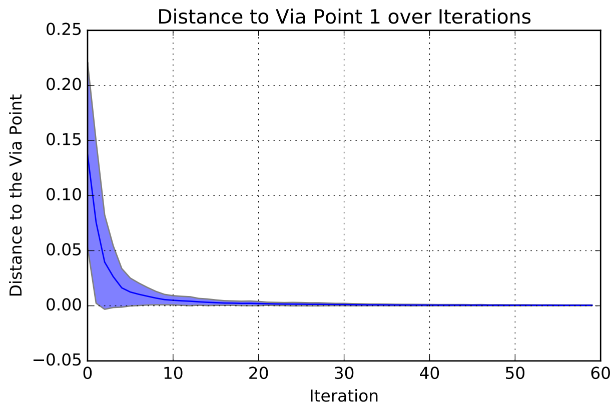


(b) Sampled Trajectories after RWR, i.e. the Relevances were all set to 1

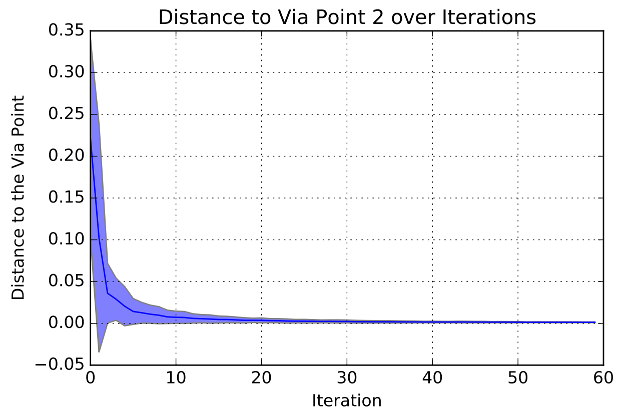
Figure 2.2.: RWPO Experiments for 3 Via Points



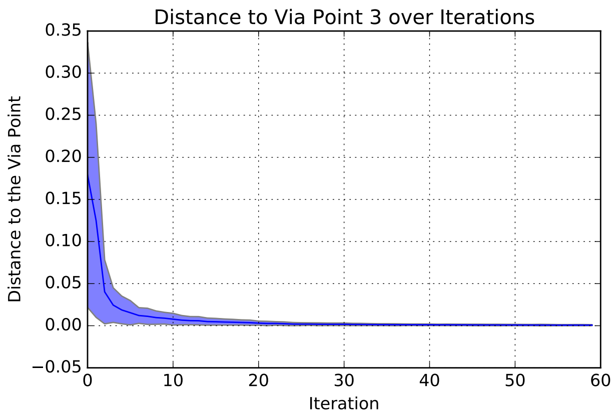
(a) Collected Reward over Iterations



(b) Distance to Via Point 1



(c) Distance to Via Point 2



(d) Distance to Via Point 3

Figure 2.3.: Collected Reward and Distance to the Via Points over Iterations for the 3D - 3 Via Point Learning Problem

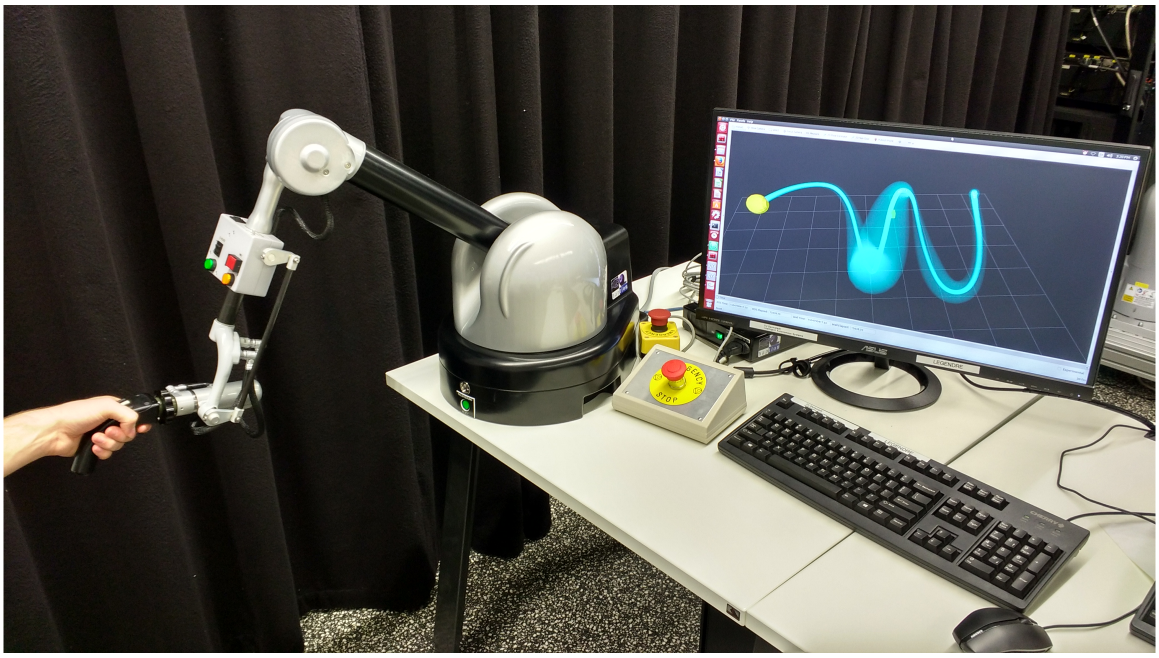


Figure 2.4.: Human Manipulating the Haption Virtuose 6D
2

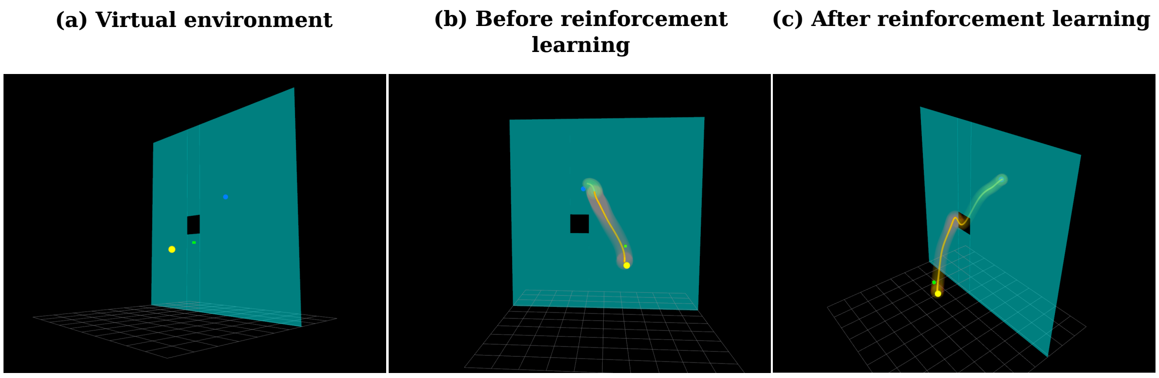

(a) Virtual environment
(b) Before reinforcement learning
(c) After reinforcement learning

Figure 2.5.: Virtual Environment and Trajectory Distributions
3

center of the window in the wall and end at the position marked by the blue sphere. Figure 2.5(a) shows the virtual environment, containing the via points, the cube and the wall. In the first phase of the experiment, users tried to perform the task ten times without force feedback. The results are shown in Figure 2.5(b). The users did not manage to navigate the cube through the hole in the wall. Subsequently, the system optimized the distribution over trajectories using the RWPO algorithm. The optimized trajectories were then used to give force feedback to the users. The users were requested to try to perform the task with force feedback ten times. The use of force feedback decreased the distance to the center of the window for all and the distance to the end for three out of five users. This is shown in Figure 2.5(c). [1]

[2] Marco Ewerton, David Rother, Jakob Weimar, Gerrit Kollegger, Josef Wiemeyer, Jan Peters and Guilherme Maeda *Assisting Movement Training and Execution with Visual and Haptic Feedback, p.2* 2018

[3] Marco Ewerton, David Rother, Jakob Weimar, Gerrit Kollegger, Josef Wiemeyer, Jan Peters and Guilherme Maeda *Assisting Movement Training and Execution with Visual and Haptic Feedback, p.26* 2018

3 Implementation Details

3.1 Learning the Relevance Functions

In [1], Ewerton et al. model the relevance functions directly as weighted sums of sigmoids. The optimal relevance functions are computed by solving an optimization problem w.r.t. the reward function stated in 2.5. This reward function rewards a high standard deviation of the sampled values for the current objective and punishes a high standard deviation of the sampled values for the other objectives. However, this way of computing the relevance functions was tested on only three objectives: A start, center and end point on a given trajectory. [1]

A first problem arises when one tries to apply the same reward function 2.5 on more than three objectives. The reward function takes into account the standard deviation of the distances for the current objective σ_o and a sum over the standard deviations of the distances between the current objective and the other via points. The sum is subtracted from σ_o. Usually, the sum is smaller than σ_o because the other via points are far from the current objective, providing a small standard deviation over the distances. However, the value of the sum grows with the number of objectives. For four objectives, the sum is already so big that the current objective is punished too hard in the reward function. This prevents the algorithm from learning properly, as the value of the reward function gets too small. Figure 3.1(a) shows the learnt relevance functions for a 3D problem and 5 via points when using the method described in [1]. Relevance function 4 is zero because there were numerical instabilities during the computation, caused by the high punishment in the reward function. Figure 3.1(b) shows the reward for via point 4. The zero variance is caused by the numerical instabilities.

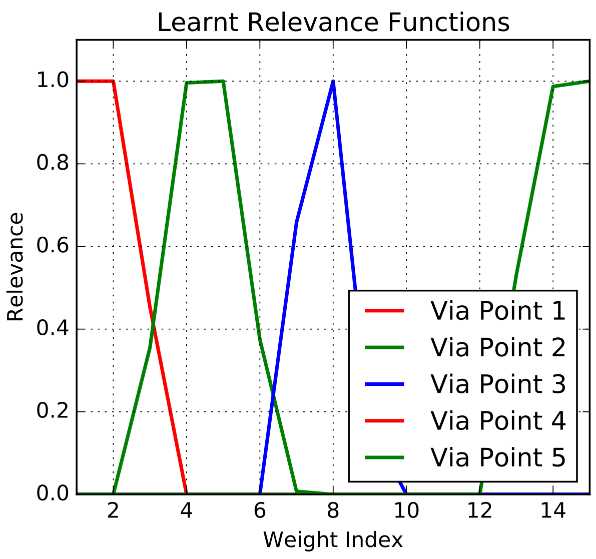
Another problem of computing the relevances like in [1] is the expensive computation. One has to compute the standard deviation of the distances between the current objective and each other objective over all sampled trajectories in each iteration. This has a complexity of $O(n_{iterations} * n_{samples} * n_{objectives})$. After that, the reward for each objective has to be computed. This requires going over all samples again. Therefore, the complexity for this procedure is the same as for computing the standard deviations. The complexity of computing the relevance functions is therefore $O(2 * n_{iterations} * n_{samples} * n_{objectives}) = O(n_{iterations} * n_{samples} * n_{objectives})$. Furthermore, modeling the relevance functions explicitly means a more complicated implementation and the learning requires more hyperparameters (See Section 3.2 Hyperparameters).

Regarding the sum problem, this can be fixed by dividing the sum by the number of objectives minus one, s.t. $R_{\rho,o} = exp(\beta_{relevance}(\sigma_o - (N-1)^{-1} \sum_{i \neq o} \sigma_i))$, where $N > 1$ is the number of objectives. Experiments prove that this concept works, as illustrated in Figure 3.2.

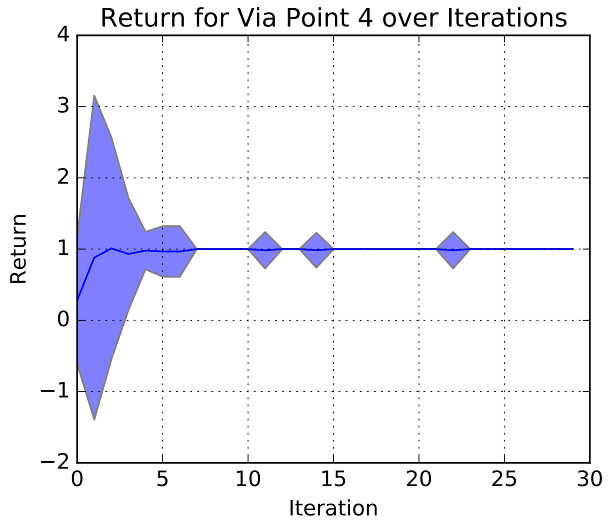
To solve the high computation time problem, Ewerton proposes a new way of computing the relevances. It makes use of PCC. Simply put, PCC indicates how two variables are linearly related. PCC is a value between minus one and one. A PCC with value plus or minus one means that two variables x and y are linearly related. Plus one means total positive, zero means no and minus one means total negative linear correlation. A relevance coefficient can be expressed as the absolute value of the PCC between a random weight and the minimal distance between a point on a sampled trajectory and the current objective. In other words, one can define a covariance matrix Σ_{weight} with small covariance, e.g. 10^{-7}. This covariance matrix and the mean of the ProMP are then used to sample weighted trajectories. Finally, the PCC between the minimum of the distances to the current objective and the sampled weights is computed. The absolute of the PCCs can now be used as relevance coefficients in order to weight the covariance matrix, as shown in 2.4. Computing the relevances that way has a complexity of $O(n_{samples} * n_{objectives})$.

Experiments also show that computing the relevances is much faster when using PCC. Figure 3.3(a) shows the learnt relevance functions for a 3D problem with 4 via points. The optimized trajectories are shown in Figure 3.3(b). The RWPO algorithm took 4 seconds in total, compared to 13 seconds, when modeling relevance functions explicitly. The results of this approach for the same problem are shown in Figure 3.3(c) and (d). 20 iterations and 40 samples per iteration were used for computing the relevances the old way and 60 iterations and 120 samples per iteration were used in the policy optimization part.

Using PCC to compute the relevance coefficients brings many advantages and avoids the problems of the approach stated in [1]: The relevance functions and a reward function do not have to be modelled directly and the computational cost is reduced, since only the distance between the current objective and the points along the current trajectory have to be computed for every sample, instead of computing the standard deviation of the distances of a trajectory point to each via point for each objective. The computation time for learning the relevance coefficients is reduced heavily, because fewer distances and no standard deviations have to be computed. However, the relevance functions are not optimized anymore,



(a) Learnt Relevance Functions



(b) Numerical Instabilities in the Reward for the 4th Via Point

Figure 3.1.: Learning of Relevance Functions for 5 Via Points in a 3D Problem without Sum Normalization

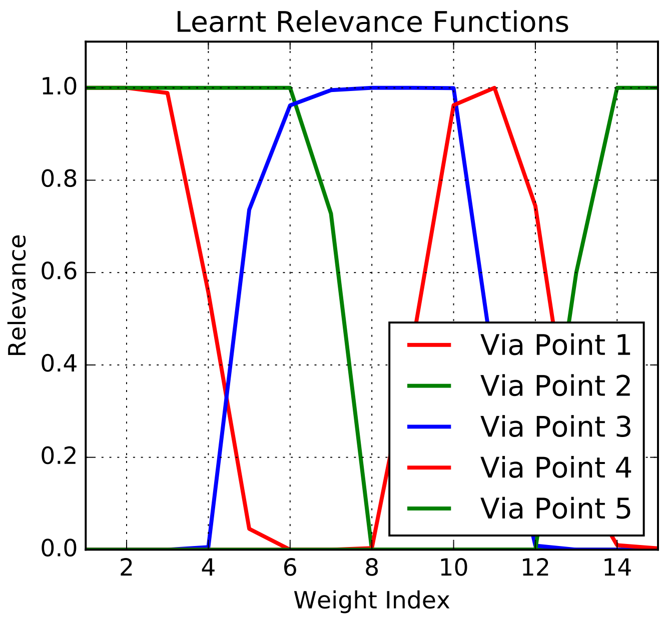
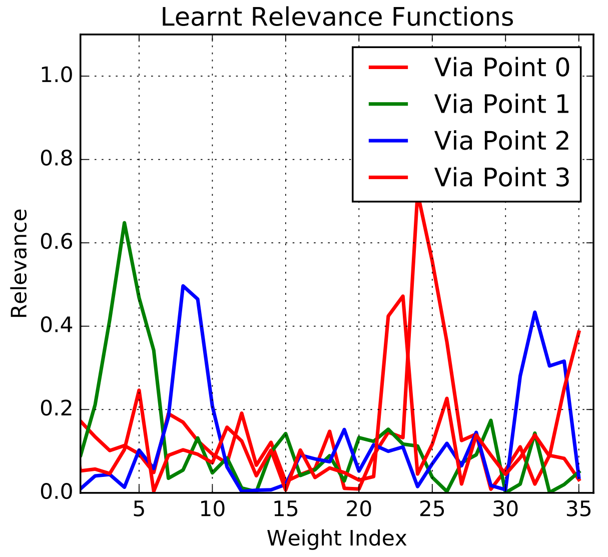
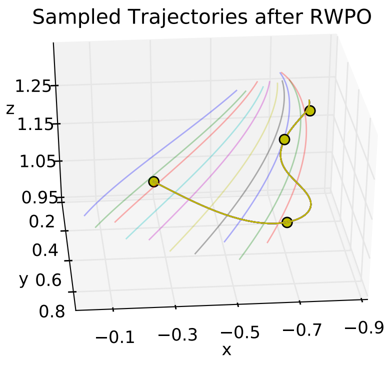


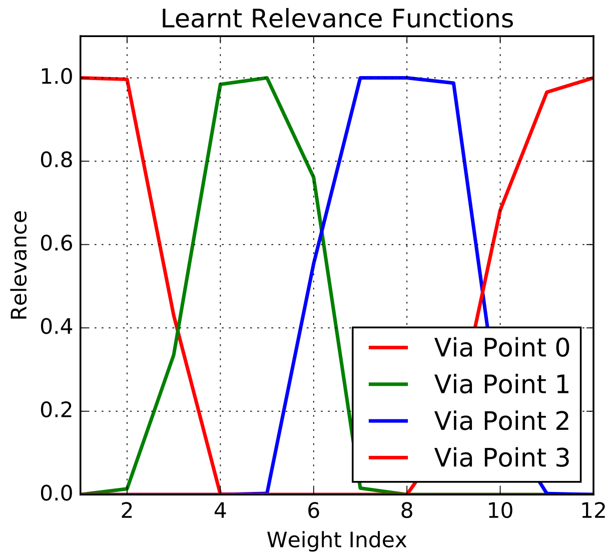
Figure 3.2.: Learnt Relevance Functions for 5 Via Points with Sum Normalization



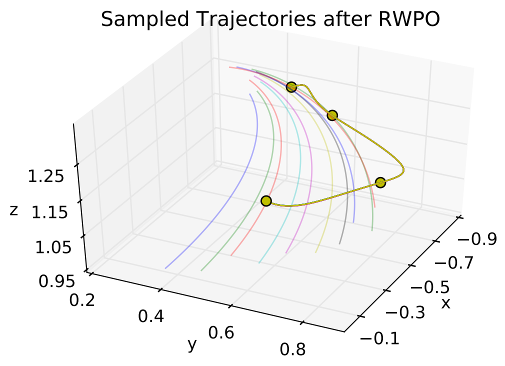
(a) Relevance Coefficients Using PCC



(b) Optimized Trajectories Using PCC Relevance Coefficients



(c) Learnt Relevance Functions Using Explicit Modelling



(d) Optimized Trajectories Using Explicit Relevance Functions

Figure 3.3.: RWPO Results; Indirect vs Direct Modelling of Relevances; 3D problem with 4 Via Points

since there is no explicit reward function. Therefore, a first intuition is that the relevance coefficients need to be updated online during the policy optimization. Another intuition is that online updates would make the algorithm more dynamic. For example, if a relevance function which is not able to optimize a distance to a certain via point was learnt in the first iteration, another, "better" one can be learnt a few iterations later.

3.1.1 Unknown Via Points

A problem arises when the PCC method is used to learn the relevance coefficients for via points which are far from the demonstrations. The algorithm is "unsure" and provides results which are similar to the results when using normal RWR, especially when the relevances are updated frequently. Figure 3.4 shows the demonstrations and generated via points used for the experiments. The results of the experiments with PCCs as relevances are shown in Figure 3.5. (a) is a plot of the estimated PCCs during the last iteration of RWPO. For this scenario, the relevances were updated in each RWPO iteration. (b) shows the sampled trajectories after the termination of the algorithm. It shows that no feasible solution which goes through all via points was found. Furthermore, the variance of the learnt ProMP is high, as the sampled trajectories differ from each other. The learnt relevances after the last update for RWPO with relevance update each 10 iterations are shown in Figure 3.5(c). The trajectories after RWPO are shown in (d). The plot illustrates that the ProMP has low variance, as the sampled trajectories overlap. Again, no feasible policy was found regarding the via points. This is also the case when not updating the relevances at all. The relevances during this scenario are plotted in Figure 3.5(e) and the corresponding trajectories after RWPO can be found in (f).
On the other hand, explicit relevances are also of no help, as can be seen in Figure 3.6. Although the relevances plotted in (a) look fine, again no feasible policy was found when looking at the sampled trajectories at (b). The "best" results are provided by explicit relevances and PCCs without update, because the trajectories manage to pass near the 1st, 2nd and last via point. Furthermore, the variance of the learnt ProMP and consequently the variance of the trajectories is lowest. For the experiments, the explicit relevances were learnt using 30 iterations á 60 samples. 120 iterations á 240 samples were used for the policy optimization with both explicit and implicit relevances. The scenario with explicit relevances took the longest time, which was approximately 39 seconds in total. The experiments show that both explicitly and implicitly modeled relevances struggle with unknown via points. The PCC variant without relevance coefficients update took only 11.49 seconds, while the one using explicit relevance functions took 27 seconds for learning the relevances and another 11.31 seconds for the policy optimization. This shows that using PCCs as relevances is far more computationally efficient.

To sum up, direct and indirect relevance modeling have problems on via points far from the demonstration. However, they manage to optimize the trajectories s.t. they pass through the first and last via point. PCC works best if the relevances are not updated. In that case, it shows similar results like RWPO with direct relevance modelling, but it is much faster, as the computation of the explicit relevances is expensive.

3.1.2 Rescaled PCCs

In the beginning of this section, an alternative way of learning the relevance coefficients was shown. It computes the PCCs between random weights and the corresponding distances to the via points when applying those weigths onto the covariance matrix of the ProMP. The absolute values of the computed PCCs are the relevance coefficients.
A problem of the implicit relevance modelling is that it does not provide good results for via points which are far from the initial demonstrations. Increasing the number of iterations and samples for this kind of problem is of little use, as the reward already converges at about 100 iterations. This is illustrated in Figure 3.7. Also, the best results were achieved without updating the relevances during the policy optimization.

The problem of bad trajectories for unknown via points when using PCCs can be solved partially by taking a closer look at the relevance functions. Figure 3.8(a) shows the relevances and Figure 3.8(b) shows the sampled trajectories after policy optimization, computed using PCC. This time, 100,000 trajectories were sampled for the calculation of the PCCs. No PCC updates were used. The computation of the PCCs took about 4 seconds. By using that many trajectory samples, the noise due to a specific sample is removed.
It will shortly become clear that the relevances computed using PCC resemble the explicitly modelled ones. The covariance matrix Σ_w of the ProMP is a NxN-matrix. The number of relevance coefficients (weights) has to be N, as shown in Equation 2.4. The value of N depends on the number of Gaussian basis functions and degrees of freedom: $N = T * n_{dofs}$, where T is the number of Gaussian basis functions of the ProMP and n_{dofs} is the number of degrees of freedom. In this experiment, the number of degrees of freedom is three, because the learning is done in Cartesian. The number of Gaussian basis functions is $T = n_{dofs} * n_{viapoints}$ (This is justified by a heuristic in Section 3.2). This yields $N = n_{dofs}^2 * n_{viapoints} = 36$. Regarding the PCC relevances, it now becomes clear that the relevances have different maximum values for each degree

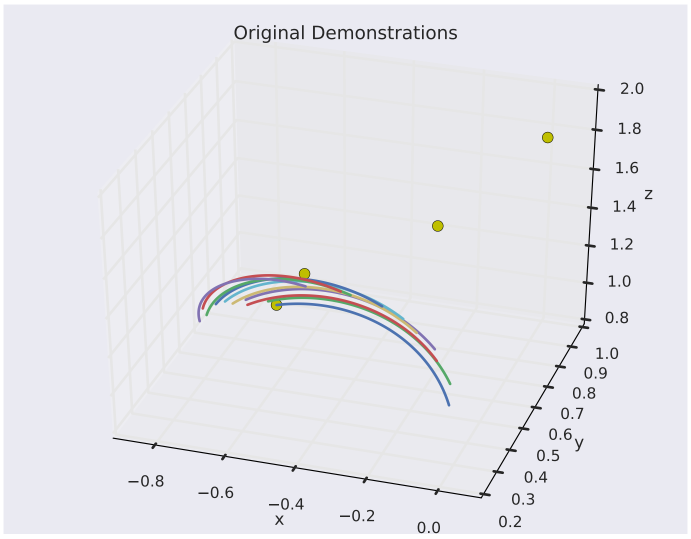
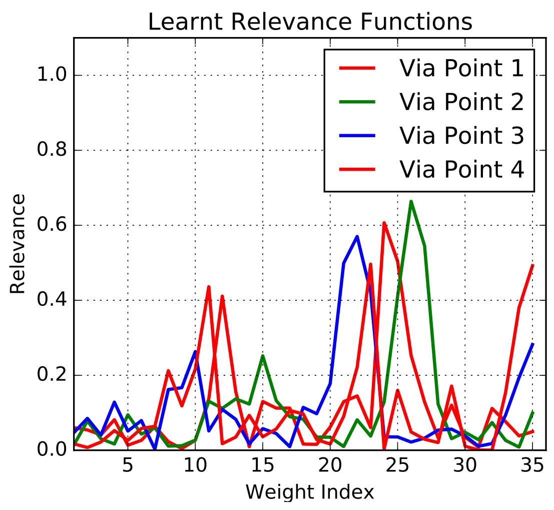


Figure 3.4.: Initial Demonstrations and Randomly Generated Via Points far from the Demonstrations

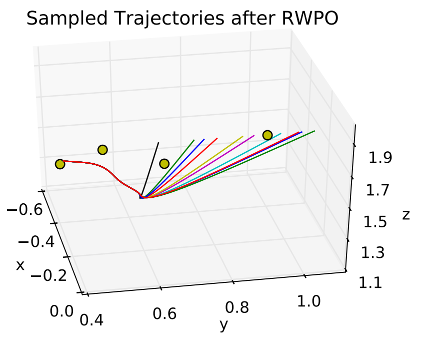
of freedom. Weight indices 0 to 11 are responsible for the first DOF, 12 to 23 for the second one and 24 to 35 for the last one. On the other hand, when modelling the relevances explicitly, the same relevances are applied n_{dofs} times to Σ_w. That means that PCC relevances are computed separately for each degree of freedom. Furthermore, PCC relevances rarely reached a value above 0.7 during the experiments. On the one hand, explicit relevances are sigmoids with values between 0 and 1. That means that explicit relevances are more certain per definition, as they are determined to have a value of 1 when they are most certain. PCCs on the other hand are aware of uncertainty and yield lower values in general. An interesting fact regarding uncertainty of PCCs is seen in Figure 3.8(a): The PCCs are uncertain around weight indexes 15 and 20. Those are responsible for distance minimization along the y axis. That means that the algorithm does not know how to minimize the distance to the via points along the y axis. On the other hand, explicit relevances are certain per definition and they have the same value for each degree of freedom.

In order to achieve relevances which reach values of 1 and look similar to explicit relevances, one could try to rescale the PCCs. A first idea would be to divide each relevance by the maximum observed relevance value. This means multiplying each relevance with a scaling factor s, where $s = 1/max(relevances)$. The resulting PCCs and sampled trajectories after RWPO are shown in Figure 3.8(c) and (d). Still, not every relevance function has a maximum of 1. Furthermore, there is still some variance between the second and third via point regarding the trajectory samples.

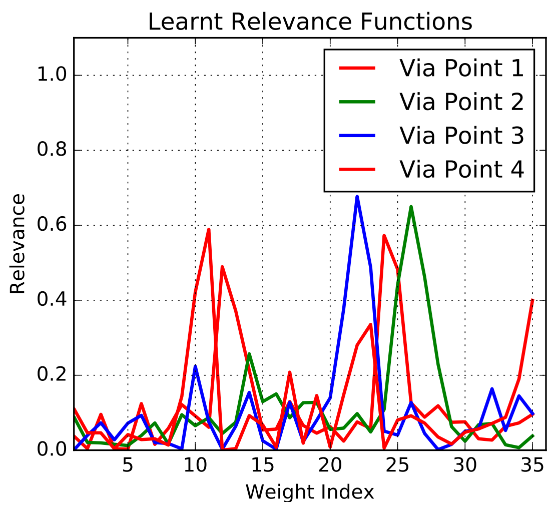
In order to exploit the relevances more and get results similar to the results when using explicit relevances, the absolute values of the computed PCCs can be rescaled further. This can be done by choosing a different scale factor s, s.t. $x_{threshold}s = 1$, where $x_{threshold} < 1$ is a threshold value of a relevance function, e.g. 0.1. If $xs > 1$ for a given relevance value x, xs is set to 1. In the experiments, $x_{threshold} = (2n_{viapoints})^{-1}$ was chosen. This yields $s = ((2n_{viapoints})^{-1})^{-1} = 2n_{viapoints} = 8$. It means that relevance values of $8^{-1} = 0.125$ and above are set to one. The purpose of this heuristic is to make the scaling factor dependant on the number of via points. The rescaled PCCs for the current problem are shown in Figure 3.8(e). The sampled trajectories after using these PCCs in the policy optimization are shown in Figure 3.8(f). The plot illustrates that the algorithm managed to provide better results than the algorithm which used non-rescaled PCCs, since the sampled trajectories pass through the 2nd via point. This shows that rescaled PCCs resemble explicit relevances and lead to better results than the normal PCCs. Normal PCC tend to "stick" to the demonstrations and explore less, as the sampled trajectories prefer to follow the initial demonstrations instead of going through the 2nd via point. The results show that rescaled PCCs can be used to improve the performance and computation time (Only 4 seconds were needed for the PCC relevances, compared to 27 seconds for the explicit relevances) in a setup where the via points are far from the original demonstrations. Rescaled PCC show similar results as explicit relevances, but they are a lot faster to compute. A difference between rescaled PCCs and explicit relevances is that the rescaled PCCs inherit the uncertainty of PCCs, while explicit relevances have the same certainty for each degree of freedom per definition. The "hole" between indexes 15 and 20 in Figure 3.8(c) and (e) was inherited from the initial PCCs in 3.8(a).



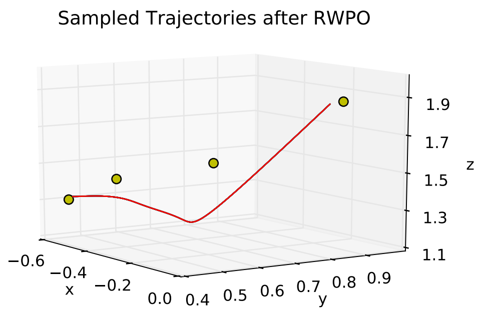
(a) Relevance Coefficients when Using PCC; PCC Update every Iteration



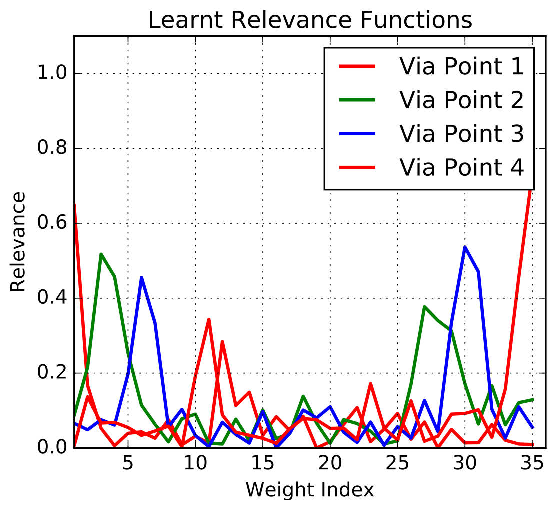
(b) Sampled Trajectories when Using PCC; PCC Update every Iteration



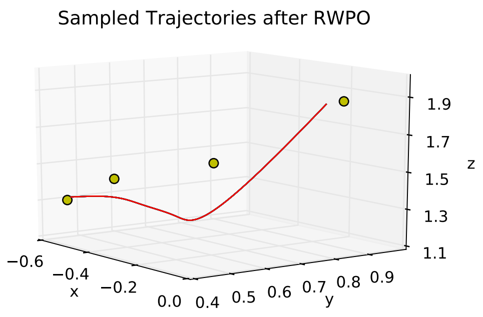
(c) Relevance Coefficients when Using PCC; PCC Update every 10 Iterations



(d) Sampled Trajectories When Using PCC; PCC Update every 10 Iterations

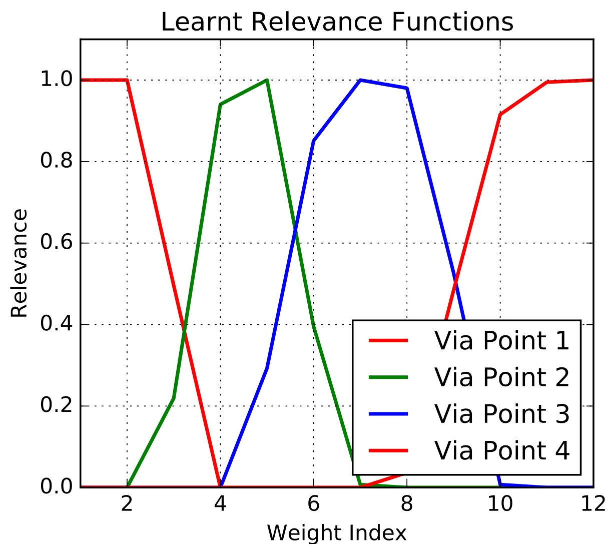


(e) Relevance Coefficients When Using PCC; No PCC Update

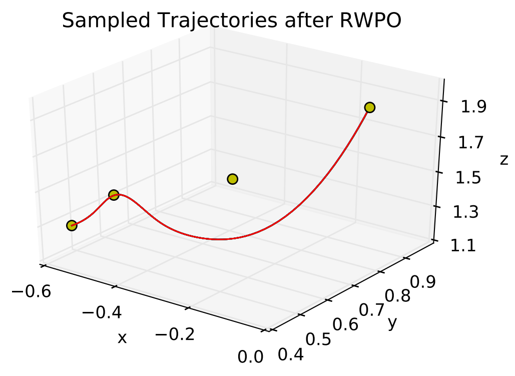


(f) Sampled Trajectories When Using PCC; No PCC Update

Figure 3.5.: Comparison Between Different Update Rates for the Relevances on a 3D Problem with Via Points far from the Original Demonstrations; Implicit Relevance Modelling



(a) Learnt Relevance Functions



(b) Optimized Trajectories

Figure 3.6.: Results of Explicit Relevance Modelling on a 3D Problem with Via Points far from the Original Demonstrations

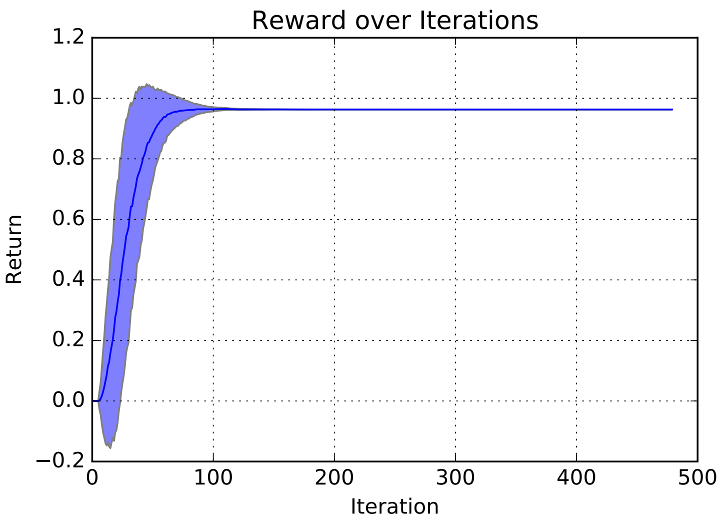


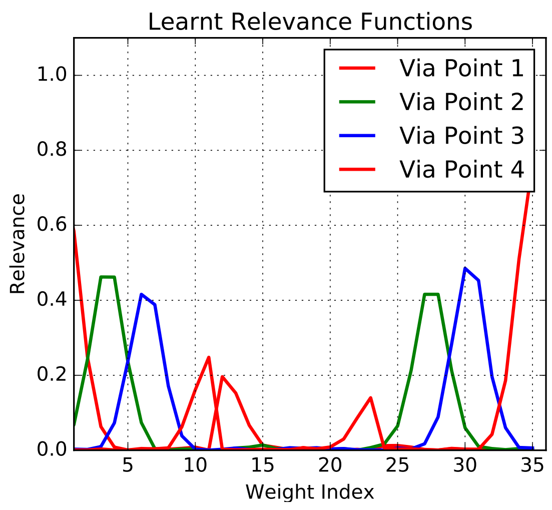
Figure 3.7.: Reward over Iterations for the 3D Problem with Via Points far from the Original Demonstrations

To sum up, RWPO provides best results when using explicit relevance functions. However, the high computation time makes explicit relevances infeasible for more complex problems with multiple via points and DOFs. This is why Ewerton proposed using the PCC between some random weights and the distances between the corresponding trajectories and each via point as relevance coefficients. However, PCC relevances are more uncertain than explicit relevances, they are prone to noise and never reached a value higher than 0.7 during the experiments. Therefore, PCCs provide inaccurate results. To solve the noise problem, a high number of trajectories can be sampled for the PCC computation. To make the PCCs reach higher values than 0.7 in practice, the computed PCCs can be rescaled s.t. a value above a given threshold gets scaled to 1. Experiments show that these fixes provide accurate results, while keeping the computation time low. This approach of computing the relevances is therefore going to be preferred in the remaining experiments.

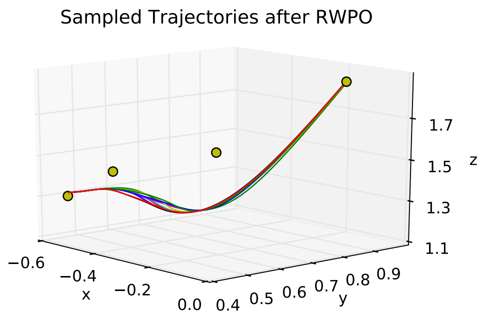
3.2 Hyperparameters

3.2.1 ProMP

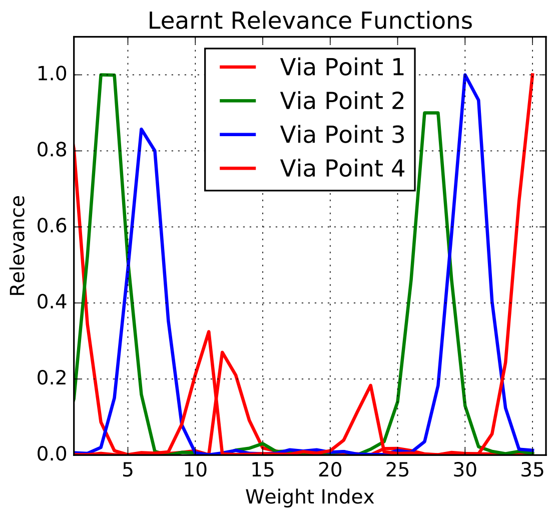
RWPO optimizes a ProMP learnt from demonstrations based on given objectives. The ProMP has following hyperparameters: The number of Gaussian basis functions and a regularizer λ for the Ridge Regression. Regarding the number of



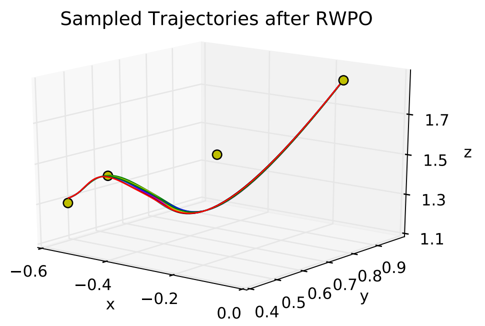
(a) Relevance Coefficients when Using PCCs



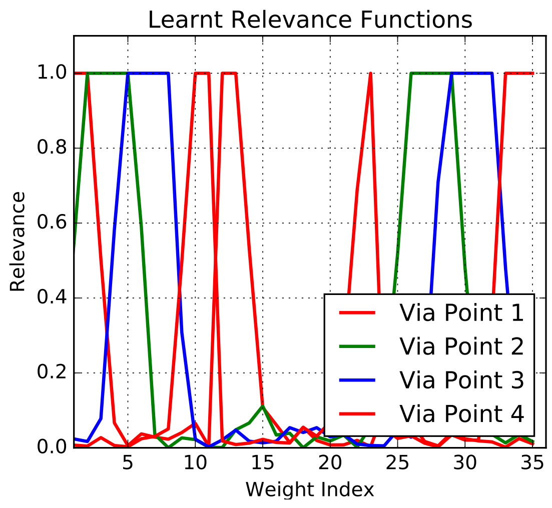
(b) Sampled Trajectories when Using PCCs



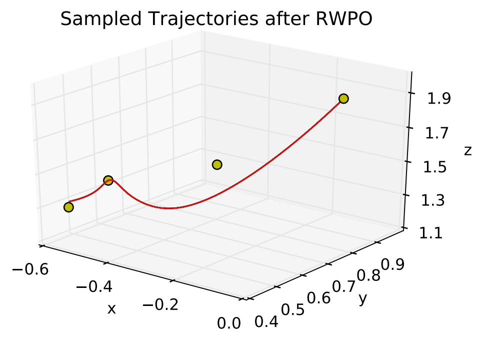
(c) Relevance Coefficients when Using Rescaled PCCs; Scaling Factor is the Reciprocal of the Maximum



(d) Sampled Trajectories when Using Rescaled PCCs; Scaling Factor is the Reciprocal of the Maximum



(e) Relevance Coefficients when Using Rescaled PCCs; Scaling Factor is the Reciprocal of a Threshold



(f) Sampled Trajectories when Using Rescaled PCCs; Scaling Factor is the Reciprocal of a Threshold

Figure 3.8.: Comparison Between Normal and Rescaled PCCs on a 3D Problem with Via Points far from the Original Demonstrations

basis functions, a basis function for each degree of freedom and each objective, i.e. $K = n_{dofs} * n_{viapoints}$ seems feasible. All the experiments follow that rule except the Haption Virtuose 6D experiments in Chapter 5. The Haption experiments required a higher number of basis functions, since the trajectories had a lot more time steps than in the other experiments (80 vs ca. 500). A too low number of basis functions means a too simple model which is unable to learn and a too high number can lead to a too complex model prone to overfitting. In the experiments, $\lambda = 10^{-11}$ was chosen.

3.2.2 Relevance Functions

Explicit Modelling of the Relevances

When modelling the relevance functions explicitly, one has to choose a proper reward weight $\beta_{relevance}$ for the reward function. A smaller weight means that samples far from the via points are not punished that hard. A weight which is too big can cause numerical instabilities and even an overflow. In the experiments, $\beta_{relevance} = 200$ has proved to be a good reward weight.
Another hyperparameter is the steepness k of the sigmoids used for the relevance functions. In the experiments, k was set to 1.

PCCs as Relevance Coefficients

When using the PCC between some weights and the distances to the via points, one has to choose the scale of the weights. In the experiments, the weights were chosen to have a variance of 10^{-7}. The update rate of the relevance coefficients also has to be set manually. It depends on the problem. For good demonstrations, the relevance coefficients can be updated in every iteration. For via points which exceed the space spanned by the demonstrations, computing the relevances only once has proven more effective, as shown in the previous section.

3.2.3 Policy Optimization

For the policy optimization, the reward weight β_{policy} has to be set manually. Again, it is set to 200, with similar considerations as for $\beta_{relevance}$.
Two more hyperparameters, which are justified in Subsection 4.3.2, are the length and jerkiness weights β_1 and β_2 (See Equation 4.1). Jerk and length punishment were used during the experiments in Subsections 3.1.1 and 3.1.2 and during the experiments in gazebo (Chapter 4) and rviz (Chapter 5). These parameters should be set s.t. the values obtained by multiplying them to the jerkiness and length of the sampled trajectories during RWPO have the same domain as the values of the computed distances for the first iterations of the algorithm. For the 3D tasks in 3.1.1 and 3.1.2, β_1 was set to 0.1 and β_2 was 1000. In the 7D learning experiments in gazebo, the two parameters were set to $\beta_1 = 0.15$ and $\beta_2 = 100$. For the rviz experiments, the values $\beta_1 = 0.1$ and $\beta_2 = 10,000$ were used.

4 Gazebo Experiments

In this chapter, the RWPO algorithm is applied in an object disentangling task in a simulated 3D environment. The task is to grasp an object, which is entangled with another object, using a robot arm. Then the object has to be transported to a given point in task space. The setup and implementation details are described in the following sections.

4.1 Introduction to Gazebo and ROS

4.1.1 Gazebo

Gazebo is an open-source 3D simulator made for simulating robots in complex indoor and outdoor environments. It is similar to game engines, yet it offers much more accurate physics simulation, a suite of sensors and interfaces for both users and programs. It has an integrated GUI and a model editor. A robot can be designed directly in Gazebo. [16]

4.1.2 ROS

Robot Operating System (ROS) is an open-source, meta-operating system for robots. It provides typical OS services, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also has tools and libraries for obtaining, building, writing, and running code across multiple computers. [17]
ROS is a distributed framework of processes (aka Nodes). These nodes can be grouped into Packages and Stacks, which can be easily shared and distributed. Nodes can also publish messages as well as subscribe to different topics to receive messages. There is always a Master node which helps all the other nodes find each other. Services are another way that nodes can communicate with each other. Services allow nodes to send a request and receive a response. [18]
Different nodes can have different tasks. Regarding a robot arm, different nodes can be defined for publishing e.g. the current position and orientation of the end-effector (odometry). One can then define a node which subscribes to the odometry node and computes a distance between an object in task space and the current position of the end-effector. [18]

The experiments described in the following section were conducted in the simulation environment gazebo. ROS nodes were used for recording demonstrations and running the trajectories sampled after RWPO. Furthermore, the robot model uses different nodes, e.g. for odometry publishing.

4.2 Experimental Setup

The simulation setup for disentangling is shown in Figure 4.1. The setup includes two 7DOF robot arms called "iiwas" and a table. On the table, there are two U-shapes with an S-shape hanging on each of them. For the experiments, only the left robot arm is being controlled. The task can be extended to use both arms in the future. The main aim of the experiments is to learn two different trajectories, each one starting at the position of the end-effector, as shown in the figure. These trajectories are going to be executed by the robot arm, s.t. for each trajectory, it goes to one of the shapes, picks up the S-shape, brings it to a certain point in task space and puts it down. Collisions have to be avoided by the robot along the trajectories.

4.3 Learning Proper Trajectories

To learn how to control a robot arm in such a way that it passes through defined via points in order to avoid collisions or reach a desired object like the s-shape, one has to learn trajectories over joint configurations. Therefore, the learning has to be done on 7D trajectories containing joint configurations in the iiwas case.
One could also try to learn in 3D and compute the inverse kinematics for the sampled trajectories in the end. Computing the inverse kinematics can be done using *MoveIt!* [19], a ROS kinematics framework for gazebo. However, MoveIt! does not always find a solution for a specific 3D via point. This can be fixed by interpolating between the solutions for two different points in task space. However, experiments with the setup shown in Figure 4.1 show that MoveIt! does not respect the fact that the robot arm should always have a configuration s.t. the end-effector is above the table.

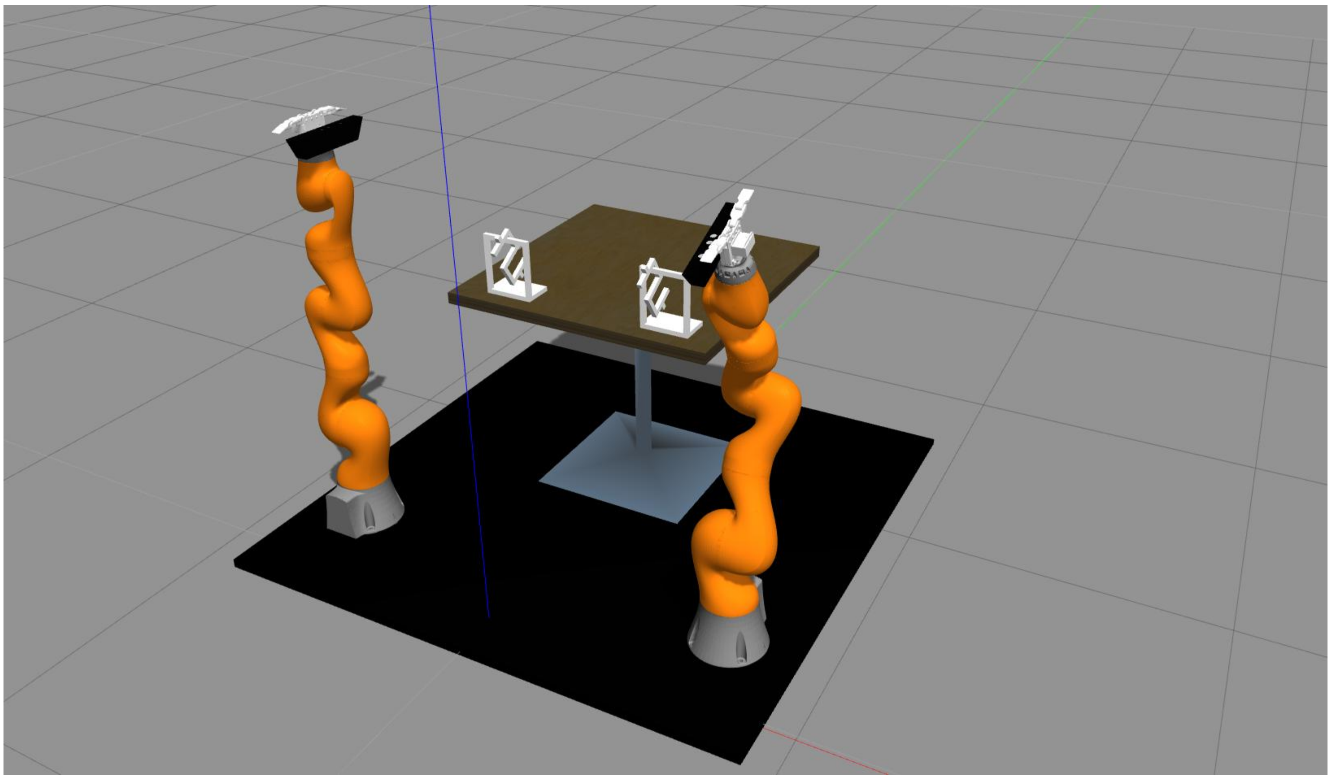
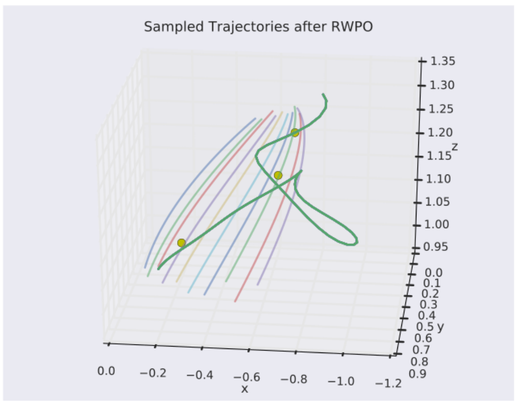


Figure 4.1.: The Simulation Setup in Gazebo

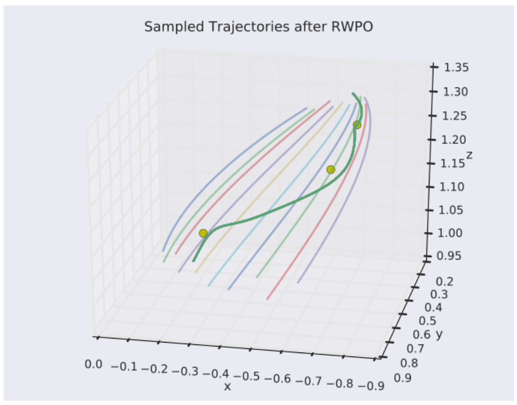
4.3.1 Reward Computation

The reward function 2.6 takes into account the minimal distance along a sampled trajectory to the current via point. This distance can be computed between two 3D via points, corresponding to a certain joint configuration. Defining 3D via points is convenient, because a 7D configuration for a 3D point in task space is not always known a priori. However, when learning in 7D, the sampled 7D trajectories will have to be mapped onto corresponding 3D trajectories in order to be able to compute the distances. This can be done by computing the forward kinematics for each sampled trajectory. This can again be done by MoveIt!. However, computing the forward kinematics that way takes a lot of computation time. Experiments show that learning for one via point with 3D mapping via MoveIt! takes about two minutes, while it takes only 23 seconds without mapping.
There are other, more efficient ways of computing the forward kinematics. These include efficient C implementations and *tf*[20]. tf is a ROS framework used to operate in a distributed system. It keeps track of all the information about the coordinate frames of a robot and makes it available to all ROS components in the system. These ways of computing the forward kinematics are considered future work and are not discussed in this paper.

To avoid mapping into 3D, the via points can be modelled directly in 7D. However, the right joint configuration to reach a point in space is not always given. It depends on the demonstrations and on the specific task (For example, the robot arm can be brought into a certain position manually by the human in the lab, but not in a dangerous area, like e.g. a radioactive site).
When the joint configuration for a certain end-effector position in task space is not known a priori, one can try to first optimize roughly for the 3D point. The robot will learn 7D trajectories which contain new joint configurations which are near to the one which corresponds to the 3D via point. To learn these trajectories, the reward has to be computed according to the distance between a point on a 3D trajectory and the 3D via point. This requires calculating the forward kinematics. The learnt 7D trajectories can now be used as new demonstrations. When mapped into 3D, these trajectories will contain points which are near to the desired 3D via point. The joint configuration which belongs to the 3D point which is nearest to the desired 3D via point can be used as a via point. The 7D trajectories can then be optimized with respect to this 7D via point. This procedure is considered future work and will not be investigated in this paper.



(a) No Punishment of Trajectory Length and Jerkiness



(b) Trajectory Length and Jerkiness Punishment

Figure 4.2.: Sampled 3D trajectories with and without Length and Jerkiness Punishment for a 7D Problem on 3 Via Points

4.3.2 Constraining the Jerkiness and Trajectory Length

For higher dimensions and increasing number of via points(Starting at 6 via points in 3D and one via points in 7D), the learnt trajectories can get long and jerky, as this is not constrained in the reward function(Equation 2.6). Such a trajectory is shown in Figure 4.2(a). To fix this problem, the reward function can be adjusted s.t. it punishes the total jerkiness and length of the sampled trajectories. This yields

$$R = exp(-\beta_{policy} - \beta_1 length - \beta_2 jerkiness). \tag{4.1}$$

β_1 and β_2 are two new hyperparameters and are used to weight the jerkiness and length. In the 7D learning experiments, a value of 0.15 was used for β_1 and 100 for β_2. The jerkiness and length of a trajectory can be computed via finite differences. However, this means a longer computation time, but the results in Figure 4.2(b) show that this is an acceptable tradeoff. The computation time was 42 seconds for (a) and 63 seconds for (b).

4.4 Simple Grasping Experiment

In this section, a simple object disentangling experiment in gazebo using the left iiwas arm is shown. Figure 4.3 shows the setup of the experiment. The goal of the left iiwas arm is to go near the entangled s- and u-shape, starting at the position where each joint angle is set to 0. Then, the robot has to grasp the s-shape at its upper end and take it to a point in task space which is behind the u-shape and to the left of the table.
Regarding this information, 3 via points can be defined: The starting position, the grasping position for the end-effector near the upper end of the s-shape and the end position to the left of the table and behind the u-shape. For this experiment, the via points are hardcoded in 7D in order to avoid explicit mapping between 7D and 3D during the computation. However, the via points should be defined in 3D in a real-world experiment, because it is hard and time-consuming to find the 7D configuration corresponding to a 3D via point by just moving the robot arm around. Figure 4.4 shows the initial demonstrations and 3D points corresponding to the defined via points in a 3D coordinate system.

4.4.1 Learning for 3 Via Points

A first problem occurs when trying to learn a trajectory which passes through all via points. Results show that the relevance coefficients for the 2nd and 3rd via point overlap for each DOF. This is illustrated in Figure 4.5(a). This prevents the robot from learning: The collected reward is too small and does not converge (Figure 4.5(b)). The distances to the 2nd and 3rd via points also can not be minimized (Figure 4.5(c),(d)). The sampled trajectories after RWPO are shown in Figure 4.5(e). It gets clear that the algorithm can not decide through which via point to go, because the same weights are responsible for minimizing the distance to the 2nd and 3rd via points.

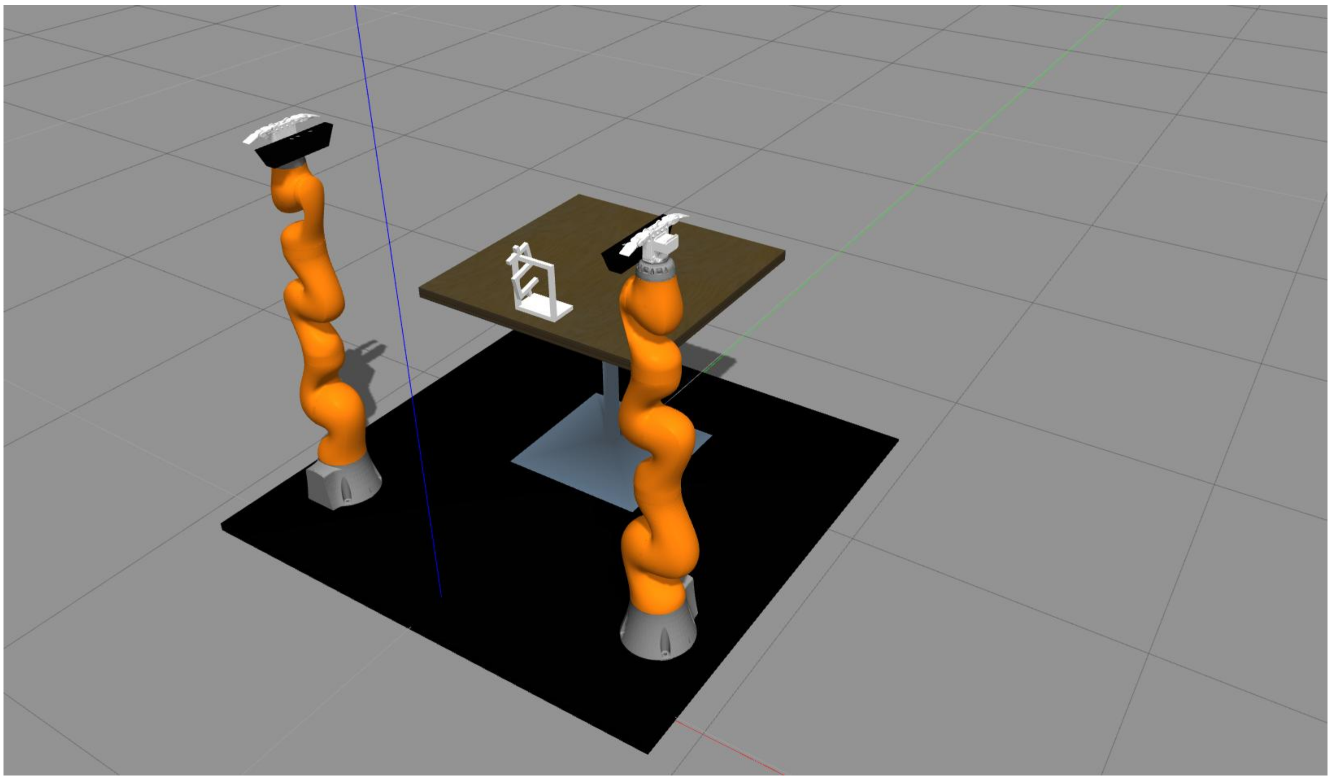


Figure 4.3.: Simplified Simulation Setup

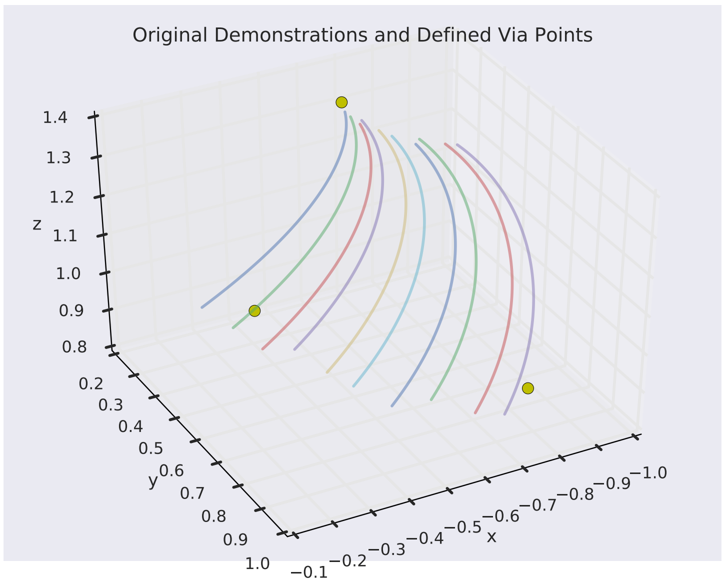
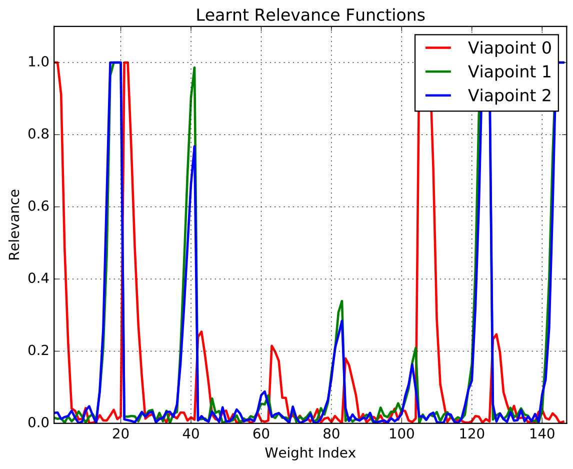
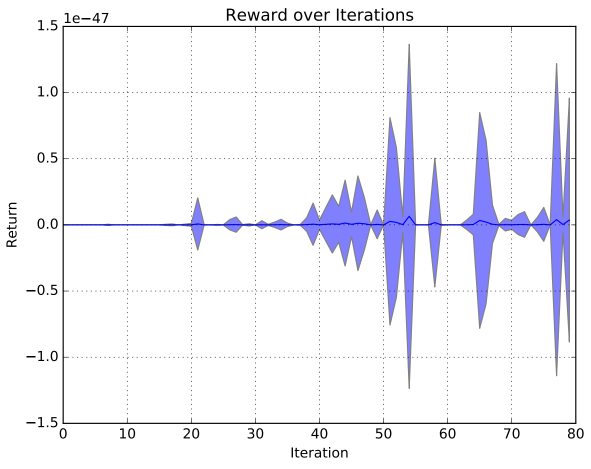


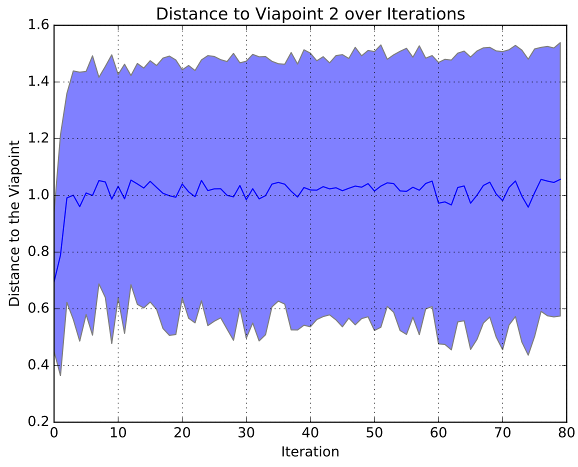
Figure 4.4.: Initial Demonstrations and Defined Via Points



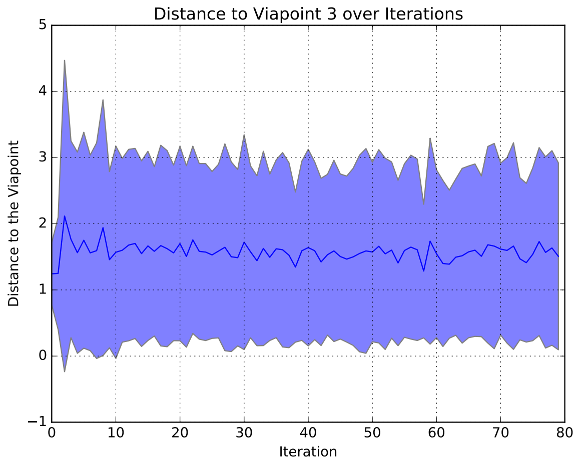
(a) Learnt Relevances



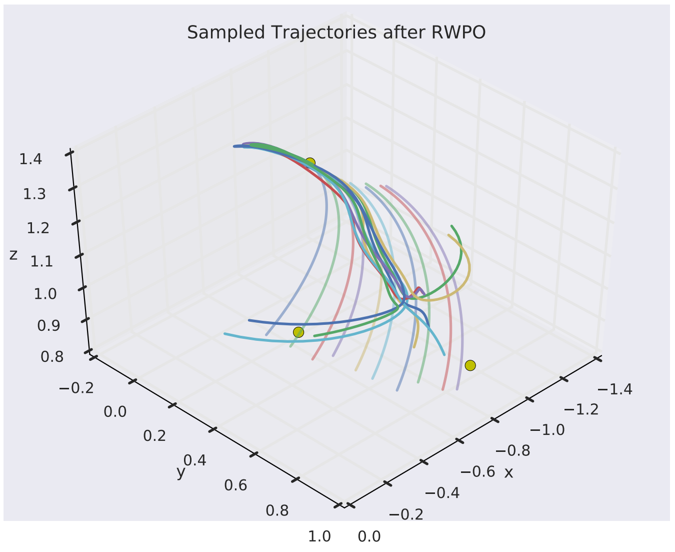
(b) Reward over Iterations



(c) Distance to the 2nd Via Point over Iterations

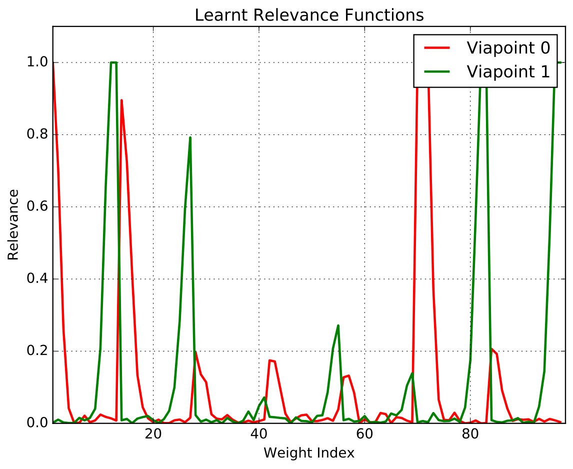


(d) Distance to the 3rd Via Point over Iterations

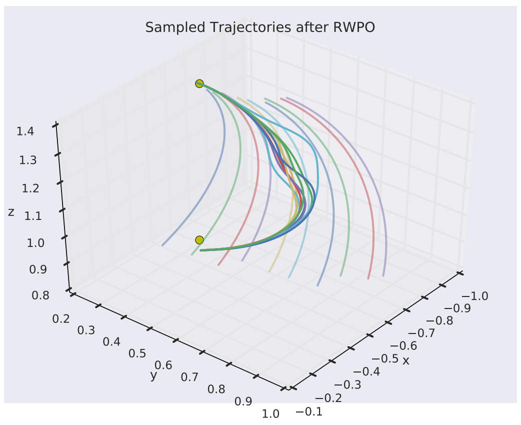


(e) Sampled Trajectories after RWPO

Figure 4.5.: Illustration of Results of 7D learning on 3 Via Points During the Gazebo Experiments

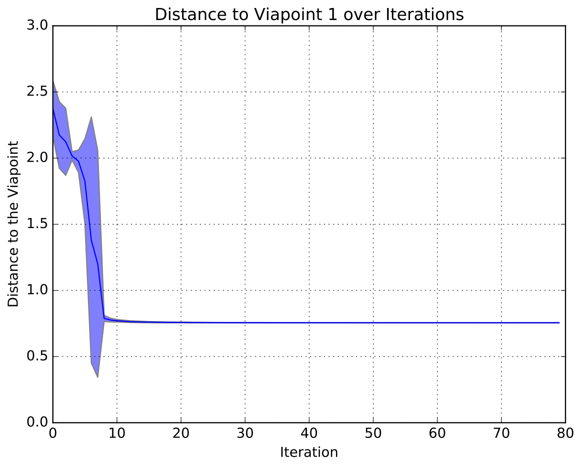


(a) Learnt Relevances for the 1st Sub-Task

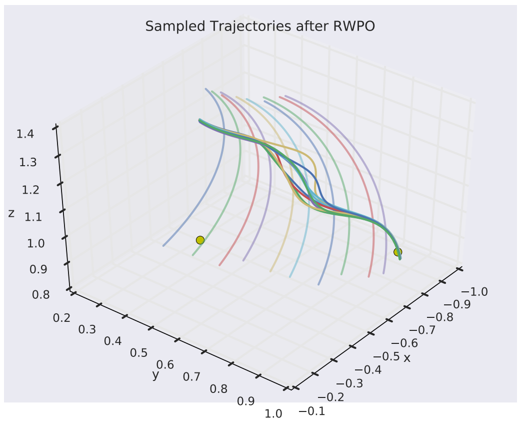


(b) Sampled Trajectories after RWPO for the 1st Sub-Task

Figure 4.6.: Results of Learning for 2 Via Points; First Sub-Task



(a) Distance to the 1st Via Point over Iterations for the 2nd Sub-Task

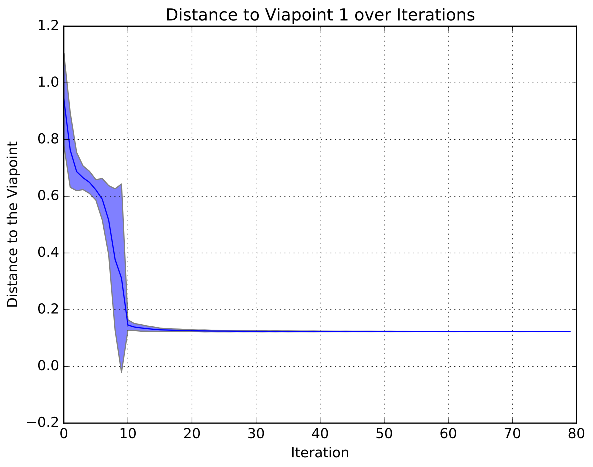


(b) Sampled Trajectories after RWPO for the 2nd Sub-Task

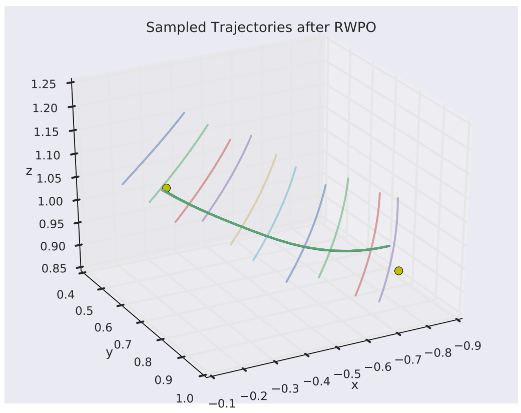
Figure 4.7.: Results of Learning for 2 Via Points; Second Sub-Task

4.4.2 Learning two Times for 2 Via Points

An idea on how to solve the previous problem is to split it into two tasks: The first task is to reach the s-shape, while starting at the joint-angle-at-zero position. The second task would be to reach the final position, starting at the position of the s-shape. Thus, the weighting dependancy between the 2nd and 3rd via point can be solved.
The first task is easy to solve. The robot is able to reach the s-shape after about 45 seconds of learning. The learnt relevances and sampled trajectories after RWPO are illustrated in Figure 4.6(a),(b). The results show that the learning of the first task has succeeded.
Solving the second task is a bit harder, since the starting position is the position of the s-shape, while on the other hand, the demonstrations start near the zero configuration. Figure 4.7(a) shows that the distance to the 1st via point can not be reduced below 0.7. In other words, the algorithm "prefers" the demonstrations during learning. Figure 4.7(b) shows the sampled trajectories. The robot is unable to start directly at the position of the s-shape.
The problem described in the previous paragraph can be solved by a simple trick: First, the point on the demonstrations which is closest to the joint configuration needed to reach the s-shape is computed. After that, the demonstrations can be "cut" s.t. they now start a bit before that point. That way, the demonstrations are more helpful for the algorithm in order to reach the s-shape. Figure 4.8(a) shows that the distance to the 1st via point can now be reduced way below 0.7. Figure 4.8(b) shows the sampled trajectories after RWPO together with the cut demonstrations.



(a) Distance to the 1st Via Point over Iterations for the 2nd Sub-Task



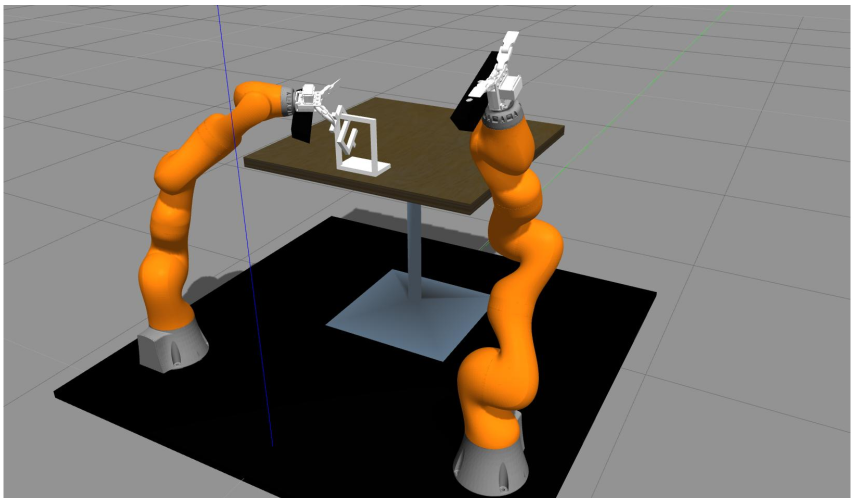
(b) Sampled Trajectories after RWPO for the 2nd Sub-Task

Figure 4.8.: Results of Learning for 2 Via Points; Second Sub-Task; Cut Demonstrations

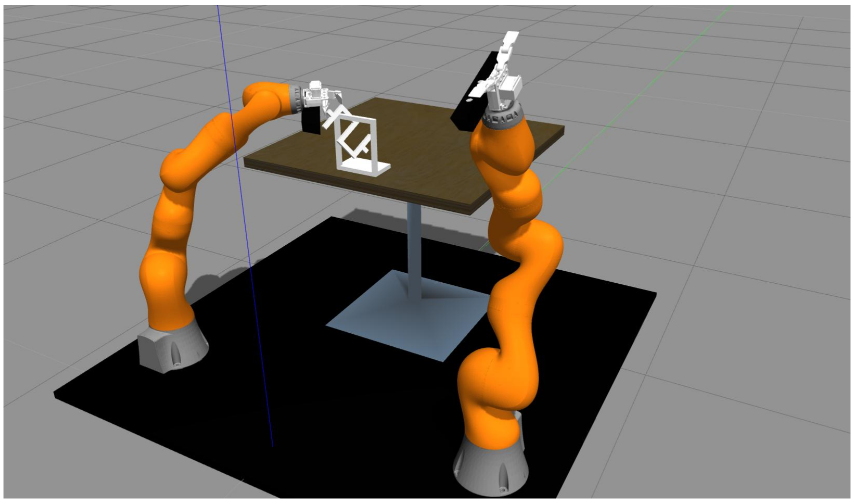
4.4.3 Object Grasping and Disentangling

Running the learnt trajectories in simulation shows that the robot is able to reach each via point. However, the grasping is hard and does not succeed everytime, as the accuracy varies for each learning process. Figure 4.9(a) shows the robot reaching for the s-shape. However, this grasping attempt failed as shown in Figure 4.9(b). Figure 4.9(c) shows the robot without the s-shape at the final position.

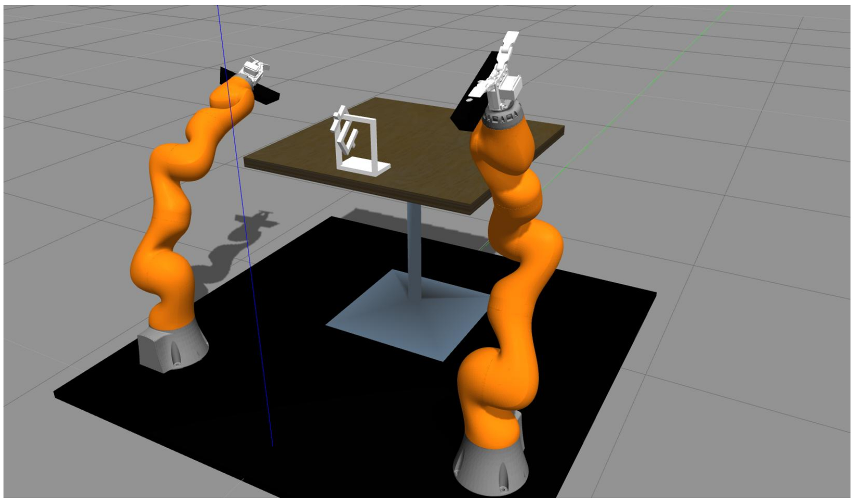
It is hard for the robot to grasp an object and disentangle it from another object alone, because the learning algorithm provides results with a varying accuracy. This problem can be solved by letting a human teleoperate the robot. This way, the robot can guide the human along the learnt optimal trajectory. When the position of the s-shape is reached, the human can apply force to the end-effector in such a way that he manages to grasp the s-shape and disentangle it. Once the robot has grasped the object, the human can teleoperate it further along the learnt trajectory towards the last via point. This procedure can be repeated several times, while recording each attempt. The robot can then learn a distribution along those adjusted trajectories in order to be able to do the object disentangling task more accurately and without failing. This procedure is a joint learning between the human and the robot, as they help each other along the way towards the optimal solution.



(a) The Robot Reaches for the S-shape



(b) The Robot Misses the S-shape



(c) The Robot at the Final Position

Figure 4.9.: Object Grasping and Disentangling Experiments with the Left Iiwas Arm in Gazebo

5 Experiments with Haption Virtuose 6D

5.1 Experimental Setup

This chapter describes an experiment with the Haption Virtuose 6D device. The device is shown in Figure 2.4. The experiment is similar to the one shown in Chapter 6.2 of [1], but with raised complexity.

5.1.1 Virtual Environment

The experiment has the same prerequisites as the haptic feedback experiment shown in [1]. The simulation environment is ROS Visualization (rviz). The setup consists of a starting and an end point and a wall which has a window. The starting point is before and the end point behind the wall. There also is a virtual stick, which is attached at the end-effector of the Haption Virtuose 6D Device.

5.1.2 Definition of the Problem

In the following experiment, a user has to teleoperate Haption device in order to bring the stick from the starting point to the end point, passing through the window with a certain orientation. In order to make the task easier for a human, the Haption device should guide the user by providing force feedback. The force feedback is generated the same way as in the experiment conducted in [1].
The idea is to optimize initial demonstrations w.r.t. to the starting and end position, the position of the window and the desired orientation of the stick. RWPO is going to be used for the trajectory optimization. The experiment advances in three ways compared to the haptic feedback conducted by Ewerton et al. [1]. First, the position of the window can be generated randomly, while Ewerton et al. only used a fixed window. Second, the trajectories are going to be optimized w.r.t the orientation of the bar, additionally to the position. This is done in order to obtain an orientation s.t. the bar can pass through the window without hitting the wall. Third, the number and dimensionality of via points are increased to four and 6D, compared to three and 3D for the experiment of Ewerton et al. [1]. The second and third via points are the position and the orientation of the bar a bit before and after the window. This has to be done in order to ensure that the stick is turned into the right orientation before and back to the initial orientation after the window in order to avoid collisions with the wall due to changing the orientation too late (before) or too soon (after).

5.2 Implementation and Results

Since the orientation of the stick has to be optimized additionally to the position, the via points become 6 dimensional: The orientation can be expressed as a quaternion. A quaternion has 4 dimensions: x,y,z and w. x,y and z are imaginary parts and encode the rotation axis. w encodes the rotation along this axis [21]. Quaternions are normalized, i.e $x^2 + y^2 + z^2 + w^2 = 1$. That means that w can be computed directly if x,y and z are given: $w = \sqrt{1 - x^2 - y^2 - z^2}$ [22].
In order to solve the task for different window positions using RWPO, the demonstrations have to explore the task space. The demonstrations used for the following experiments, together with four randomly generated Via Points, are shown in Figure 5.1. Note that the demonstrations were recorded in 6D, thus containing position and the x,y and z part of the orientation quaternion of the end-effector. The plot shows only the position.
In the following experiments, the length and jerkiness of the trajectories are punished in the reward function. The relevances are represented by rescaled PCCs. A first idea for solving the task defined above is to optimize a ProMP for each 6D via point simultaneously using RWPO. However, a problem appears: The learnt relevance functions overlap. This is shown in Figure 5.2. For (a) and (b), 30 Gaussian basis functions were used for the ProMP. As seen in (a), the relevances for the 2nd and 3rd via point overlap-the system can not learn how to optimize for both these via points and chooses to optimize only for the 3rd one, as shown in (b). When using 35 Gaussian basis functions, overfitting occurs additionally. The relevances in (c) look similar to the ones in (a), but the sampled trajectories have more variance, while still failing to pass through the 2nd via point. This is shown in (d).

The stated problem can be solved by first optimizing in 3D for the position. Then, a new ProMP in 6D can be learnt

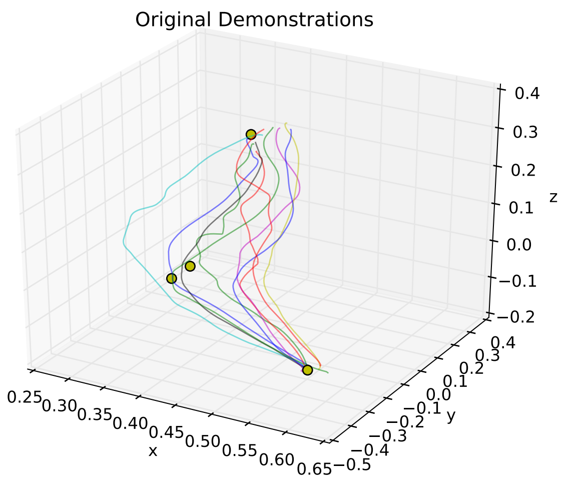
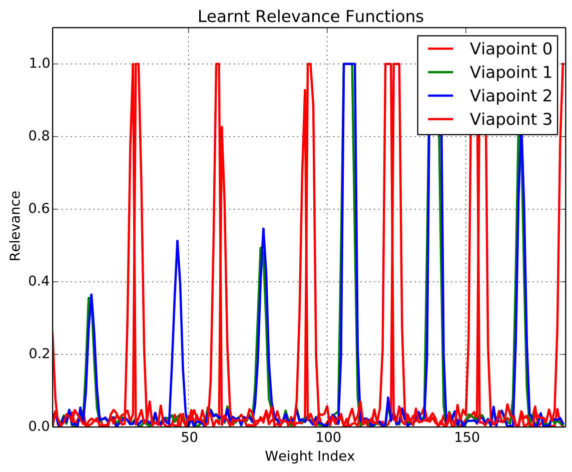


Figure 5.1.: Initial Demonstrations and Four Randomly Generated Via Points

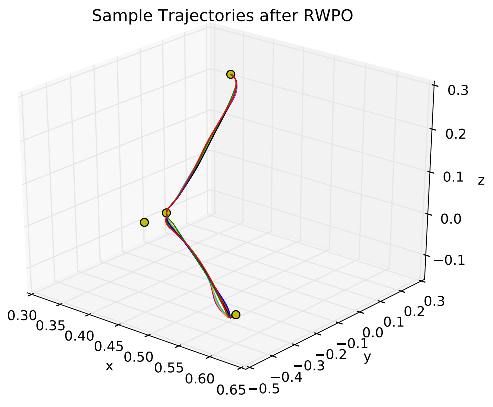
and optimized w.r.t. the orientation. The 6D ProMP has the trajectories optimized w.r.t. the position, combined with the 3D orientation recorded in the demonstrations. In the experiments, 30 Gaussian basis functions were used for the position and 31 for the orientation. The number of basis function correlates with the trajectory length: For the gazebo experiments in the previous chapter, the trajectory length was only 80, while it varies from 400 to 800 in the Haption experiments, depending on the duration of the execution. The learnt relevances w.r.t the position of the end-effector are shown in Figure 5.3(a). It shows that the algorithm can distinguish better between the 2nd and 3rd via point. When optimizing for the position, the indexes of the points which are important for optimizing a trajectory for each via point, are saved. They are used in the next step to ensure that the new trajectories have the desired orientation for each via point. In other words, these indexes couple the position and orientation of the end-effector. Figure 5.3(b) shows the learnt relevances for the orientation. The left half of the figure is "empty", as this part is relevant for optimizing the position, but not the orientation. The sampled trajectories of the position are shown in Figure 5.3(c). Now, the trajectories pass through every via point. As shown in Figure 5.3(d), the orientation of the end-effector after RWPO is also on spot, as it respects the via points and goes through them.

The mean and standard deviation of the optimized ProMP can now be used in order to calculate force feedback for the Haption device, as described in Chapter 4 of [1]. The position on a trajectory is used for computing the force feedback and the next position and orientation of the end-effector are set and displayed in simulation according to the next 6D point on the latent trajectory. Figure 5.4 shows the stick on different positions along the optimized trajectory distribution. It shows that the orientation of the bar is set perpendicularly to the window when the bar is passing through it. Figure 5.5 shows the bar passing through the window from different perspectives to prove that no collisions occur. The window position can be generated randomly, as long as it repects the joint limits of the Haption device and the size of the wall in simulation. Figure 5.6 shows trajectory distributions after RWPO for different window positions.

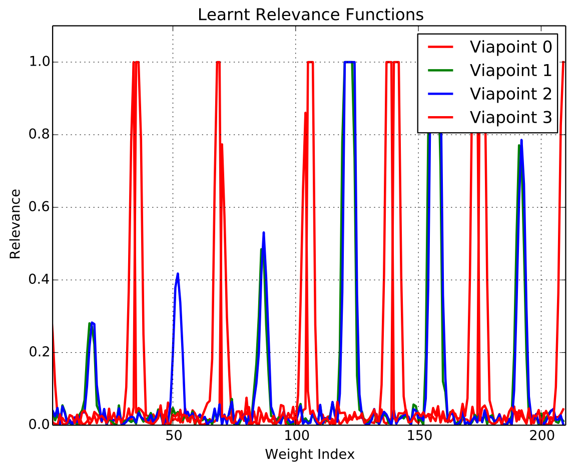
The stated teleoperation task can be solved using RWPO with four via points. However, this solution does not guarantee that no collisions with the wall will occur. For example, if the bar is made longer and the via points stay the same, collisions may occur, as one end of the bar could hit the wall if the rotating motion is executed too late and too close to the window. This can be solved by punishing the number of collisions in the reward function. This requires implementing a collision detection and defining different reward functions for the start, end and window. As the solution with four via points works, the collision punishment approach is not stated in this paper and is considered future work. The via point solution can be also extended in such a way that the algorithm decides where to put the via points depending on extra information like window size or bar length. With that in mind, smart via point selection will also minimize the risk of collisions.



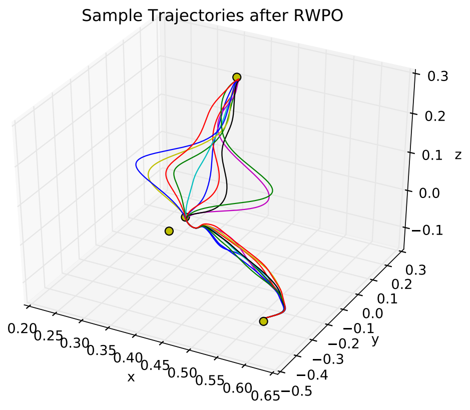
(a) Learnt Relevances Using 30 Gaussian Basis Functions



(b) Sampled Trajectories after RWPO with 30 Gaussian Basis Functions

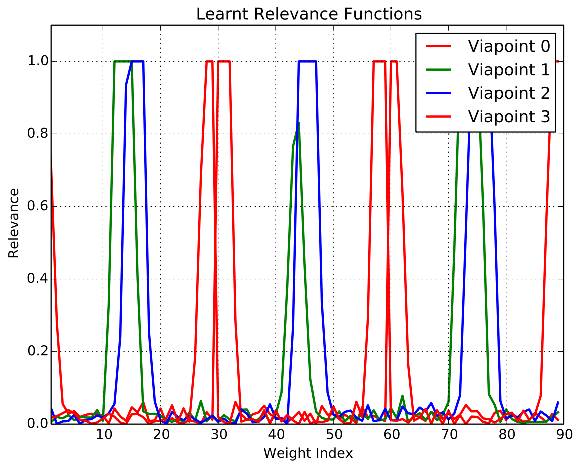


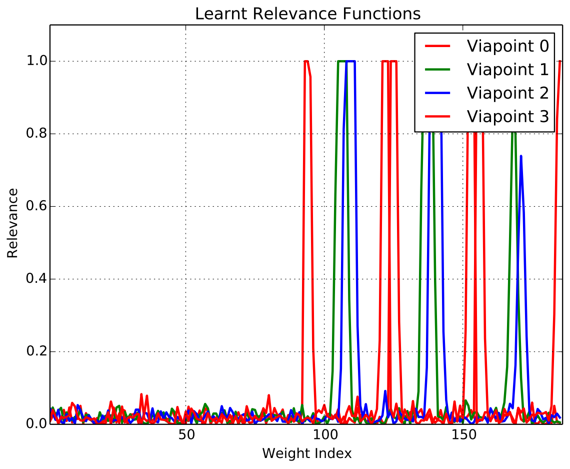
(c) Learnt Relevances Using 35 Gaussian Basis Functions



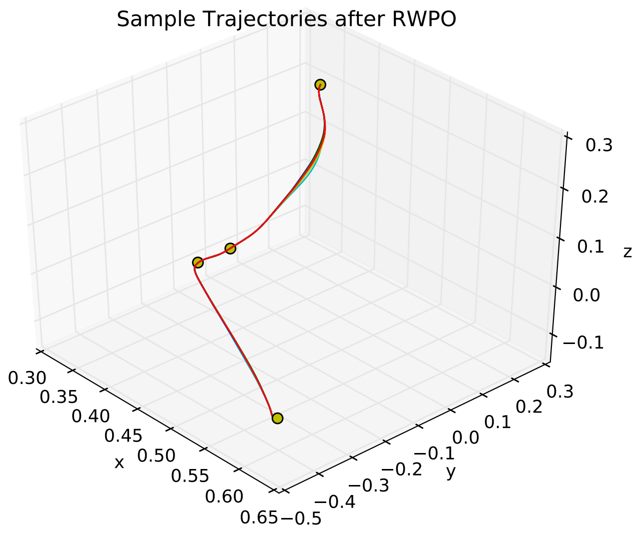
(d) Sampled Trajectories after RWPO with 35 Gaussian Basis Functions

Figure 5.2.: Learnt Relevances and Sampled Trajectories When Optimizing the Position and Orientation Simultaneously

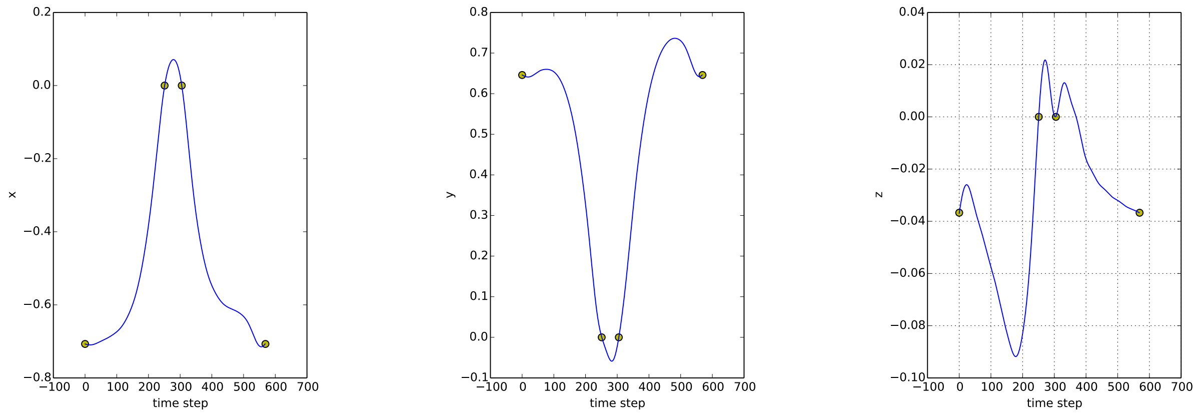




(a) Learnt Relevances for the Position Using 30 Gaussian Basis Functions **(b)** Learnt Relevances for the Orientation Using 31 Gaussian Basis Functions



(c) Position of the Sampled Trajectories after RWPO



(d) Orientation of the Sampled Trajectories after RWPO

Figure 5.3.: Learnt Relevances and Sampled Trajectories When Optimizing the Position and Orientation Subsequently

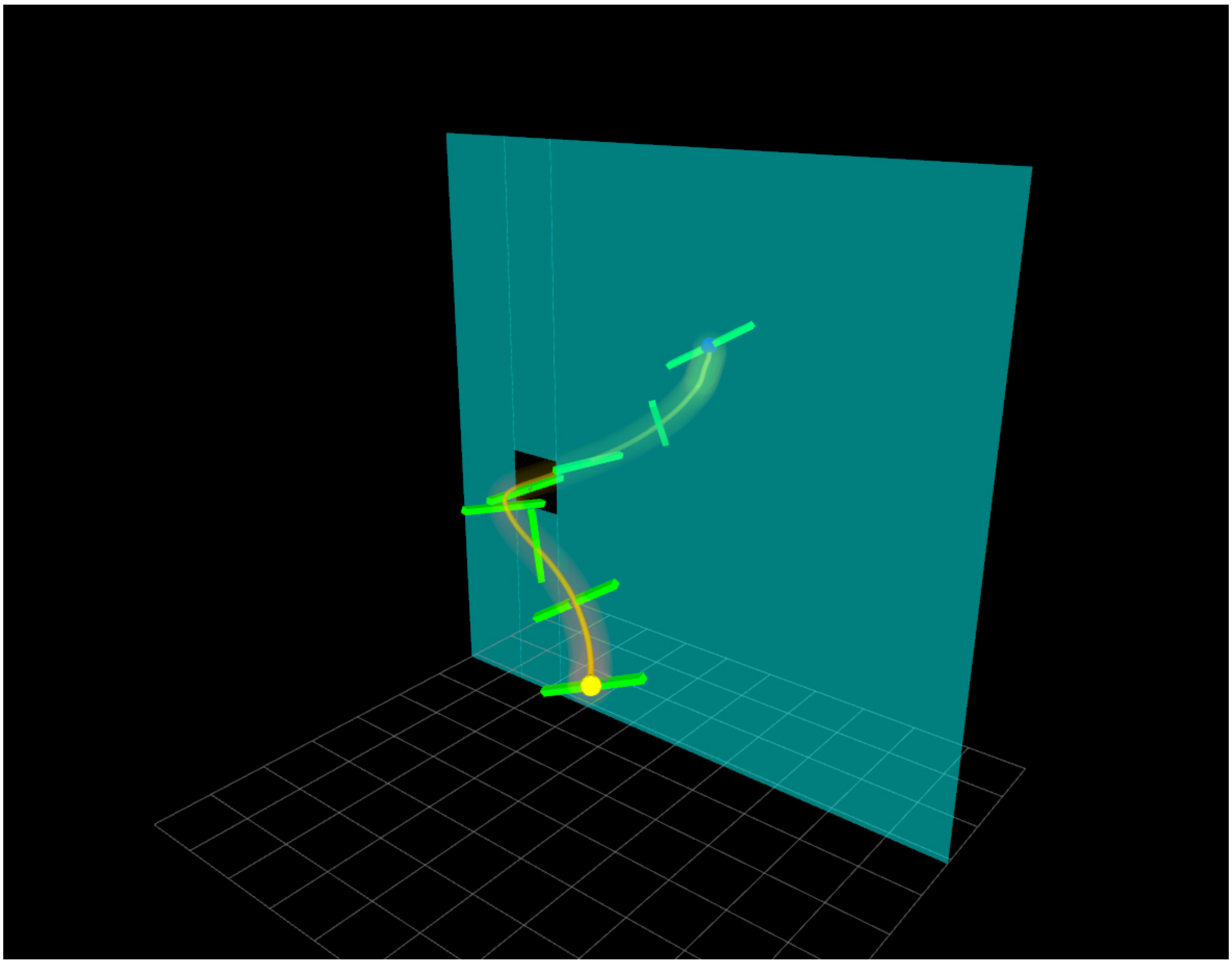
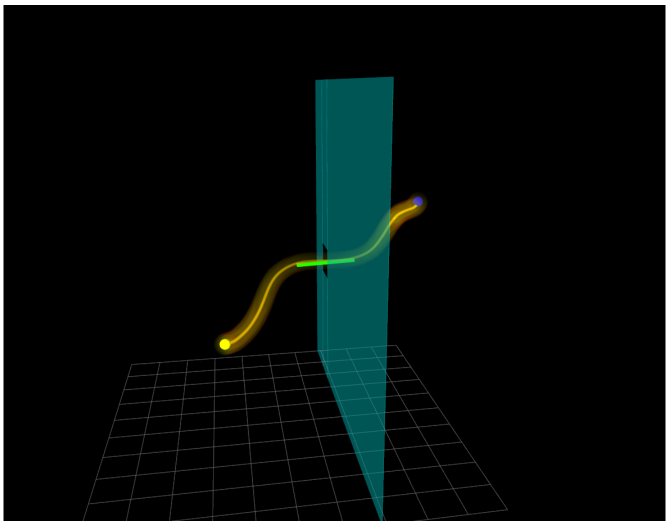
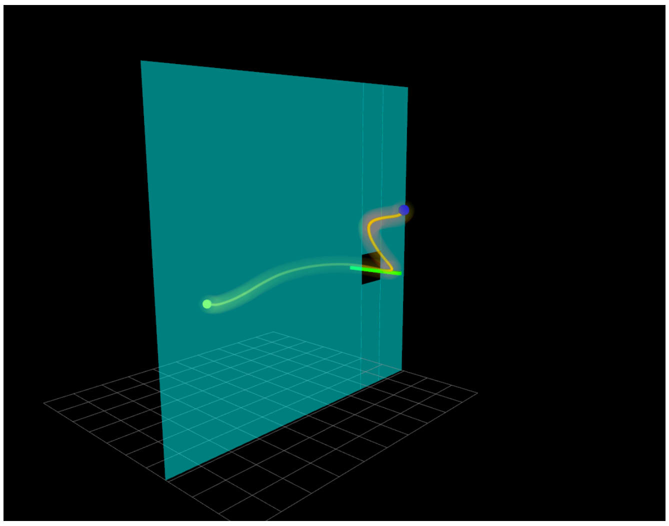


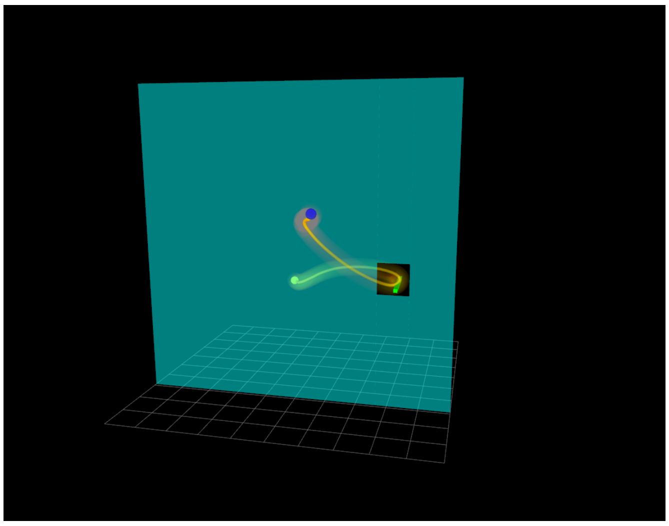
Figure 5.4.: Moving the Bar Along the Optimized Trajectory Distribution



(a) Perspective 1

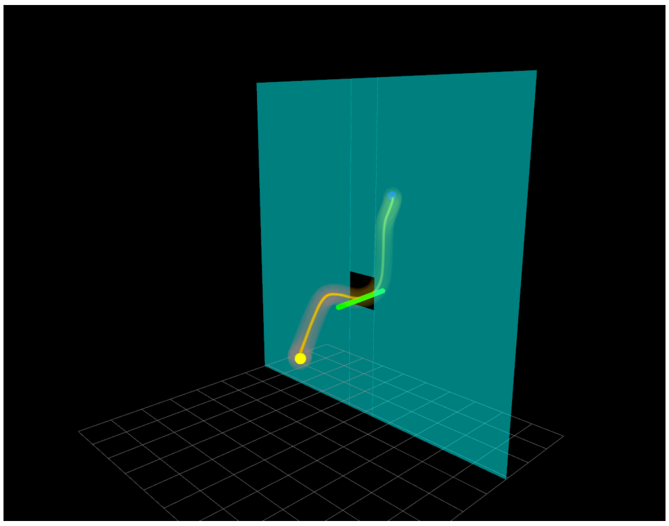


(b) Perspective 2

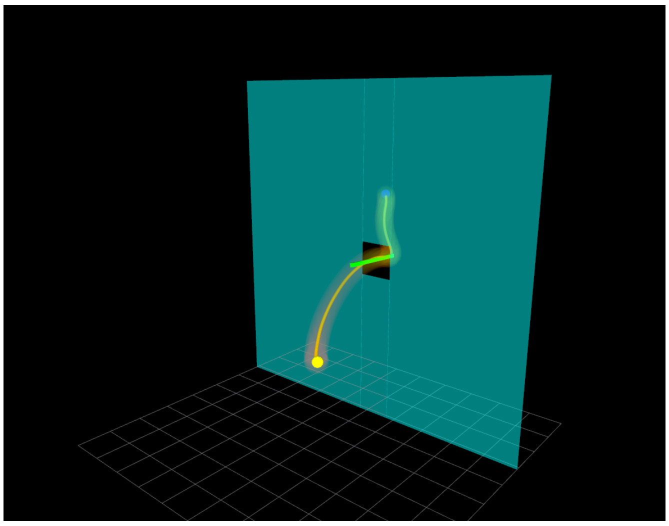


(c) Perspective 3

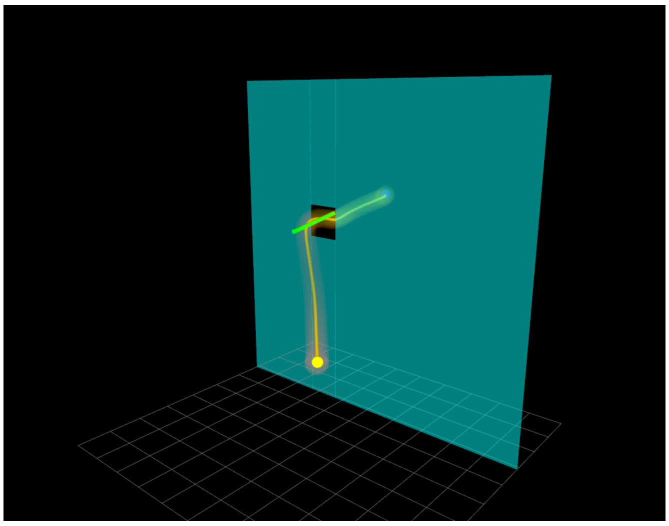
Figure 5.5.: Bar Passing Through the Window Seen from Different Perspectives



(a) Window Position 1



(b) Window Position 2



(c) Window Position 3

Figure 5.6.: Learnt Trajectories for Different Window Positions together with the Bar Passing Through the Window

6 Conclusion and Future Work

RWPO is a reinforcement learning algorithm which can optimize a distribution of trajectories w.r.t. different criteria like trajectory length and jerkiness, but also w.r.t. via points along a trajectory, such that the new trajectories pass through the via points. The trajectory distribution is represented by a ProMP. The mean and covariance of this ProMP are optimized based on a reward function during the policy optimization of RWPO. RWPO uses relevance functions in order to sample smartly. Relevance functions indicate which policy parameters are most relevant for changing a distance between a given point of a trajectory and a certain via point.
This work focused on evaluating RWPO in different setups. It showed that the algorithm is scalable and can be used in different domains. The algorithm was applied in an object disentangling and a teleoperation task.

This work unraveled a problem which occurs when more than 3 via points are defined. For more than three via points, the relevances can not be computed the way shown in [1]. Numerical instabilities occur because the rewards of the other via points are subtracted from the reward of the current via point. This work showed that the problem can be solved by dividing the sum over rewards by the number of via points minus one, if more than 2 via points are used.
Another problem of the relevance computation proposed in [1] is the high computation cost. Therefore, this work introduced a new, efficient way of computing the relevance coefficients. This is done by using Pearson's Correlation Coefficient (PCC). Because the absolute values of PCCs often do not reach values higher than 0.7 in practice, these were rescaled in such a way that values above a certain threshold got 1. The effectivity and efficiency of rescaled PCCs were shown in further experiments.
The algorithm struggles in experiments with via points far from the initial demonstrations. Fine-tuning the hyperparameters may fix the problem. Future work may investigate the role of the hyperparameters further and consider other solutions to this problem.
Another aspect which was not considered in [1] was the length and jerkiness of learnt trajectories. These were punished in the reward function during the experiments shown in this work.

This work showed two applications of RWPO: Object disentangling in gazebo using a 7-DOF robot arm and teleoperation of a bar in rviz using the Haption Virtuose 6D device. For the first experiment, the learning was done in joint space, i.e. in 7D, for up to three via points. For the second experiment, the learning was done in 6D, as the orientation of the bar was also needed to be adjusted in order to pass through a window in a wall and avoid collisions. This work also introduced heuristics for dealing with the different hyperparameters of RWPO.

During the gazebo experiments, 7D via points were used. However, the joint configuration to reach a certain via point defined in 3D is often not known in practice. Future work is going to investigate how to learn a joint configuration which belongs to a 3D via point. An idea is to use tf, a ROS framework, or efficient C implementations for mapping trajectories from 7D into 3D.
Learning in 7D is not easy, because there are redundancies regarding the task of reaching a certain position in task space. An idea to optimize the learnt trajectories even more is to let a human teleoperate the robot arm along the learnt trajectories and correct parts of the trajectories which are not optimal, like the grasping of an object. These new demonstrations can then be recorded and then optimized again by the robot, until sufficiently good trajectories are found.
Future work regarding teleoperation will be investigating how to adjust the force feedback in such a way that a human user can teleoperate the robot the easiest and most comfortable way possible. Moreover, the robot should be compliant to the human if it is not certain on a different part of a trajectory, and should let the human provide corrections.

To sum up, this work unraveled different sub-optimal aspects of RWPO and fixed them. It then showed that RWPO can be applied in different tasks in the domains of motion planning, teleoperation and object disentangling.

Bibliography

[1] M. Ewerton, D. Rother, J. Weimar, G. Kollegger, J. Wiemeyer, J. Peters, and G. Maeda, “Assisting movement training and execution with visual and haptic feedback,” *Frontiers in Neurorobotics*, 2018.

[2] J. Peters and S. Schaal, “Using reward-weighted regression for reinforcement learning of task space control,” *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.

[3] J. van den Berg, S. Miller, D. Duckworth, H. Hu, A. Wan, X.-Y. Fu, K. Goldberg, and P. Abbeel, “Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations,” *2010 IEEE International Conference on Robotics and Automation*, 2010.

[4] A. Coates, P. Abbeel, and A. Y. Ng, “Learning for control from multiple demonstrations,” *ICML ’08 Proceedings of the 25th international conference on Machine learning*, pp. 144–151, 2008.

[5] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Probabilistic movement primitives,” *Advances in Neural Information Processing Systems 26*, pp. 2616–2624, 2013.

[6] S. Lichiardopol, “A survey on teleoperation,” *DCT report*, 2007.

[7] I. Havoutis and S. Calinon, “Learning assistive teleoperation behaviors from demonstration,” *HRI ’16 The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, pp. 59–66, 2016.

[8] M. J. A. Zeestraten, I. Havoutis, and S. Calinon, “Programming by demonstration for shared control with an application in teleoperation,” *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1848–1855, 2018.

[9] M. J. A. Zeestraten, I. Havoutis, S. Calinon, and D. G. Caldwell, “Learning task-space synergies using riemannian geometry,” *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada*, pp. 73–78, 2017.

[10] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, “Chomp: Gradient optimization techniques for efficient motion planning,” *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[11] J. A. Bagnell and J. Schneider, “Covariant policy search,” *Proceeding of the International Joint Conference on Artifical Intelligence*, 2003.

[12] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, “Stomp: Stochastic trajectory optimization for motion planning,” *2011 IEEE International Conference on Robotics and Automation*, 2011.

[13] P. Englert, “Locally weighted learning,” *Seminar Thesis, Proceedings of the Autonomous Learning Systems Seminar*, 2012.

[14] S. Schaal, C. G. Atkeson, and S. Vijayakumar, “Real-time robot learning with locally weighted statistical learning,” *International Conference on Robotics and Automation (ICRA2000)*, vol. 1, pp. 288–293, 2000.

[15] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Using probabilistic movement primitives in robotics,” *Advances in Neural Information Processing Systems 26*, pp. 2616–2624, 2013.

[16] http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1, “Beginner: Overview,” Last access on August 14, 2018.

[17] http://wiki.ros.org/ROS/Introduction, “What is ros?,” Last access on August 14, 2018.

[18] http://wiki.ros.org/ROS/Tutorials, “Ros tutorials,” Last access on August 14, 2018.

[19] http://moveit.ros.org/, “Moveit!,” Last access on August 4, 2018.

[20] T. Foote, “tf: The transform library,” *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, 2013.

[21] https://answers.unity.com/questions/147712/what-is-affected-by-the-w-in quaternionxyzw.html, “What is affected by the w in quaternion(x,y,z,w)?,” Last access on August 29, 2018.

[22] http://wiki.alioth.net/index.php/Quaternion, “Quaternion,” Last access on August 29, 2018.

[23] https://www.spss-tutorials.com/pearson-correlation coefficient/, “Pearson correlations – quick introduction,” Last access on July 28, 2018.

A Appendix

Specifications of the computer used for all experiments in this report except the Haption Virtuose 6D experiments in Chapter 5:

Model	Acer Aspire E15
CPU	Intel Core i5-6200U
GPU	NVIDIA GeForce GTX 950M with 2GB Dedicated VRAM
RAM	8GB DDR4
Disk	1000GB HDD+96GB SSD
OS	Linux Ubuntu 16.04 LTS 64bit

Specifications of the computer used for the Haption Virtuose 6D experiments in Chapter 5:

CPU	Intel Core i7-4790K CPU @ 4.00GHz x 8
GPU	GeForce GTX 970/PCIe/SSE2
RAM	31.3 GiB
Disk	787.0 GB
OS	Linux Ubuntu 14.04 LTS 64bit