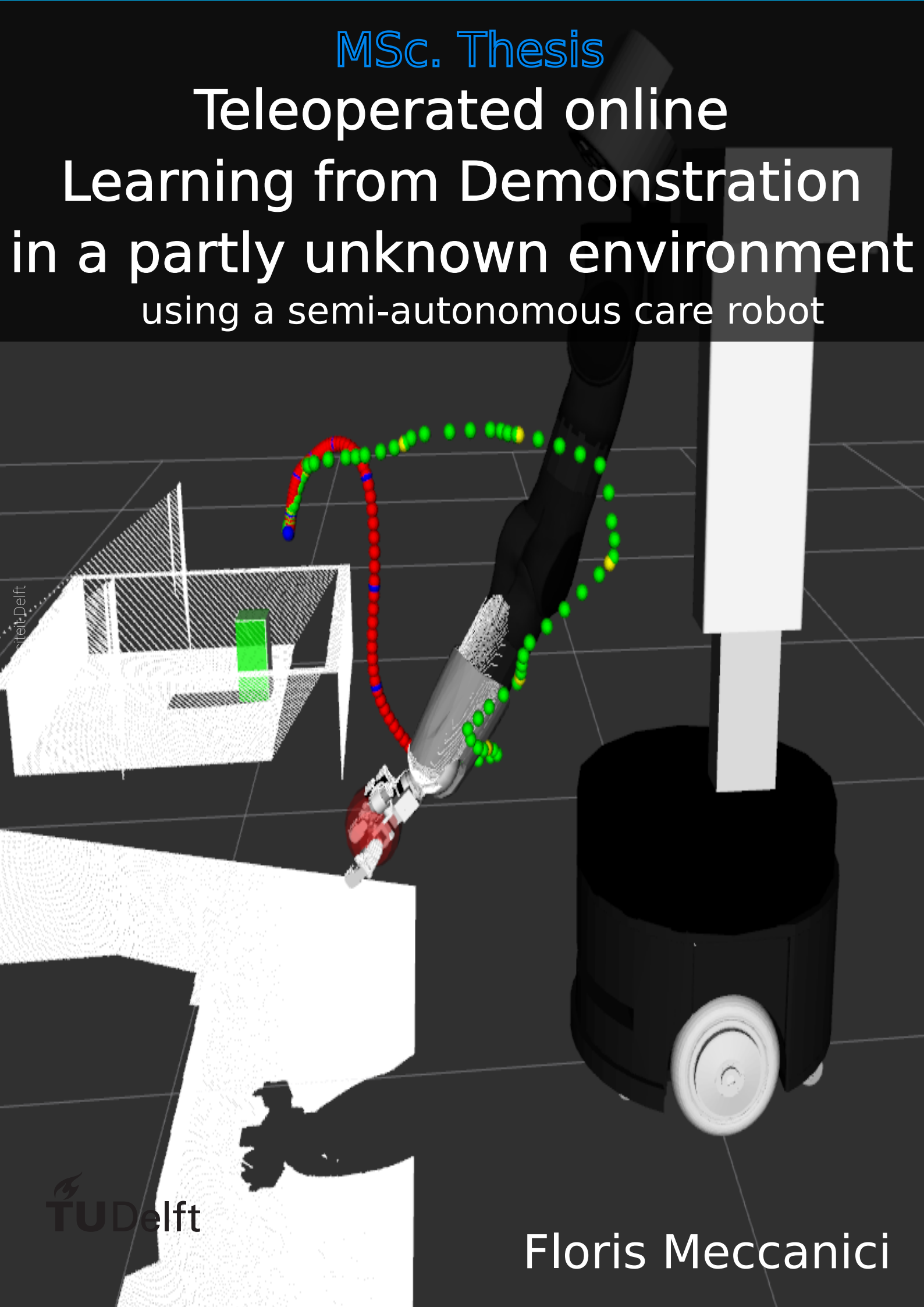
MSc. Thesis

Teleoperated online Learning from Demonstration in a partly unknown environment

using a semi-autonomous care robot



TU Delft

Floris Meccanici

Teleoperated online Learning from Demonstration in a partly unknown environment

using a semi-autonomous care robot

by

F. Meccanici

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday February 18, 2021 at 13:00.

Student number:	4225201	
Project duration:	September 1, 2020 – February 18, 2021	
Thesis committee:	Prof. dr. ir. D. Abbink,	TU Delft, supervisor
	Dr. ir. L. Peternel,	TU Delft, supervisor
	Dr. ir. C. Heemskerk,	Heemskerk Innovative Technology, supervisor
	Ir. D. Karageorgos,	Heemskerk Innovative Technology, supervisor

This thesis is confidential and cannot be made public until February 18, 2021.

TU Delft

Preface

The general approach in creating motions for a pick-and-place task is to use a path planner, which relies on a completely *known* environment to create a collision free motion that reaches the goal pose. During my internship at Heemskerk Innovative Technology I worked on automating a pick-and-place task on a care robot and found out that when the environment is partly *unknown* due to inaccurate object detection, the path planner can fail even a very basic task like reaching for a box, either due to a goal deviation or because collisions occur with an unforeseen obstacle:

Incorrect goal position

Unforeseen obstacle

This problem triggered my research in using Learning from Demonstration to teach the robot a trajectory model that takes into account these *unknown* parts of the environment, and that allows to adapt this model when the reaching motion fails. In my literature research I investigated different methods to create and adapt such model, in which I found that these methods mainly focus on physically moving the end effector and that literature was missing on using teleoperation. In my thesis I filled this gap by developing a teleoperated method for creating and adapting a trajectory model on a care robot.

I want to thank my supervisors from the TU Delft, Luka Peternel and David Abbink for their critical point of view and thorough feedback, as well as their positivity and enthusiasm which motivated me to continue and finish my work. Furthermore I would like to thank my supervisors Dimitrios Karageorgos, Cock Heemskerk and all others at HIT for their valuable input, support and optimism, which helped me a lot to finish my work. I had a great time, learned a lot and am grateful that I had the opportunity to graduate at HIT.

F. Meccanici
Delft, February 2021

Contents

1

Scientific paper

Teleoperated online Learning from Demonstration in a partly unknown environment using a semi-autonomous care robot

Floris Meccanici[1,2]
Supervised by: Luka Peternel[1], Dimitrios Karageorgos[2], David Abbink[1], Cock Heemskerk[2]

***Abstract*— The general approach to generate collision free motion in a constraint environment is to use path planners, which demand a known environment and potentially fail otherwise. Learning from Demonstration (LfD) can be used instead to teach the robot unknown parts of the environment, such as a goal deviation or an unforeseen obstacle. The general approach is to train a model offline and expect it to perform well afterwards. Problems arise however when the model is trained insufficiently or unknown variations have occurred in the environment, which demand for refinement of the model. In online learning the operator is allowed to refine a predicted trajectory during execution time, where the state-of-the-art methods focus on kinaesthetic teaching. The contribution of this research is the development of a teleoperated online learning method, where the operator can make refinements by moving a haptic stylus (Phantom Omni) in the desired direction. By doing this, a force is felt proportional to the magnitude of refinement. After creating such a refined trajectory, it is used to update a condition dependent probabilistic trajectory model. The proof of concept was shown on a 2D example and on a simulated robot that shows that we can adapt an initial model when unknown variations occur and that the method is able to deal with different object positions and initial end effector poses. To show if other people can use the method, a human factors experiment is performed, comparing the developed method against three other methods on how much time it takes to successfully adapt a model (refinement time) and on the perceived workload. Two different parameters are varied, which are the teaching device (stylus or keyboard) and the learning mechanism (online or offline). The expectation was that both online with stylus has the lowest refinement time and workload, but the results show that only online has a significant improvement over offline methods ($p = 7.94 \times 10^{-12}$ and $p = 0.000512$ respectively). This is explained by the fact that only small corrections have to be made and in a maximum of three degrees of freedom (DoFs). No significant difference was found between keyboard and stylus ($p = 0.755$ and $p = 0.302$ respectively). An explanation for this is that this is task, person and implementation dependent. The recommendations are to evaluate the proof of concept on the real robot and to extend the method with orientation refinement, such that more complex tasks can also be dealt with. We hypothesize that with these tasks the combination of online with stylus does perform the best.**

I. Introduction

The amount of people over 65 years old in the US only has increased from 3.1 million in 1900 to 56 million in 2020, and is expected to increase to 88.5 million by 2050, constituting 20% of the US population [1]. With such an increasingly aging society, the need for health care is rising [2]. To account for this need, more people need to work within the health care sector, increasing the health care cost and when this demand is not filled, the quality of care will decrease. To fill this gap, Heemskerk Innovative Technology (HIT) build a care robot, which is aimed to alleviate the workload of the care workers by performing Activity of Daily Living (ADL) tasks. This increases the quality of care by enabling for more client centered care while improving client autonomy, since 83% of US care-home residents need assistance with three to six basic ADLs [2]. Examples of such ADL tasks are preparing meals, table laying/clearing and loading/unloading the dishwasher [3]. Currently the robot is able to perform ADL tasks via *teleoperation*, which requires constant human attention, thus imposing high workload on the operator. Additionally when multiple robots are deployed, multiple operators are needed to control them. These factors limit the economic feasibility and practical usefulness of the robot. The solution for this is to increase the autonomy of the robot and since a large part of ADL tasks consist of pick-and-placing, the focus of this research was on a sub-task of this: reaching motions.

A. Problem definition

The general approach of solving pick-and-place tasks is to detect objects, determine the goal pose and plan a collision free path towards this goal [4]. Problems arise however with motion planning in uncertain and unstructured environments [5], where only partial knowledge of the surroundings is available. Motion planners assume that the environment is perfectly known and that it remains static [4], which is violated when partial knowledge of the environment is available, for example due to sensor inaccuracies or an unforeseen obstacle. Within such partially unknown environments, motion planners can fail [5], as illustrated in Fig. 1.

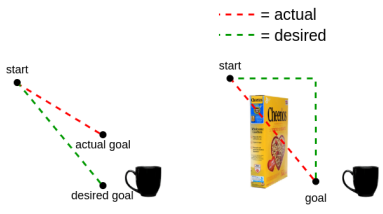


Fig. 1. Left: Goal deviation due to sensor inaccuracies, Right: Unforeseen obstacle

Learning from Demonstration (LfD) or imitation learning can be used instead to teach these unknown parts of the

[1]Cognitive Robotics departments, Faculty of Mechanical, Maritime and Materials Engineering, Delft University of Technology, Mekelweg 2, 2628CD Delft, The Netherlands
[2]Heemskerk Innovative Technology B.V., Mijnbouwstraat 120, 2628RX Delft, The Netherlands

environment to the robot by relying on the presence of a human operator [5]. Trajectory-level LfD is used to teach a low level policy or mapping function between states and corresponding actions [6], where the state in this application is environmental information and the action is the end effector motion of the robot. Trajectory-level LfD can be divided into probabilistic and deterministic learning [6]. In deterministic learning, such as Dynamic Movement Primitives (DMP) [7], the demonstrations are modeled using a second order differential equation with input the goal position and output the end effector position/orientation. Due to this deterministic modeling only one demonstration is needed, and it is guaranteed to converge towards the goal position. However no generalizations can be made towards other environmental inputs than the goal position, such as object or obstacle geometry information and the goal position has to be known beforehand. Also, because no variance is encoded in the demonstrations, constraints in the environment will not be generalized correctly, of which an illustration is depicted in the top images of Fig. 2.

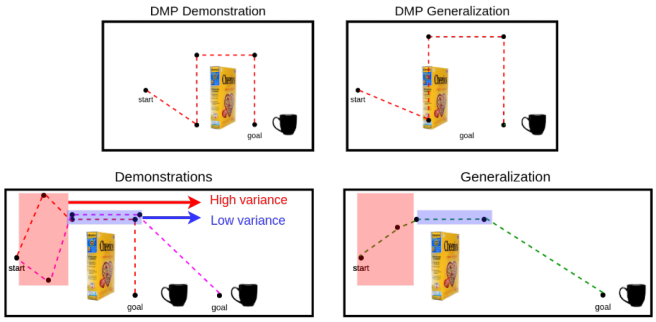


Fig. 2. Top and bottom images show deterministic and probabilistic encoding respectively

In probabilistic learning on the other hand, a probability distribution is build from the demonstrations. This does allow for encoding the variance of the demonstrations, and thus enables the generalization of environmental constraints, of which an illustration is depicted in the bottom images of Fig. 2. The disadvantage is that to achieve the encoding of the variance, multiple demonstrations are needed. Moreover the probability distribution can be conditioned on any known environmental state, for example the object position, weight or size and positions and orientations of obstacles in the workspace [8]. Since we want to generalize towards known environmental variations as much as possible and to generalize unknown constraints in the environment, probabilistic learning is better suited than deterministic learning.

The general LfD approach is to train a model offline and expect it to perform well afterwards. Problems arise however when the trained model is incorrect, either due to insufficient training or unknown variations that have occured in the environment. This demands for methods that are able to refine the initially trained model, of which the state-of-the-art can be divided into online [8][9][10][11] and active learning [5][12]. In online learning the operator is able to refine the motion during execution time, while active learning uses an uncertainty measurement to query the operator for a new demonstration. Since we want to adapt the model towards *unknown* changes in the environment, and since active learning needs to encode the environment to determine the uncertainty in the prediction, these methods are not suited.
In online learning the operator has the role to intervene when the model needs to be adapted, and is therefore better suited in an *unknown* environment. State-of-the-art online learning methods mainly use kinaesthetic teaching (physically moving the end effector) to refine the motion [8][9][11], and teleoperation is limited to [10].

Teleoperated online learning is preferred because it does not require an operator to be constantly physically present, improves the safety of the operator and allows for teaching multiple robots simultaneously. In [10] the variance of the previous demonstrations is used to determine the arbitrage function between automation and the human, also called the level of automation. When the amount of previous demonstrations is high, the amount of motion adaptation the human can perform is low. Since we want to always be able to quickly adapt the motion to deal with unknown parts of the environment, this method is less suited. In [9] and [11] the stiffness of the end effector is dependent on the external forces applied by the operator, and therefore allows for quick adaptation of the executed trajectory. However applying external forces is not possible in a teleoperated setting, and therefore these methods are not suited. An alternative solution is performed in [8], where the end effector has a constant low stiffness, which always allows for quick adaptation of the motion. A teleoperated online learning technique that allows for quick adaptation of the executed predicted motion to account for changes in the environment using teleoperation is not present in the current literature.

This research was aimed to close this gap by providing the following contributions:

1) Development of a Learning from Demonstration (LfD) framework for a care robot, which uses the detected object as input and the end effector trajectory as output, thus providing a semi-autonomous solution.
2) A teleoperated online learning method that can quickly adapt the executed predicted motion using a haptic stylus (Phantom Omni).
3) A proof of concept analysis of the developed method in 2D and using a robot simulation environment on different object positions and initial poses, in addition to a human factors experiment to show how other people perform using the developed method compared to a baseline.

II. Method Design

To achieve teleoperated online learning, a general LfD framework is build using Probabilistic Movement Primitives (ProMP). Regular ProMP [13][14] is extended by conditioning on an external state variable $\vec{s}$ as described in [8] and [13], which will be called conditioned-ProMP throughout the rest of this research. After an initial model is trained offline using conditioned-ProMP, predictions are refined online and the model is updated using the online learning framework. Instead of kinaesthetic teaching as was done in [8], a novel teleoperated method is used to adapt the motion. An overview of the implemented method in this research is depicted in Fig. 3.

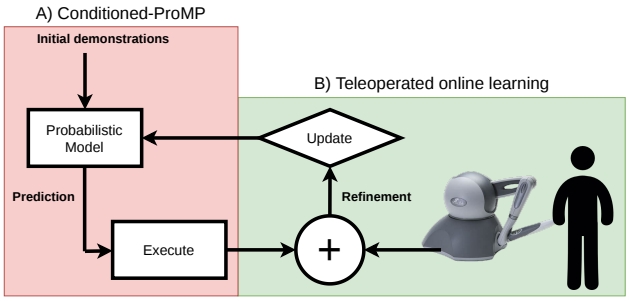


Fig. 3. Overview of the implemented framework. Initial demonstrations are used to train a model offline using conditioned-ProMP. This model generates a prediction, which is adapted online using the teleoperated refinement mechanism, after which it is used to update the model.

A. Conditioned-ProMP

ProMP is a locally weighted learning method [15], which means that each demonstrated trajectory $\vec{\tau}_M$ (Cartesian position and orientation in quaternions, see Appendix C for the details of creating the initial demonstrations) is modeled using a linear combination of N amount of weights $w_{M,N}$ multiplied by a corresponding basis function ϕ_N. The demonstrated trajectories are sampled using T time steps, and are approximated via N number of basis functions:

$$\begin{aligned}\vec{\tau}_M &= \Phi\vec{w}_M + \vec{\epsilon} \qquad (1)\\ &= \begin{bmatrix} \phi_1(1)w_{M,1} + \phi_2(1)w_{M,2} + \ldots + \phi_N(1)w_{M,N} \\ \phi_1(2)w_{M,1} + \phi_2(2)w_{M,2} + \ldots + \phi_N(2)w_{M,N} \\ \vdots \\ \phi_1(T)w_{M,1} + \phi_2(T)w_{M,2} + \ldots + \phi_N(T)w_{M,N} \end{bmatrix} + \vec{\epsilon},\end{aligned}$$

where Φ is the basis function matrix:

$$\Phi = \begin{bmatrix} \phi_1(1) & \phi_2(1) & \ldots & \phi_N(1) \\ \phi_1(2) & \phi_2(2) & \ldots & \phi_N(2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(T) & \phi_2(T) & \ldots & \phi_N(T) \end{bmatrix}, \qquad (2)$$

w_M are the weights and $\vec{\epsilon}$ is a vector with length T containing zero mean independently identically distributed Gaussian noise which means that $\vec{\epsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I}_{T\times T})$ [16]. Fig. 4 visualizes such trajectory modeling using locally weighted learning where Gaussian basis functions are used, but any other function can also be used.

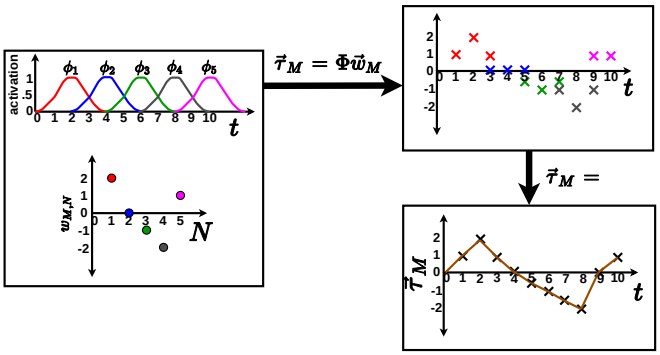


Fig. 4. Illustration of the local linear model process. On the left images the Gaussian basis functions and the corresponding weights are depicted. On the right images, a visualization of Φw_M is shown. The crosses correspond to the color coded Gaussian basis function at a certain time step multiplied by a weight. For example the first red cross $\phi_1(1)w_{M,1} = 0.5 \cdot 2 = 1.0$. When multiple crosses are present at one time step, they are added. For example at $t = 7$, $\phi_3(7)w_{M,3}$ is added to $\phi_4(7)w_{M,4}$. Doing this for each time step results in the local linear model of $\vec{\tau}_M$ as depicted in the bottom right graph.

Using the Φ matrix and the demonstrated trajectory $\vec{\tau}_M$, we can compute the weight vector from Equation (1) using ordinary least squares:

$$\begin{aligned}\vec{w}_M &= (\Phi^T\Phi)^{-1}\Phi^T\vec{\tau}_M \qquad (3)\\ &= [w_{M,1}, w_{M,2}, \ldots, w_{M,N}]^T\end{aligned}$$

For each trajectory $\vec{\tau}_M$ a weight vector $\vec{w}_M$ is derived and stacked along a matrix:

$$\mathbf{W} = \begin{bmatrix}\vec{w}_1 & \vec{w}_2 & \ldots & \vec{w}_M\end{bmatrix} \qquad (4)$$

The mean $\vec{\mu}_{\vec{w}}$ and covariance $\Sigma_{\vec{w}}$ of this matrix along the horizontal axis can be determined and used to build a probability distribution, which is assumed to be a Gaussian with mean $\vec{\mu}_{\vec{w}}$ and variance $\Sigma_{\vec{w}}$:

$$P(\vec{w}) = \mathcal{N}(\vec{\mu}_{\vec{w}}, \Sigma_{\vec{w}}) \qquad (5)$$

After a probability distribution over the weights is determined, the weights are conditioned on an external state variable $\vec{s}$, as described in [13]. In this application, $\vec{s}$ is the object position that we are trying to reach, which is extracted using an Aruco marker detection algorithm [17]. This means the joint probability is modeled between $\vec{w}$ and $\vec{s}$, thus the probability that both $\vec{w}$ and $\vec{s}$ occur. It is assumed that the joint probability distribution is normally distributed:

$$P(\vec{w} \cap \vec{s}) = \mathcal{N}(\vec{\mu}_{joint}, \Sigma_{joint}) \qquad (6)$$

$$\text{where } \vec{\mu}_{joint} = \begin{bmatrix}\vec{\mu}_{\vec{w}} \\ \vec{\mu}_{\vec{s}}\end{bmatrix}, \ \Sigma_{joint} = \begin{bmatrix}\Sigma_{\vec{w}\vec{w}} & \Sigma_{\vec{w}\vec{s}} \\ \Sigma_{\vec{s}\vec{w}} & \Sigma_{\vec{s}\vec{s}}\end{bmatrix}$$

This enables conditioning the weights $\vec{w}$ on the object position $\vec{s}$ by computing the following conditional probability distribution:

$$P(\vec{w}|\vec{s}) = \mathcal{N}(\vec{\mu}_{\vec{w}|\vec{s}}, \Sigma_{\vec{w}|\vec{s}}) \qquad (7)$$

Due to the properties of a multivariate conditional distribution [18], the conditional mean vector and covariance matrix

are described by Equation (8) and Equation (9).

$$\vec{\mu}_{\vec{w}|\vec{s}} = \vec{\mu}_{\vec{w}} + \Sigma_{\vec{w}\vec{s}}\Sigma_{\vec{s}\vec{s}}^{-1}(\vec{s} - \vec{\mu}_{\vec{s}}), \tag{8}$$

$$\Sigma_{\vec{w}|\vec{s}} = \Sigma_{\vec{w}\vec{w}} - \Sigma_{\vec{w}\vec{s}}\Sigma_{\vec{s}\vec{s}}^{-1}\Sigma_{\vec{s}\vec{w}} \tag{9}$$

In order to prevent the matrix $\Sigma_{\vec{s}\vec{s}}$ becoming singular, a small amount of noise $\Sigma_n = 0.03 \cdot \mathbf{I}$ is added [19]:

$$\Sigma_{\vec{s}\vec{s}} = \Sigma_{\vec{s}\vec{s}} + \Sigma_n \tag{10}$$

After that the following conditional probability distribution can be constructed:

$$p(\vec{\tau}|\vec{s}) = \mathcal{N}(\vec{\mu}_{\vec{\tau}|\vec{s}}, \Sigma_{\vec{\tau}|\vec{s}}) \tag{11}$$

The basis function matrix Φ, which is the linear mapping between $\vec{w}$ and $\vec{\tau}$, can be used to transform $\vec{\mu}_{\vec{w}|\vec{s}}$ and $\Sigma_{\vec{w}|\vec{s}}$ using Equation (1). Since the distribution of $\epsilon \sim \mathcal{N}(0, \sigma^2\mathbf{I}_{T\times T})$, $\vec{\mu}_{\vec{\epsilon}} = 0$ and the mean vector and covariance matrix of the conditional distribution $P(\vec{\tau}|\vec{s})$ are given by:

$$\vec{\mu}_{\vec{\tau}|\vec{s}} = \Phi\vec{\mu}_{\vec{w}|\vec{s}} \tag{12}$$

$$\Sigma_{\vec{\tau}|\vec{s}} = \sigma^2\mathbf{I}_{T\times T} + \Phi\Sigma_{\vec{w}|\vec{s}}\Phi^T \tag{13}$$

This $\vec{\mu}_{\vec{\tau}|\vec{s}}$ represents the generalized trajectory, which is dependent on the object position $\vec{s}$. In addition $\Sigma_{\vec{\tau}|\vec{s}}$ denotes the covariance, or uncertainty in the prediction.

B. Teleoperated online learning

To conveniently adapt the executed predicted end effector motion online via teleoperation, both visual and haptic feedback are provided to the operator, which are explained throughout the rest of this section. The online learning framework starts with an initial model that produces a prediction $\vec{\tau}_{\mathrm{d}}$, which is converted to joint positions $\vec{u}_{\mathrm{r}}$ using the inverse kinematics of the robot as can be seen in Fig. 5 [20].

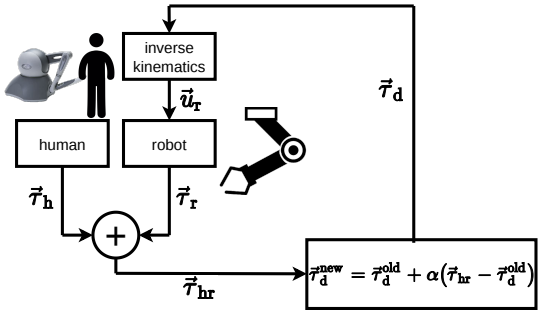


Fig. 5. Online learning framework, adapted from [8]

When executing these joint positions, the controller produces the trajectory $\vec{\tau}_{\mathrm{r}}$ and the teleoperator is able to adapt this trajectory, resulting in $\vec{\tau}_{\mathrm{hr}}$. To do this, the operator moves the master device in the desired adaptation direction and while doing so the operator feels a force proportional to the magnitude of the refinement. This haptic feedback is determined by a spring damper system, which relates the normalized master position to the magnitude and direction of the force feedback in the master device:

$$\vec{F}_{master} = -K_{fb}(K \cdot \vec{p}_{master} + D \cdot \dot{\vec{p}}_{master}), \tag{14}$$

where K and D are diagonal matrices containing the stiffness and damping coefficients in x, y and z direction, where $k_{xx} = k_{yy} = k_{zz} = k = 50$N/m and $d_{xx} = d_{yy} = d_{zz} = d = 2$Ns/m which are empirically determined. Furthermore $\vec{p}_{master}$ is the current master position relative to the initial master position, $\dot{\vec{p}}_{master}$ is the time derivative of $\vec{p}_{master}$ and K_{fb} is the force feedback scaling factor which is set to 1. When the human decides to intervene the currently executed prediction, he/she moves the master device into the desired refinement direction. This means that the operator is able to adapt the position commands from the reference trajectory, of which an illustration is depicted in Fig. 6.

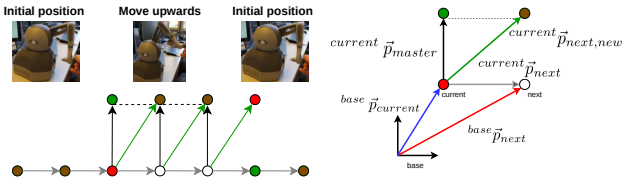


Fig. 6. Illustration of how the operator can apply a correction to the executed trajectory. The green and red dots represent the master and slave position respectively. When the master and slave are on the same position, the dot has a brown color. The white dot represents the reference trajectory, where the executed trajectory converges to if the operator does not apply any input.

In Fig. 6 it can be seen that the position at the next time step is adapted, rather than the current time step. This future adaptation is similar to the usage of the look-ahead path error for determining haptic guidance forces, as this was found to result in stable vehicle behavior [21][22]. In this research this principle is used the other way around, where the current master position is used to influence the executed trajectory at the next time step. Since each executed trajectory has a fixed amount of $n = 75$ datapoints and the total time is empirically fixed to $T = 20$, the look-ahead time is fixed to 0.133s, similar to what was chosen in [22]. Both position adaptation of the current and future time step have been implemented and it was empirically confirmed that adapting the position of future time steps results in less overshoot and thus more stable behavior (see Appendix B for the implementation details of adapting the current time step).

To achieve this adaptation of the executed trajectory (red dot in Fig. 6), the position vector of step $i+1$ relative to step i is calculated, respectively called the $next$ frame and $current$ frame:

$${}^{current}\vec{p}_{next} = {}^{current}R_{base}({}^{base}\vec{p}_{next} - {}^{base}\vec{p}_{current}) \tag{15}$$

where $\vec{p} = [x,\ y,\ z]^T$ and ${}^{current}R_{base} = \mathbf{I}_{3\times3}$

The master position is normalized such that the zero position is approximately in the middle of the workspace, which can be seen in Fig. 6. Therefore to get the position of the human intervention (green dot in Fig. 6) combined with the reference position, we need to add this normalized master position ${}^{current}\vec{p}_{master}$ to ${}^{current}\vec{p}_{next}$:

$${}^{current}\vec{p}_{next,new} = {}^{current}\vec{p}_{next} + {}^{current}\vec{p}_{master} \tag{16}$$

The end effector position expressed in *base* frame is then send to the inverse kinematics of the robot:

$$^{base}\vec{p}_{next,new} = ^{base}\vec{p}_{current} + ^{base}R_{current}(^{current}\vec{p}_{next,new}) \tag{17}$$

After the operator creates $\vec{\tau}_{\text{hr}}$, which is basically $^{base}\vec{p}_{next,new}$ at every time step, the prediction $\vec{\tau}_{\text{d}}$ is updated using:

$$\vec{\tau}_{\text{d}}^{\text{new}} = \vec{\tau}_{\text{d}}^{\text{old}} + \alpha(\vec{\tau}_{\text{hr}} - \vec{\tau}_{\text{d}}^{\text{old}}), \tag{18}$$

where $\alpha \in [0, 1]$ indicates how much the difference between $\vec{\tau}_{\text{hr}}$ and $\vec{\tau}_{\text{d}}^{\text{old}}$ will change $\vec{\tau}_{\text{d}}^{\text{old}}$. This value can be adapted based on the confidence of the operator in its refinement, but in this application α is set to 1 as done in [8]. This loop continues until some measure of success is reached (operator is satisfied in [8]) such that $\vec{\tau}_{\text{hr}} = \vec{\tau}_{\text{d}}^{\text{old}}$ and $\vec{\tau}_{\text{d}}^{\text{new}} = \vec{\tau}_{\text{d}}^{\text{old}}$, resulting in no update.

Visual feedback is provided through the wrist and head camera view, and additionally the point cloud of the environment is shown, enabling visualization of the prediction ($\vec{\tau}_{\text{d}}^{\text{old}}$) and refined trajectory ($\vec{\tau}_{\text{d}}^{\text{new}}$) with respect to the environment as can be seen in Fig. 7.

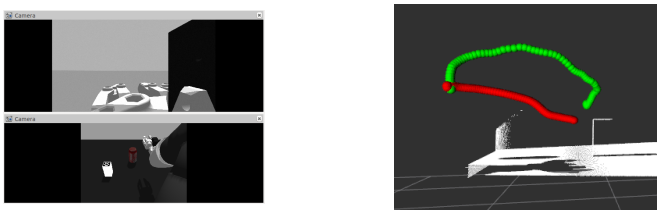


Fig. 7. Left: Wrist and head camera view, Right: Point cloud and visualization of trajectories. Green: Refined trajectory, Red: Predicted trajectory.

After refining the trajectory using Equation (18), $\vec{\tau}_{\text{d}}^{\text{new}}$ is used to update the conditioned probability distribution. First $\vec{w}_M$ is calculated from $\vec{\tau}_{\text{d}}^{\text{new}}$ using Equation (3), after which a vector $\vec{x}$ is created by appending $\vec{s}$ to $\vec{w}_M$: $\vec{x} = [\vec{w}_M^T, \vec{s}^T]$. $\vec{x}$ is then used with Welford's method for updating the mean and covariance incrementally, which means one data sample is used [8][23]. This method is able to quickly update the mean and covariance, without having to store all previous data. Such Welford update step for one incoming data point $\vec{x}$ is given as:

$$\vec{\mu}_{new} = \vec{\mu}_{old} + \frac{1}{M}(\vec{x} - \vec{\mu}_{old}) \tag{19}$$

$$\Psi_{new} = \Psi_{old} + \frac{(M-1)}{M}(\vec{x} - \vec{\mu}_{old})^T(\vec{x} - \vec{\mu}_{old}) \tag{20}$$

$$\text{with} \quad \Sigma = \frac{\Psi}{M-1},$$

and a comparison between regularly calculating the covariance and Welford's method can be found in Appendix B.

C. Data pre-processing

The raw data generated for the initial demonstrations and $\vec{\tau}_{\text{hr}}/\vec{\tau}_{\text{d}}^{\text{old}}$ are processed before being input to the conditioned-ProMP model. The first step is to align the demonstrations with the same condition using Dynamic Time Warping (DTW). Subsequently the trajectories are resampled to contain 10 datapoints, after which the trajectories are converted to be relative to the object position instead of the base frame. This allows to generalize towards different initial end effector poses in addition to different object positions. An overview of these steps is depicted in Fig. 8, and will be discussed in more detail in the sections below. In the online learning procedure, both $\vec{\tau}_{\text{hr}}$ and $\vec{\tau}_{\text{d}}^{\text{old}}$ are aligned using DTW, resampled and converted to be relative to the object position.

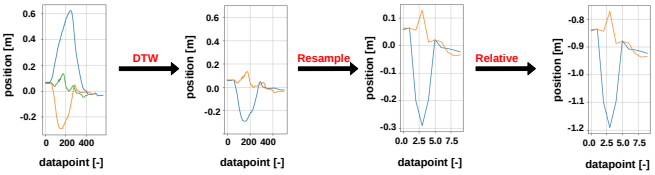


Fig. 8. Overview of the demonstration data pre-processing steps

1) Initial demonstrations: These are not performed at equal speed, so to compare the demonstrations with similar conditions, they have to be aligned in time. To achieve this, DTW is used which determines similarities between two time series, and is able to align them using an optimal warping path [24][25]. To illustrate the need for such alignment, an example is shown in Fig. 9.

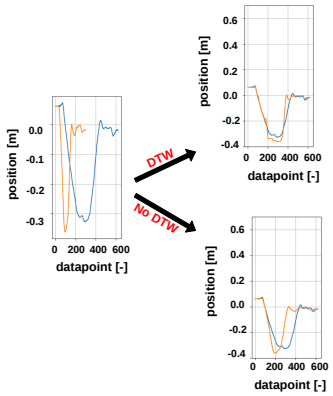


Fig. 9. A comparison between using Dynamic Time Warping to align two trajectories, and just resampling the trajectories to have the same amount of datapoints

The implementation of DTW is done using the FASTDTW algorithm [26][27], which details are shown in Appendix D. First the difference in condition is calculated between each demonstrated trajectory, and if this difference is below a threshold (0.01m) they are assumed to have the same condition. This results in a difference matrix, which is used in combination with the threshold to determine the trajectories with the same condition:

$$\begin{bmatrix} d_{11} & d_{12} & \dots & d_{1M} \\ d_{21} & d_{22} & \dots & d_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ d_{M1} & d_{M2} & \dots & d_{MM} \end{bmatrix} \tag{21}$$

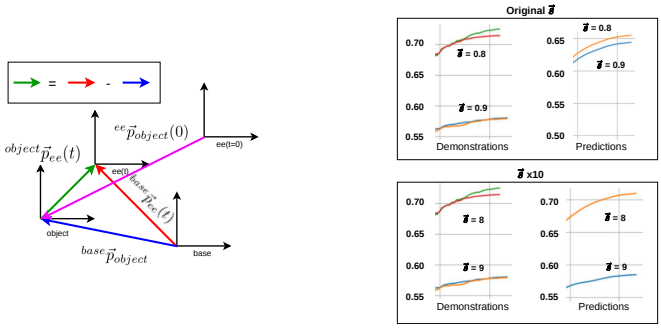


Fig. 10. Steps taken to convert the demonstration to be relative to the object position. Green arrow: trajectory relative to the object, Purple arrow: object position relative to the end effector at $t = 0$, which is multiplied by 10 and used as input to the model. An illustration of why this is needed is depicted in the right image.

When multiple trajectories are demonstrated for the same condition, we need to select two demonstrations for alignment since DTW is only able to compare two time-series [28]. This is done by using the distance value returned after applying the DTW algorithm. Each demonstration is compared with the other demonstrations, and the sum of the distances is calculated:

$$distance(i) = \sum_{j=1}^{M} \text{FASTDTW}(i,j) \quad \forall i \in \{1, 2, \dots M\} \tag{22}$$

The trajectory with the lowest sum distance is chosen as the reference trajectory [28]. This trajectory is assumed to be the most similar to the other trajectories for this condition, thus in terms of similarity it represent the demonstrations for this condition the best. The reference trajectory and the trajectory most similar to the reference are the only trajectories used for alignment using DTW.

Subsequently the trajectories are resampled to contain $n = 10$ datapoints after which they are transformed to be relative to the object position. This n is empirically determined to be a trade-off between overfitting the demonstrations and not capturing important environmental features. To make the trajectory relative to the object position, the following vector is created as function of time:

$$^{object}\vec{p}_{ee}(t) = {}^{object}R_{base}({}^{base}\vec{p}_{ee}(t) - {}^{base}\vec{p}_{object}) \tag{23}$$

where $^{object}R_{base} = \mathbf{I}_{3\times3}$

The vectors $^{base}\vec{p}_{ee}(t)$, $^{base}\vec{p}_{object}$ and $^{base}\vec{p}_{ee,object}(t)$ are represented by the red, blue and green arrow depicted in Fig. 10. The purple vector $^{ee}\vec{p}_{object}(0)$ is multiplied by 10, after which it is used as input $\vec{s}$ to the conditioned-ProMP model and the need for this multiplication is depicted in Fig. 10. When a prediction is made it is relative to the object position, so to execute it we assume a static object and add $^{base}\vec{p}_{object}$ to $^{object}\vec{p}_{ee}(t)$ to get $^{base}\vec{p}_{ee}(t)$.

2) Online learning: In order compute Equation (18), $\vec{\tau}_{\text{d}}^{\text{old}}$ and $\vec{\tau}_{\text{hr}}$ need to be equal in length. This is not the case in general, for example when the trajectory needs to stop earlier to not kick over the object. This is why a resampling method is needed, for which (spherical) linear interpolation is used [29]. Either $\vec{\tau}_{\text{d}}^{\text{old}}$ or $\vec{\tau}_{\text{hr}}$ is upsampled to match the trajectory with the maximum size. Problems arise however when subtracting these two trajectories after resampling, since increasing the amount of samples will "stretch" the trajectory. An illustration of this problem is depicted in the left image of Fig. 11. This means that these trajectories cannot be subtracted directly after resampling, and we need a method that aligns these trajectories. To accomplish this, again Dynamic Time Warping (DTW) is used [24]. A high level illustration of how DTW is used here is shown in Fig. 11, but for more details Appendix B should be consulted.

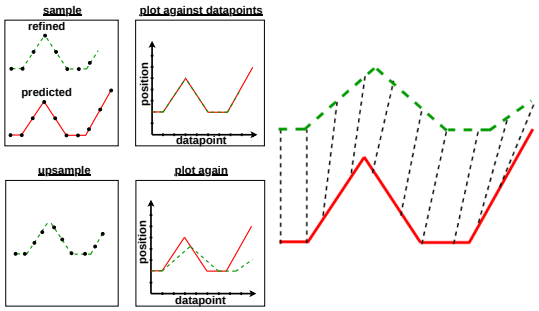


Fig. 11. Left: Illustration of the problem arising when resampling either the predicted or the refined trajectory. Right: Illustration of how Dynamic Time Warping aligns the predicted (red) and refined (dashed green) trajectory.

An overview of the complete framework, including the creation of the initial model and the refinement loop, is depicted in Algorithm 1.

III. Experimental Methods

To demonstrate the proof of concept of the developed teleoperated online learning method, we performed an experiment on a 2D setup, not related to the care robot and similar to what was done in [8]. Furthermore the proof of concept is demonstrated in a 3D robot simulation environment for different object positions and initial poses, after which a human factors experiment is performed to compare the developed method against three other baseline methods. The conditioned-ProMP model in itself has also been evaluated on basic theoretical examples to evaluate its functionality, of which details are shown in Appendix A.

A. 2D example

This setup consists of a particle that has constant velocity in x-direction, and the goal is to let this particle move through two viapoints as depicted in Fig. 12. Instead of the object position, $\vec{s}$ is defined as the y-coordinates of the two viapoints. The x-coordinates are fixed at $x_1 = 2.0$ and $x_2 = 3.6$ and the y-coordinates y_1 and y_2 are varied between -10, 0 and 10, which results in 9 different possible situations [8]. An initial model is trained to reach the viapoints at $\vec{s} = [0, 0]$ as can be seen in Fig. 12. When executing a prediction, the green particle moves along the red line, while the operator can apply a position command in y-direction by moving the stylus upwards (positive position)

Algorithm 1: Online learning of conditioned-ProMP

for *each object position* **do**
  create initial raw demonstrations
  determine the two most similar demonstrations and use these for DTW (Equation (22))
  resample to 10 datapoints
  convert demonstrations to be relative to $\vec{s}$ (Equation (23))
end
Initialize μ_{joint} and Σ_{joint} using these demonstrations (Equation (6))
start refinement loop:
for *each condition* $\vec{s}$ **do**
  determine $\vec{\mu}_{\vec{w}|\vec{s}}$ and $\Sigma_{\vec{w}|\vec{s}}$ (Equation (8) and Equation (9))
  determine $\vec{\mu}_{\vec{\tau}|\vec{s}}$ and $\Sigma_{\vec{\tau}|\vec{s}}$ (Equation (12) and Equation (13))
  $\vec{\tau}_d = \vec{\mu}_{\vec{\tau}|\vec{s}}$
  while *prediction is not successful* **do**
    compute joint position commands $\vec{u}_r$ from $\vec{\tau}_d$
    robot executes $\vec{\tau}_r$ and the teleoperator is allowed to refine this trajectory $\rightarrow \vec{\tau}_{hr}$
    apply DTW to $\vec{\tau}_d$ and $\vec{\tau}_{hr}$
    $\vec{\tau}_{\mathrm{d}}^{\mathrm{new}} = \vec{\tau}_{\mathrm{d}}^{\mathrm{old}} + \alpha(\vec{\tau}_{\mathrm{hr}} - \vec{\tau}_{\mathrm{d}}^{\mathrm{old}})$ (Equation (18))
  end
  $\vec{w}_M = (\Phi^T\Phi)^{-1}\Phi^T\vec{\tau}_{\mathrm{d}}^{\mathrm{new}}$ (Equation (3))
  $\vec{x} = [\vec{w}_M^T, \vec{s}^T]$
  use $\vec{x}$ to update $\vec{\mu}_{joint}$ and Σ_{joint} using Welford (Equation (19) and Equation (20))
end

and downwards (negative position):

$$y_i = \beta \cdot \vec{p}_{master,z} \tag{24}$$

where $\vec{p}_{master,z}$ is the master position in z-direction (same as in Equation (14)), multiplied by a factor $\beta = 10$. This is a different approach compared to [8], where the operator can apply y-acceleration commands using a keyboard. But because in our method the refined trajectory for the robot end effector is generated by giving position commands, we chose to also do this in the 2D example. The initial prediction $\vec{\tau}_{\mathrm{d}}^{\mathrm{old}}$ and $\vec{\tau}_{\mathrm{d}}^{\mathrm{new}}$ are depicted as red and green 2D trajectories respectively.

Fig. 12. Setup used in the 2D example, the blue circles and red line represent the viapoints and predicted trajectory respectively. Left: Initial model trained at $\vec{s} = [0, 0]$, which fails to reach the viapoints. Right: Refinement created using teleoperation, depicted as a green line

To determine the performance of the predictions, the mean square error

$$MSE = \frac{1}{2}\Big((\vec{\tau}_d(2.0) - y_1)^2 + (\vec{\tau}_d(3.6) - y_2)^2\Big) \tag{25}$$

between the viapoint and prediction y-coordinates is calculated as function of the amount of observed conditions [8]. The initial amount of observed conditions is 1 at $\vec{s} = [0, 0]$ and after observing each of the 9 conditions, all the conditions are evaluated again for their MSE. Then additional generalizations are performed on $\vec{s} = [\pm 20,\ \pm 20]$, $\vec{s} = [\pm 15,\ \pm 15]$ and $\vec{s} = [\pm 5,\ \pm 5]$.

Another analysis is performed by placing an obstacle in the 2D environment after an initial model is trained for three conditions (see Appendix B). The lower left corner of the obstacle is placed at $x = 0.5$ and the prediction will collide with it, as can be seen in Fig. 13. This model is again adapted, but with the additional goal of avoiding the obstacle after which the generalizations of the model are evaluated.

Fig. 13. Prediction and refinement with an obstacle, the red and green trajectory are the prediction and refinement respectively.

B. Robot simulation environment

In a 3D simulation environment two different scenarios are considered: a table and dishwasher scene (see Fig. 14). In both scenarios an object is placed inside the scene, that needs to be reached without collision with the environment and without kicking the object over: the success criteria (see Appendix E for the details).

Fig. 14. 3D environments used for the analysis of the developed method. Left: Table environment, Right: Dishwasher environment

In the table and dishwasher environment, different experiments are performed, of which the properties are depicted in Table I.

Initial trajectories	Total conditions	Condition type	Setup
3	9	Object position	Table
2	4	Object position	Dishwasher (b_2)
6	12	Initial EE pose	Dishwasher (b_1)

TABLE I

ROBOT SIMULATION PROOF OF CONCEPT EXPERIMENTAL PROPERTIES, SEE APPENDIX B FOR THE INITIAL MODEL EVALUATIONS.

The amount of initial trajectories was empirically determined to generate predictions that do not have a shifted initial pose. When only one or two conditions were initially demonstrated (as was done in the 2D example), the model has to extrapolate and the initial pose is shifted (see Appendix B for details). For each condition the prediction is evaluated on the success criteria and if these are not satisfied, the prediction is refined using the developed method. It is analyzed for how many conditions refinements were necessary until the model is able to generate successful predictions for all of them.

In the dishwasher environment two analyses have been performed. In the first analysis, an initial model is trained to reach the object in basket position b_1 without collision with the dishwasher and without kicking the object over. After training the initial model, the basket is pulled outwards (basket position b_2), the generated predictions will fail and adaptation of the model is needed, which is illustrated in Fig. 15. The second analysis is done with basket position b_1 on the capability to handle different initial end effector poses.

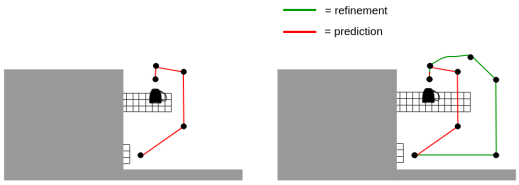


Fig. 15. Left: Basket position b_1 (correct prediction), Right: Basket position b_2 (incorrect prediction, refinement needed)

C. Human factors experiment

The goal of this experiment is to show that in addition to the proof of concept, other people are also able to use the developed method. This is done by comparing it against three other methods that can adapt a trained model, which are analyzed on how much time it takes to successfully (success criteria met) adapt a model (refinement time) and on the perceived workload by the operators, evaluated using NASA TLX [30] (see Appendix E for the GUI). After the experiment each participant is asked which method they liked the most and why. The experimental pipeline depicted in Fig. 16 shows the complete experiment workflow and the physical setup, which will be further explained throughout this section.

The same pulled out basket (b_2) experimental setup is used as in the proof of concept experiment, but each participant is going to adapt the model for only one object position ($^{base}\vec{o}_1 = \begin{bmatrix}0.8 & 0.05 & 0.7\end{bmatrix}$) and the end effector always starts at the same initial pose. Three identical initial models are adapted for each of the four methods per participant. For each model the operator has eight attempts to update the model, and per update 10 refinement attempts. As can be seen in Fig. 16, a refinement is defined as the

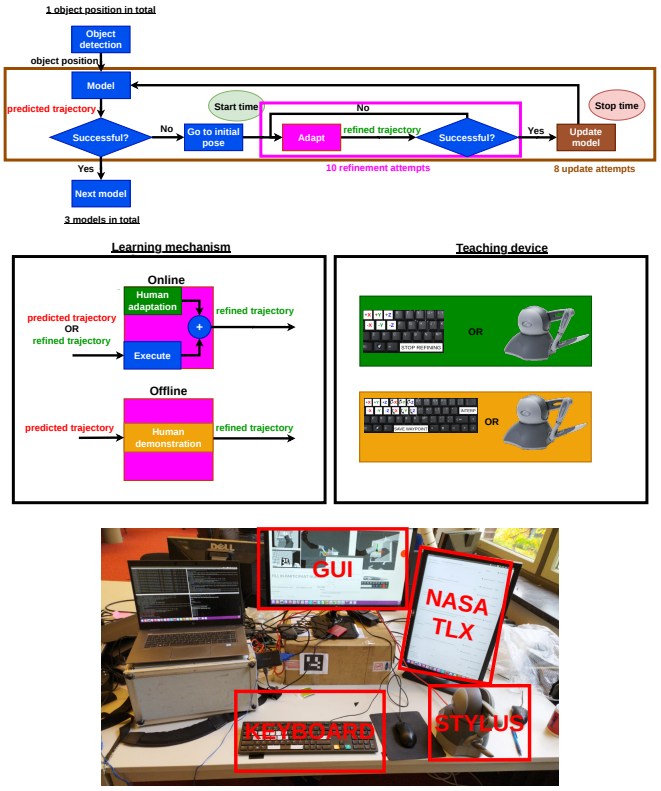


Fig. 16. Top: Experiment pipeline, where the purple block indicates where the human interaction takes place. This interaction is done with four methods, combining online and offline with a haptic stylus and the keyboard. In the online methods, the operator can choose between refining the current prediction or the previous refinement. Bottom: Experimental setup

creation of an adaptation of the prediction (or previous refinement). If the operator failed to adapt the model within these attempts, we continue with the next model and exclude this participant from the refinement time hypotheses testing.

Since state-of-the-art LfD methods focus on offline learning [31][32][33][34], this learning mechanism is chosen as the first baseline condition. The second condition that is varied is the teaching mechanism, for which the state-of-the-art literature uses either kinaesthetic teaching [8][5][11][9][35], teleoperation [31][32][33][34][10], shadowing [36], and although not well represented in scientific literature, the teach pendant is frequently used by operators to program industrial robots [37]. Because the robot in this application does not have a gravity compensation mode, kinaesthetic teaching is not possible and cannot be used as a baseline. The teleoperated methods either already use the Phantom Omni [31][34], the Haption Virtuose 6D [10] or use the other arm of a Baxter robot [32]. Since the Haption Virtuose 6D only has the addition of torque feedback, thus is not a significantly different device, this is not chosen as a baseline. Using another arm is not relevant for this application and the Phantom Omni is already used. Furthermore shadowing requires sensors to be placed on the human to extract the state of the human, which requires a complex setup and is thus infeasible. Because a representation of a teach

pendant is feasible to build, is a completely different teaching device than the Phantom Omni and is frequently used in industry, this teaching method is chosen as a baseline.

Such teach pendant is used to specify waypoints by moving the end effector using a set of buttons, after which they are being interpolated to generate a demonstration. A representation of the teach pendant is implemented using a keyboard, where the operator can adapt both position and orientation in x, y and z direction using the keys as shown in Fig. 17. Storing waypoints and interpolating between them is done by pressing the space and enter button respectively, as depicted in Fig. 17.

The combination of offline learning with the haptic stylus (OffStyl) and the keyboard (OffKey) is selected as baseline, in addition to online learning with the keyboard (OnKey). These methods are compared to the developed method in this research, which will be abbreviated by OnStyl.

Fig. 17. Left: Teach pendant, Right: Implemented representation of the teach pendant using a keyboard

OffKey is the representation of the teach pendant, so with both position and orientation teaching in an offline setting. In OnKey, the same developed online learning framework is used, except the x, y and z keys are used to make corrections to the motion instead of the stylus. This also means that no force feedback is provided to the operator. When using OffStyl, the motion is continuously tracked, but the operator is allowed to interactively couple/decouple the master from the slave (see Appendix C). The reason for choosing this is because in the state-of-the-art kineasthetic online teaching methods [9][8][11], and the research on teleoperated online teaching [10], the motion is also continuously tracked. After teaching a trajectory, instead of using a Welford update step, the matrix containing the weights of all the present trajectories $\mathbf{W}$ as depicted in Equation (4), is extended and the mean and covariance are recalculated. In the online methods, the success criteria are evaluated while doing the online refinement, whereas in the offline methods the refined trajectory is evaluated after creation by executing it again. In the refinement time calculations only the time spend actually creating the refined motion is recorded, as shown by start and stop time in Fig. 16. Before starting each method, the operator is trained in generating successful refinements to ensure approximate steady state behavior over the participants (see Appendix E). Because for a valid statistical evaluation of the refinement time, all the models for each method need to be adapted, the participant was excluded from the hypothesis test for the refinement time if training for one or more methods has failed. This is the same as when a participant failed to adapt one or more models for one or more methods.

The expectation is that the combination of online with stylus has the lowest refinement time and workload. To statistically evaluate this, a one sided t-test is performed, using null and alternative hypotheses:

$$H_{0,1} : \mu_{stylus} > \mu_{keyboard}, \quad H_{1,1} : \mu_{stylus} < \mu_{keyboard},$$
$$H_{0,2} : \mu_{online} > \mu_{offline}, \quad H_{1,2} : \mu_{online} < \mu_{offline},$$
$$\text{where median } \mu = \mu_{refinement\ time}$$

$$H_{0,3} : \mu_{stylus} > \mu_{keyboard}, \quad H_{1,3} : \mu_{stylus} < \mu_{keyboard},$$
$$H_{0,4} : \mu_{online} > \mu_{offline}, \quad H_{1,4} : \mu_{online} < \mu_{offline},$$
$$\text{where median } \mu = \mu_{workload}$$

The null hypotheses $H_{0,i}$ are used to test if the median refinement time and workload using the stylus is higher than using the keyboard and online is higher than offline. The alternative hypotheses $H_{1,i}$ are defined as the opposite, namely the median refinement time and workload are lower for the stylus compared to the keyboard and online is lower than offline.

Because both the refinement time and the workload data are not normally distributed, Wilcoxon signed rank-test is used instead of a one sided t-test [38][39] (see Appendix E). Furthermore the order effects of presenting the different methods have been counterbalanced using a balanced Latin Square experimental design [40] (see Appendix E). When a participant failed the training or failed to adapt a model, the group this participant belonged to is repeated to ensure this counterbalancing. It should be noted that since the workload of the failed participants was included in the statistics, these are not properly counterbalanced due to some groups being repeated more than others.

18 participants have been gathered for this experiment, which have filled in a questionnaire with a small amount of information before starting the experiment: their gender, field of work, left or right handedness, gaming (WASD) and teleoperation experience (see Appendix E for this information and the GUI). Before the experiment, the participants signed a consent form and the research is approved by the Human Research Ethics Committee of Delft University of Technology.

IV. Results

A. 2D example

From Fig. 18 and Fig. 19 it can be seen that after observing and adapting the prediction of four conditions, the MSE was zero for all the upcoming conditions. When evaluating the predictions for every condition, including the conditions from

-20 to +20 and -5 to +5, they show that the viapoints are successfully reached, of which details the can be found in Appendix B.

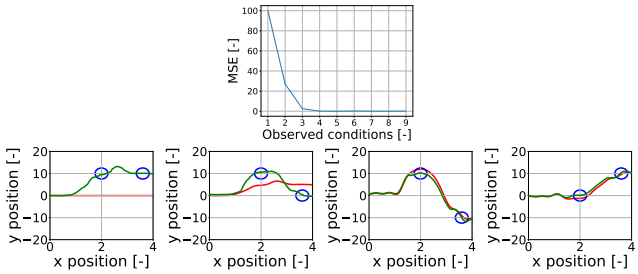


Fig. 18. Top: MSE as function of the amount of observed conditions, Bottom: prediction and refinement for each of these conditions

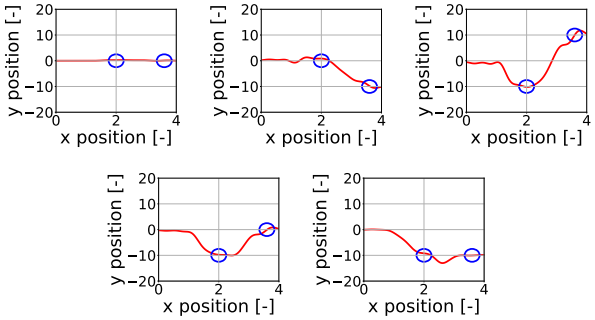


Fig. 19. Successful generalizations across other conditions

When adapting an initially trained model with an obstacle, it can be seen in Fig. 20 that for each of the nine conditions the prediction needs to be adapted. After updating the model for these nine conditions, each of them was again checked and confirmed to avoid the obstacle (see Appendix B for the details). Furthermore generalizations towards conditions -20 to 20 and -5 to 5 are evaluated, of which the results are also shown in Appendix B. For each condition except [-20, 20] and [0, 20] as shown in Fig. 21, the obstacle is successfully avoided while reaching the viapoints.

B. Robot simulation environment

The results of the table scene are depicted in Fig. 22, where it can be seen that for object positions [0.7, -0.2, 0.9] and [0.6, -0.2, 0.9] refinements were needed. After performing a second iteration across all object positions, each prediction was successful (details in Appendix B).

For the dishwasher scene with a fixed object position the results are depicted in Fig. 23, where refinements were only necessary for the first two object positions, after which the model was able to make correct predictions for the other two. Again after a second iteration over all object positions, the model was able to generate successful predictions across all four object positions (see Appendix B).

The results for the different initial poses are depicted in Fig. 24, where it can be seen that only for the first two initial poses refinements where needed. After that the model was able to generalize towards the other four initial poses,

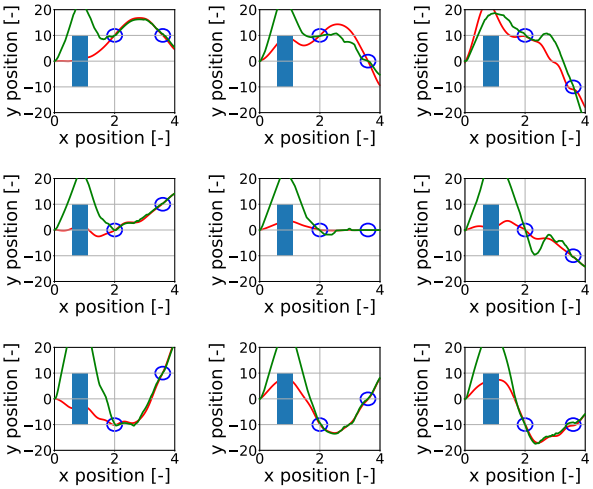


Fig. 20. When an obstacle is present, the initially trained model had to be adapted for nine conditions. The red and green line depict the predicted and refined trajectory respectively.

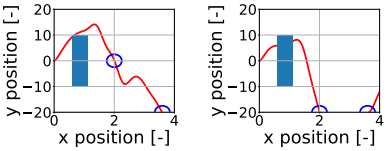


Fig. 21. Generalizations that failed in avoiding the obstacle, which are the conditions [0, -20] and [-20, -20] in the left and right figure respectively.

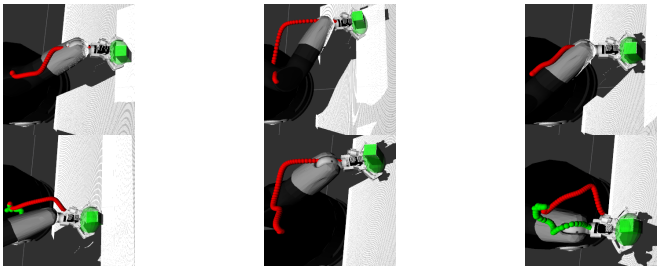


Fig. 22. Refinement loop performed on each of the other object positions. From left to right and up to down the object positions are: [0.8, 0.0, 0.9], [0.7, 0.2, 0.9], [0.7, 0.0, 0.9], [0.7, -0.2, 0.9], [0.6, 0.2, 0.9] and [0.6, -0.2, 0.9]. For the 4th and 6th object position refinements were needed as the object was kicked over and not reached (see Appendix B).

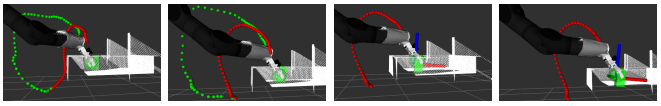


Fig. 23. Refinement loop performed on each object position inside the dishwasher with basket position b_2. From left to right and top to bottom, the object positions are ${}^{base}[0.8,\ 0.2,\ 0.9]$, ${}^{base}[0.8,\ 0.1,\ 0.9]$, ${}^{base}[0.8,\ 0.0,\ 0.9]$ and ${}^{base}[0.8, -0.1,\ 0.9]$. Refinement were only needed for the first two object positions, after which the environmental constraint is generalized towards the other two object positions.

and after iterating again over all the 12 initial poses, each prediction passes the success criteria (see Appendix B for the details).

		Methods				Wilcoxon	
		OnStyl	OnKey	OffStyl	OffKey	Online > Offline	Stylus > Keyboard
Refinement time [s]	M	89.76	97.03	336.82	314.45	$\mathbf{p = 7.94 \times 10^{-12}}$	$p = 0.755$
	25	55.55	58.47	241.28	153.73	$t = 113.0$	$t = 1437$
$N = 36$	75	153.22	144.17	498.24	574.90		
Workload [0-100]	M	36.0	39.0	47.5	48.5	$\mathbf{p = 0.000512}$	$p = 0.302$
	25	25.0	31.0	27.0	21.0	$t = 1872.0$	$t = 3112.5$
$N = 54$	75	45.0	48.0	59.0	60.0		

TABLE II

THE TABLE SHOWS THE MEDIAN (M), FIRST QUARTILE (25), THIRD QUARTILE (75) AND THE RESULTS OF THE WILCOXON HYPOTHESIS TESTS WHERE SIGNIFICANCY ($p \leqslant 0.01$) IS INDICATED IN BOLD.

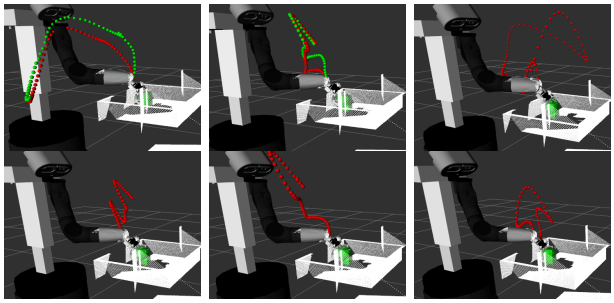


Fig. 24. First iteration of the refinement procedure across 6 different initial poses that are previously unobserved. Only for the first two initial poses refinements where needed, after which all predictions satisfied the success criteria.

C. Human factors experiment

Out of 18 participants six of them failed either in the training or to adapt a model in one of the methods, of which an overview is depicted in Table III. This means that in total 18 - 6 = 12 participants are used for the data analysis of the refinement time hypothesis, thus per method 36 models are evaluated.

			Methods			
			OnStyl	OnKey	OffStyl	OffKey
Group 1	Adapting	Model 1				
		Model 2				
	Training				5	13
Group 2	Adapting	Model 1	10			
		Model 2	10			
	Training				10	
Group 3	Adapting	Model 1			3, 7, 11	
		Model 2			11	
	Training					

TABLE III

PARTICIPANTS THAT FAILED AT ADAPTING A MODEL OR DURING THE TRAINING. GROUP 4 AND MODEL 3 HAVE BEEN OMITTED SINCE NO PARTICIPANT FAILED WITH THESE.

In Table II the median (M), first quartile (25) and third quartile (75) of the refinement time and workload are reported. The distribution of this data and the refinement time per model per method are depicted Fig. 26.

The relationship of the background information of the participants on the refinement time and workload is visualized in Fig. 25. Only the most interesting results have been reported here and for the other background information results Appendix E should be consulted. Note that no Wilcoxon signed rank test can be performed on any background parameter except the gaming experience, since the sample sizes are not equal. The p-value of the workload being higher for participants with high gaming experience is 0.048, which means that with a confidence level of 0.05 these participants experience lower workload for all the methods compared to having low game experience. For the refinement time the p-value is 0.189, and hence there is no significant difference in having high or low game experience on the refinement time.

The results of which method the participants liked the most is depicted in Fig. 27, and the details of their explanations have been reported in Table IV.

	Methods			
	OnStyl	OnKey	OffStyl	OffKey
Pros	Intuitive Low effort Good for small corrections Fast Spatial awareness easy	Easy Zero effort Less sensitive than Om Only 2DoF	Intuitive Big motion	Set waypoints accurately Not rushed More accurate than Om
Cons	Less control Rushed	Wrong button easily pressed Rushed	Demo easily messed up High spatial awareness needed	High mental effort Orientation hard

TABLE IV

REPORTED ADVANTAGES AND DISADVANTAGES OF THE METHODS

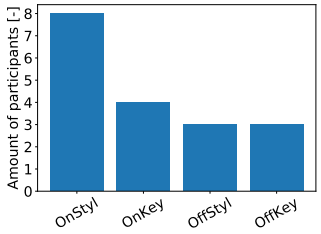


Fig. 27. Which method was liked the most

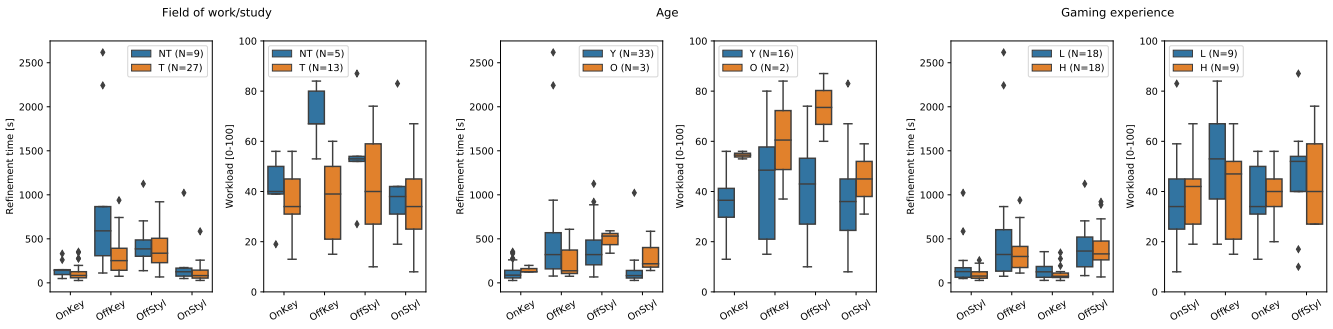


Fig. 25. Effects of field of work/study (Technical/Non-Technical), age (Young/Old), and game experience (Low/High) on the refinement time and workload.

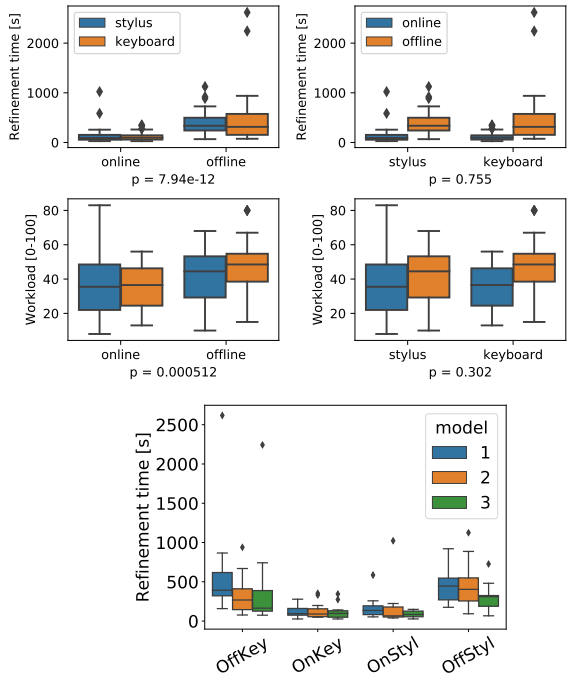


Fig. 26. Top: Visualization of the data from Table II, Bottom: Refinement time per model, where the blue, orange and green color represent model 1, 2 and 3 respectively.

V. Discussion

A. 2D example

It was shown that after adapting predictions for four different conditions, the model was able to generalize correctly towards all of them. Compared to [8], where after only three conditions the model successfully generalized, one more condition was needed in this experiment. This can be explained by the fact that in [8] they started with a model with less initial demonstrations, which means that the model is more quickly adapted. It could also mean that when using position commands to generate a refined trajectory, it is harder to exactly go through the center of the viapoints compared to using acceleration commands [8]. This can also be because in [8] when no input is given, the acceleration remains constant. While in this research, the operator has to manually keep the particle at a constant y position, so constantly applying an input.

Analyzing the 2D problem with an obstacle in Fig. 20 and in Fig. 21 it can be seen that for all conditions the prediction needs to be adapted to have successful generalizations for all considered conditions except [0, -20] and [-20, 20]. An explanation for the need of adapting for nine conditions instead of four is that three trajectories are used to train the initial model instead of one. This makes it slower to change the mean of the conditioned-ProMP model ($\vec{\mu}_{\vec{r}|\vec{s}}$ from Equation (13)), thus more refined trajectories need to be fed into the model. A solution to this is to either start with a smaller initial model, feed the same refined trajectory multiple times into the model or allow for the forgetting of older data. To achieve the latter, a forgetting factor can be used when adapting the model, for example using a Recursive Least Squares (RLS) update of the covariance matrix [41][42]. This leads to another problem called catastrophic forgetting, which means that even if the older data was still useful to solving the problem, it is still forgotten [43]. To solve this problem, continual learning algorithms could be researched [44][45].

B. Robot simulation environment

The robot simulation proof of concept experiments show that the developed method is able to adapt an initially trained model to acccount for unknown variations: goal deviation and an unforeseen obstacle. The method was able to do this by updating the model for both different object positions and initial end effector poses. It should be noted that the amount of conditions to refine is dependent on the amount of initial trajectories used to train the initial model and how exaggerated the motion is, as more trajectories and exaggerated motion both result in less updates needed to change the mean of the distribution (same as in the 2D example). An illustration of what happens when an initial model is trained with more trajectories, and when no exaggerated motion is used to update the model, is depicted in Appendix B.

Furthermore in the table and dishwasher scenarios no refinement was needed in the orientation of the end effector. When the task demands more constraint motion, it is necessary to also adapt the orientation of the prediction for example when an unforeseen obstacle forces the object to be reached from the side instead of from above. With the current developed method this is not possible, because the haptic stylus lacks torque feedback. This could be improved by using a different haptic device, such as the Haption Virtuose 6D [10]. In future work it would be interesting to analyze the refinement method with orientation extension for different environments.

To be practically feasible, correct predictions are necessary for all possible start poses and object poses. In this research a subset of all conditions is analyzed, and in future work it would be useful to do a more thorough analysis of the generalization capabilities of the method. The framework could then also be extended by incorporating different inputs to the conditioned-ProMP model, such as geometry information of the object/obstacle extracted using the RGBD camera. This method was not evaluated on the real robot, but to prove its value in reaching an object in real life partially unknown environments, further studies should be conducted. Another addition to the method could be to first optimize the position of the base of the robot with respect to the environment [46], as we want the arm to always be able to reach the object, which is in the current setup not the case.

To decrease the workload experienced by the operator, different visual cues could be investigated, for example using virtual reality to show the predicted and refined trajectory. Another example could be to show how the current refined trajectory will develop itself if the operator lets go of the master device, of which an illustration is depicted in Fig. 28. Or the current end effector position can be projected onto the nearest surface to gain better depth feedback.

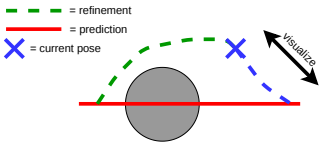


Fig. 28. Visual cue to show how the refined trajectory will develop itself when the operator stops refining

Other features to investigate are the ability to navigate through the trajectory and refine only the parts that the operator finds necessary. This way not the whole trajectory has to be executed, potentially reducing the experienced workload.

C. Human factors experiment

The hypotheses test results show that for both refinement time and workload, the median of online methods is significantly lower with confidence level of 0.01, since the p-values are 7.94×10^{-12} and 0.000512 respectively. This means that $H_{0,2}$ and $H_{0,4}$ are both rejected, thus the online methods have lower median refinement time and workload compared to the offline methods. Furthermore there is no significant difference in the median refinement time and workload using the keyboard or stylus, where the p-values are 0.755 and 0.302 respectively. This means that we cannot conclude on whether the keyboard or stylus contributes to lower median refinement time and/or workload.

The main explanation for the significance in refinement time and workload in favor of the online methods is that with online methods the operator only performs small adjustments to the executed trajectory instead of completely teaching a new one. This means that online methods are inherently faster in creating a single refinement as these are bounded by the execution time of the trajectory, and a lower amount of input from the operator is needed, possibly resulting in a lower workload. When the former statement is true, it is expected that there is a lower refinement time variability in the online compared to offline methods, which is confirmed by comparing the first and third quartile of the online and offline methods from Table II and Fig. 26. In addition to a lower input magnitude, a maximum of 3DoF can be perturbed in the online methods, which are the position of the end effector in x, y and z direction. In offline methods the orientation is added to the input (6 DoF's), which gives the operator more control over the end effector but has a higher potential of creating confusion and unintentionally perturbing DoF's. These statements are confirmed when looking at the opinion of the participants in Fig. 27, where the offline methods can be found "big motion" and "accurate" but also require "high mental awareness" and "high mental effort" for some participants. This indicates that although some participants found that the higher control contributed to accurately creating the demonstration they intended, but it also imposes higher workload and refinement time on other participants due to this possibility.

Another influence on the refinement time is the demonstrated strategy, which tended to change within and between the methods. The within change can be explained by assessing the bottom image of Fig. 26, where the refinement time decreases as function of the model number. This gives an indication that either the operator has a constant strategy in its mind and is figuring out how to translate this strategy to a demonstration using the specific method, or the strategy changes and the operator is figuring out what strategy works best. Since the offline methods have a higher slope in the medians of the refinement time compared to online methods and the method exposure is partially counterbalanced, it seems more reasonable that the participants had a more or less constant strategy but had more trouble using the offline methods to convey this strategy. This suggests that more training is needed using offline methods.

An explanation for finding no significance in using the stylus or the keyboard in both refinement time and workload, could be because the intuitiveness of the interface is person dependent. Some people reported OnKey to be easy, less sensitive than OnStyl and preferred the limited DoF's, while others easily pressed the wrong buttons and found it hard to figure out the axes. Another explanation could be because for most participants, the strategy used was pulling the arm backwards to avoid the basket. To achieve this, only one DoF has to be used and sometimes two to correct the height. This is achieved easily via the OnKey method by pressing the appropriate key, whereas using the OnStyl method easily all three DoFs are perturbed. When it is not intuitive how to use the stylus, easily multiple unintended DoF's will be perturbed using OnStyl, whereas using OnKey only one DoF will simultaneously be wrongly perturbed. When using OffStyl the six DoFs of the stylus are continuously tracked, which makes it even easier to perturb incorrect dimensions. Since in the OffKey method the motion is not continuously tracked, wrongly perturbed motion can be easily corrected. Another explanation for finding no significant difference between using the stylus and keyboard is that there is a difference in the implementation of the offline methods. In OffKey the participants can take its time, as it does not matter what is done in between the waypoints as long as the waypoints are correctly specified. OffStyl on the other hand continuously tracks the demonstrated motion, and when the operator makes a small mistake this can translate in an unsuccessful demonstration more easily. Therefore, as recommendation, OffStyl could be implemented similar to the teach pendant, where the operator can specify waypoints and interpolate between them. This is expected to show different statistical results in favor of the stylus, since the definition of offline will be more similar between the stylus and the keyboard implementation. Another difference between these methods is the result of the interpolation of the demonstration, because in OffStyl the motion is continuously tracked. Since the demonstration is interpolated to contain $n = 75$ datapoints, if the operator spends a relatively longer time at the start than the end, a lower amount of points will be placed at the beginning. This can translate to jumps in the demonstration that the operator did not intended and can lead to execution failures. This can be solved by defining more interpolation points of which a comparison is depicted in Fig. 29.

Fig. 29. Result of using different interpolation points, on the left and right image 75 and 500 interpolation points are chosen respectively

The same amount of interpolation points ($n = 75$) are present in the other methods, but in the online methods the operators tended to stick with the executed trajectory and does not spend more time at the end than the start. In OffKey, most participants placed an equal amount of points at the beginning and end. If significantly more points are placed at the end, the same behavior will present itself, but people did not have a tendency to do this. When a higher amount of interpolation points had been chosen, the difference between the stylus and the keyboard might be more pronounced.

Another finding was that people with low spatial awareness have the most trouble with the offline methods and especially OffStyl because the motion is continuously tracked. These people also had problems figuring out the orientation of the gripper using OffKey. This suggests that people with less spatial awareness should use an online rather than an offline method, or that more training is needed for offline methods.

From the median values in Fig. 25 it can be seen that there is an indication the young people with technical background and high game experience have lower refinement time and workload than the opposite. Although the influence of field of work and age cannot be evaluated statistically, the data suggests that we need young people with a technical background to use the developed method. In addition the p-value of 0.048 for the workload in favor of high game experience can statistically confirm ($\alpha = 0.05$) that we need people with high game experience to use the method. In the refinement time, the influence of these parameters is less obvious, and the p-value of 0.189 for the game experience also shows that no statistical significance is found between low/high game experience on the refinement time.

What can be seen from the failed participants in Table III is that three participants from group 3 failed adapting model 1, 2 or both. This is interesting because these participants started with OffStyl (see Appendix E for the method sequence per group), and could indicate that this method needs more training than the other methods. As can be seen in Appendix E, the median training time for the offline is higher compared to the online methods. In combination with the relative interaction time being lower for offline than online, this could lead to insufficient training. Since Table III shows that this is mostly the case with OffStyl, this could mean that OffStyl is harder to use than OffKey. Moreover participant 10 failed at both the methods involving the stylus, and this participant noted having trouble with using the device itself. Participant 13 tried to grasp the object from the side and had trouble figuring out the orientation to achieve this.

VI. Conclusion

This research presented a framework containing a conditioned-ProMP model which was shown to generate condition dependent trajectories in both a 2D problem, with via points as condition, and using the robot with the relative object position as condition. It was demonstrated that

the *teleoperated* online learning method allows humans to adapt an initially trained conditioned-ProMP model to deal with *unknown* variations: goal deviation and an unforeseen obstacle. The proof of concept is evaluated on a 2D problem and in a simulated table and dishwasher environment, where different object positions and initial end effector poses are considered. The human factors experiment showed that online methods perform significantly better on refinement time (time to successfully adapt one model) and workload compared to offline methods. The main explanation for this is that online methods require less magnitude and less DoF inputs from the operator. No significant difference was found between using the stylus and the keyboard, of which the main explanation is that this is task, person and implementation dependent. Moreover the participants subjectively liked my method the most because it is intuitive, requires low effort and is good for making small corrections. The main recommendations are to evaluate the proof of concept on the real robot and to extend the method with orientation refinement such that more complex tasks, where more DoFs need to be refined, can also be dealt with. The expectation is then that the combination of the stylus and online does perform the best.

REFERENCES

[1] W. J. S. P. DCSW and K. W. D. DSW, "Future trends in health and health care: Implications for social work practice in an aging society," *Social Work in Health Care*, vol. 52, no. 10, pp. 959–986, 2013, PMID: 24255978. DOI: 10.1080/00981389.2013.834028. eprint: https://doi.org/10.1080/00981389.2013.834028. [Online]. Available: https://doi.org/10.1080/00981389.2013.834028.

[2] C. T. Kovner, M. Mezey, and C. Harrington, "Who cares for older adults? workforce implications of an aging society," *Health affairs*, vol. 21, no. 5, pp. 78–89, 2002.

[3] E. v. d. V. Kees van Hee Michiel van Osch, *Meerwaarde van robotica in de zorg*, 2015.

[4] M. Rana, M. Mukadam, S. R. Ahmadzadeh, S. Chernova, and B. Boots, "Towards robust skill generalization: Unifying learning from demonstration and motion planning," in *Intelligent robots and systems*, 2018.

[5] G. Maeda, M. Ewerton, T. Osa, B. Busch, and J. Peters, "Active incremental learning of robot movement primitives," 2017.

[6] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[7] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[8] M. Ewerton, G. Maeda, G. Kollegger, J. Wiemeyer, and J. Peters, "Incremental imitation learning of context-dependent motor skills," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, IEEE, 2016, pp. 351–358.

[9] M. Saveriano, S.-i. An, and D. Lee, "Incremental kinesthetic teaching of end-effector and null-space motion primitives," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 3570–3575.

[10] F. Abi-Farraj, T. Osa, N. P. J. Peters, G. Neumann, and P. R. Giordano, "A learning-based shared control architecture for interactive task execution," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 329–335.

[11] F. Dimeas and Z. Doulgeri, "Progressive automation of repetitive tasks involving both translation and rotation," in *International Conference on Robotics in Alpe-Adria Danube Region*, Springer, 2018, pp. 53–62.

[12] A. Conkey and T. Hermans, "Active learning of probabilistic movement primitives," *ArXiv preprint arXiv:1907.00277*, 2019.

[13] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Using probabilistic movement primitives in robotics," *Autonomous Robots*, vol. 42, no. 3, pp. 529–551, 2018.

[14] F. team, *Python implementation of probabilistic motor primitives including a ros overlay.* 2016. [Online]. Available: https://github.com/baxter-flowers/promplib.

[15] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," in *Lazy learning*, Springer, 1997, pp. 11–73.

[16] A. Clauset, "A brief primer on probability distributions," in *Santa Fe Institute*, 2011.

[17] P. Robotics, *Software package and ros wrappers of the aruco augmented reality marker detector library*, 2017. [Online]. Available: https://github.com/pal-robotics/aruco_ros.

[18] E. C. o. S. PennState, *6.1 - conditional distributions — stat 505*, https://online.stat.psu.edu/stat505/lesson/6/6.1.

[19] L. Peternel and J. Babič, "Learning of compliant human–robot interaction using full-body haptic interface," *Advanced Robotics*, vol. 27, no. 13, pp. 1003–1012, 2013.

[20] H. Tomé, L. Marchionni, and A. R. Tsouroukdissian, "Whole body control using robust & online hierarchical quadratic optimization," in *IROS14 International Conference on Intelligent Robots and Systems*, 2014, pp. 14–18.

[21] M. Mulder, D. A. Abbink, and E. R. Boer, "The effect of haptic guidance on curve negotiation behavior of young, experienced drivers," in *2008 IEEE International Conference on Systems, Man and Cybernetics*, IEEE, 2008, pp. 804–809.

[22] H. Boessenkool, D. A. Abbink, C. J. Heemskerk, and F. C. van der Helm, "Haptic shared control improves

tele-operated task performance towards performance in direct control," in *2011 IEEE World Haptics Conference*, IEEE, 2011, pp. 433–438.

[23] B. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[24] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series.," in *KDD workshop*, Seattle, WA, vol. 10, 1994, pp. 359–370.

[25] M. P. Nemitz, R. J. Marcotte, M. E. Sayed, G. Ferrer, A. O. Hero, E. Olson, and A. A. Stokes, "Multi-functional sensing for swarm robots using time sequence classification: Hoverbot, an example," *Frontiers in Robotics and AI*, vol. 5, p. 55, 2018, ISSN: 2296-9144. DOI: 10.3389/frobt.2018.00055. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2018.00055.

[26] slaypni, *A python implementation of fastdtw*, 2019. [Online]. Available: https://github.com/slaypni/fastdtw.

[27] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[28] M. Kyrarini, M. A. Haseeb, D. Ristić-Durrant, and A. Gräser, "Robot learning of industrial assembly task via human demonstrations," *Autonomous Robots*, vol. 43, no. 1, pp. 239–257, 2019.

[29] V. E. Kremer, "Quaternions and slerp," *Embots. dfki. de/doc/seminar_ca/Kremer_Quaternions. pdf*, 2008.

[30] S. G. Hart and L. E. Staveland, "Development of nasa-tlx (task load index): Results of empirical and theoretical research," *Human mental workload*, vol. 1, no. 3, pp. 139–183, 1988.

[31] A. Pervez, A. Ali, J.-H. Ryu, and D. Lee, "Novel learning from demonstration approach for repetitive teleoperation tasks," in *2017 IEEE World Haptics Conference (WHC)*, IEEE, 2017, pp. 60–65.

[32] I. Havoutis and S. Calinon, "Learning from demonstration for semi-autonomous teleoperation," *Autonomous Robots*, vol. 43, no. 3, pp. 713–726, 2019.

[33] A. K. Tanwani and S. Calinon, "Learning robot manipulation tasks with task-parameterized semitied hidden semi-markov model," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 235–242, 2016.

[34] B. Akgun and K. Subramanian, "Robot learning from demonstration: Kinesthetic teaching vs. teleoperation," *Unpublished manuscript*, 2011.

[35] J. Hoyos, F. Prieto, G. Alenyà, and C. Torras, "Incremental learning of skills in a task-parameterized gaussian mixture model," *Journal of Intelligent & Robotic Systems*, vol. 82, no. 1, pp. 81–99, 2016.

[36] S. Calinon and A. Billard, "Incremental learning of gestures by imitation in a humanoid robot," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, ACM, 2007, pp. 255–262.

[37] R. Buchner, N. Mirnig, A. Weiss, and M. Tscheligi, *Evaluating in real life robotic environment: Bringing together research and practice*, Undetermined. DOI: 10.1109/ROMAN.2012.6343817.

[38] T. K. Kim and J. H. Park, "More about the basic assumptions of t-test: Normality and sample size," *Korean journal of anesthesiology*, vol. 72, no. 4, p. 331, 2019.

[39] R. Woolson, "Wilcoxon signed-rank test," *Wiley encyclopedia of clinical trials*, pp. 1–3, 2007.

[40] *Counterbalanced measures design - counterbalancing test groups*, https://explorable.com/counterbalanced-measures-design, (Accessed on 10/08/2020).

[41] A. Vahidi, A. Stefanopoulou, and H. Peng, "Recursive least squares with forgetting for online estimation of vehicle mass and road grade: Theory and experiments," *Vehicle System Dynamics*, vol. 43, no. 1, pp. 31–55, 2005. DOI: 10.1080/00423110412331290446. eprint: https://doi.org/10.1080/00423110412331290446. [Online]. Available: https://doi.org/10.1080/00423110412331290446.

[42] C. Paleologu, J. Benesty, and S. Ciochina, "A robust variable forgetting factor recursive least-squares algorithm for system identification," *IEEE Signal Processing Letters*, vol. 15, pp. 597–600, 2008.

[43] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[44] T. Lesort, *Continual learning: Tackling catastrophic forgetting in deep neural networks with replay processes*, 2020. arXiv: 2007.00487 [cs.LG].

[45] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2019.01.012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608019300231.

[46] K. Sirks, "Which position do i pick, if i don't know how to do the task?: A model to predict advantageous base poses for semi-autonomous robots.," 2018.

A

Conditioned-Probabilistic Movement Primitives

This section explains the theory and implemented extension of regular Probabilistic Movement Primitives (ProMP) including an evaluation of the implementation. The implementation is done in Python and a ROS wrapper is written around it to be able to interact with the model from other nodes (adding demonstrations, making predictions, resetting the model etc.).

Implementation

A first attempt was performed using an open source package which is an implementation of contextual ProMP [1]. Problems arised however when using this implementation, because it behaved unexpectedly. To show this behavior, a model was build using 6 demonstrations in the XZ plane. 3 out of these 6 trajectories are demonstrated with $x_{object} = 0.78$ and the other 3 out of 6 are demonstrated with $x_{object} = 0.94$m (see Figure A.1 and Figure A.2).

Figure A.1: Demonstrations performed for $x_{object} = 0.78$m

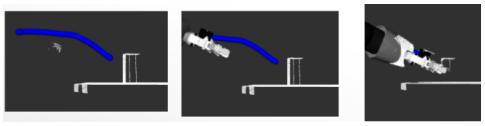


Figure A.2: Demonstrations performed for $x_{object} = 0.94$m

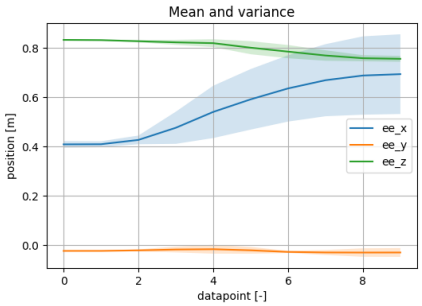


Figure A.3: Mean and variance of the demonstrations

When performing a prediction at $x_{object} = 0.91$m it is expected that at the last datapoint $y_{ee} \approx 0.8$, however when the prediction is inspected in Figure A.4 it can be seen that $y_{ee} \approx 0.48$ instead. Therefore we proceeded to test this implementation on a basic theoretical problem: linear functions.

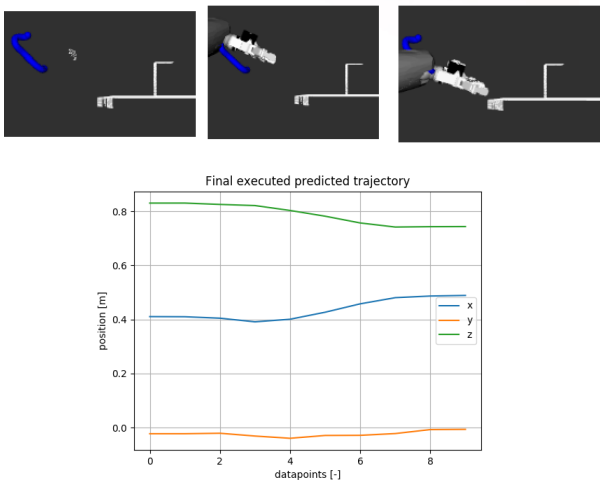


Figure A.4: Prediction at x_{object} = 0.91m

5 different input/output pairs were used to train the ProMP model from [1]. The input has dimension 1 and the output is a linear function, of which an illustration is depicted in left image of Figure A.5. Furthermore when making a prediction at input = 1.0, it is expected that the output starts at 1.0 and stops at 2.0 within timestep 0 until 1.0. Instead the linear output starts at 0.0 and stops at 1.0, which can be seen in the right image of Figure A.5. This is incorrect, and further investigation of the code showed that this is because "conditioning" is implemented as a via point at the last datapoint. This means that when the trajectory is "conditioned" on 1.0, it actually means that the last datapoint of the trajectory has to go through 1.0. This is not what we want, since the condition is not necessarily the goal position of the end effector. Hence we want to make the input/output coupling with an arbitrary input, and do not force it to go through this input.

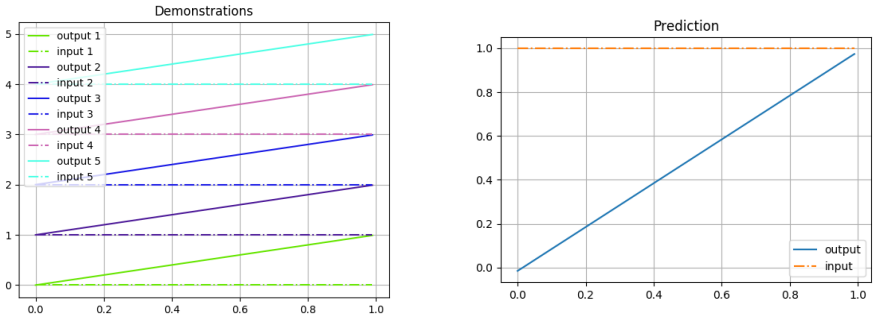


Figure A.5: Left: Input/Output pairs put into the model for evaluation, Right: Prediction with input = 1.0

Conditioning

To achieve this, [2] was used as a code base, since this was also used by [1]. This basis is extended using [3]. Each demonstration $\vec{\tau}_M$ is resampled to contain n amount of datapoints, after that $\vec{\tau}_M = \vec{y}_M$ is added to the $\mathbf{Y}$ matrix vertically:

$$\mathbf{Y} = \begin{bmatrix} \vec{y}_1 & \vec{y}_2 & \vec{y}_M \end{bmatrix} \tag{A.1}$$

Then $\vec{w}$ is constructed, after which $\vec{w}_M$ is added to the $\mathbf{W}$ matrix. The condition $\vec{s}$ of demonstration M is stacked vertically in the $\mathbf{S}$ matrix:

$$\mathbf{S} = \begin{bmatrix} \vec{s}_1 & \vec{s}_2 & \vec{s}_M \end{bmatrix} \tag{A.2}$$

After that the cross covariance matrix Σ_{joint} is determined by computing the covariance between $\mathbf{W}$ and $\mathbf{C}$ using NUMPY.COV or Welford's online update method. Σ_{ww}, Σ_{sw}, Σ_{ws} and Σ_{ss} are determined as follows:

$$\Sigma_{ww} = \Sigma_{joint}[1:N,\ 1:N] \tag{A.3}$$
$$\Sigma_{sw} = \Sigma_{joint}[N:,\ 1:N] \tag{A.4}$$
$$\Sigma_{ws} = \Sigma_{joint}[1:N,\ N:] \tag{A.5}$$
$$\Sigma_{sw} = \Sigma_{joint}[N:,\ N:] \tag{A.6}$$

Furthermore $\vec{\mu}_{\vec{w}}$ and $\vec{\mu}_{\vec{s}}$ are determined by calculating the mean over $\mathbf{W}$ and $\mathbf{C}$ respectively using NUMPY.MEAN or using Welford's online update method. These vectors are concatenated to get the total mean vector $\vec{\mu}_{joint}$. When making a prediction, these means vectors and covariance matrices are used to create the mean prediction according to the context $\vec{\mu}_{\vec{\tau}|s}$. This is done by creating a new function GENERATETRAJECTORY(CONDITION), which contents are based on what is discussed in the conditioned-ProMP section of the paper.

Listing A.1: Python function used to generate a trajectory according to a certain condition

```
1  def generateTrajectory(condition):
2          noise = np.eye(self.sigma_cc.shape[0]) * self.sigma
3
4          mu_w_given_s = self.mean_w +
5          np.dot(np.dot(self.sigma_ws, np.linalg.inv(self.sigma_ss + noise)), condition - self.mean_s)
6
7          sigma_w_given_s = self.sigma_ww -
8          np.dot(np.dot(self.sigma_ws, np.linalg.inv(self.sigma_ss + noise)), self.sigma_sw)
9
10         mu_traj_given_s = np.dot(self.Phi.T, mu_w_given_s)
11
12         sigma_traj_given_s = np.dot(self.sigma ** 2, np.eye(self.num_samples)) +
13         np.dot(np.dot(self.Phi.T, sigma_w_given_s), self.Phi)
14
15 return mu_traj_given_s
```

Line 2 represents the noise term:

$$\Sigma_{\vec{s}\vec{s}} = \Sigma_{\vec{s}\vec{s}} + \Sigma_n \tag{A.7}$$

Line 4/5 and 7/8 represent the mean and covariance over the weights given a certain condition:

$$\vec{\mu}_{\vec{w}|\vec{s}} = \vec{\mu}_{\vec{w}} + \Sigma_{\vec{w}\vec{s}}\Sigma_{\vec{s}\vec{s}}^{-1}(\vec{s} - \vec{\mu}_{\vec{s}}), \tag{A.8}$$
$$\Sigma_{\vec{w}|\vec{s}} = \Sigma_{\vec{w}\vec{w}} - \Sigma_{\vec{w}\vec{s}}\Sigma_{\vec{s}\vec{s}}^{-1}\Sigma_{\vec{s}\vec{w}} \tag{A.9}$$

Line 10 represents the trajectory given a certain condition, and is thus used as output of the function:

$$\vec{\mu}_{\vec{\tau}|\vec{s}} = \Phi\vec{\mu}_{\vec{w}|\vec{s}} \tag{A.10}$$
$$\tag{A.11}$$

Line 12/13 represents the uncertainty in the prediction:

$$\Sigma_{\vec{\tau}|\vec{s}} = \sigma^2\mathbf{I}_{T\times T} + \Phi\Sigma_{\vec{w}|\vec{s}}\Phi^T \tag{A.12}$$

Evaluation

Evaluations are performed on this conditioned-ProMP implementation on basic 2D theoretical problems to evaluate its core functionality. These problems consist of a linear, non-linear, multiple input and multiple output model. After that a quick evaluation of the conditioned ProMP model is performed on the robot simulation environment.

Single context

A linear problem was constructed, which means that the demonstrations provided are linear functions of the form:

$$y[i] = ax + b[i] \tag{A.13}$$

$$= 1x + b[i], \quad \text{where} \quad b = [1, 2 \ldots 5] \tag{A.14}$$

Where the b value is varied across the demonstrations from 1 till 5, and a is kept at the constant value of 1. $\vec{s}$ is one dimensional and is set to the corresponding value of b. Both the linear output functions as $\vec{s}$ are plotted on the left in Figure A.6. When we generalize towards $\vec{s} = 1$, it is expected that the output will be of the form $y = x + 1$, being a linear function starting at 1 and ending at 2. Assessing the right figure in Figure A.6, it can be seen that this is indeed the case.

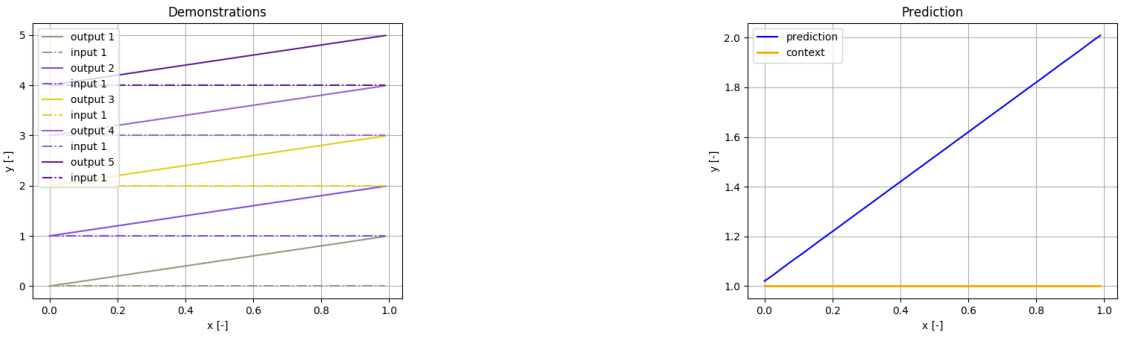


Figure A.6: Left: Demonstrations provided, Right: Prediction with condition/context = 1.0

Linear problem: 3D context

This single input single output linear model is extended by providing a 3 dimensional input vector, which is equal to $[b, b, b]$, thus having the same value for each dimension per demonstration. These demonstrations are depicted in Figure A.7. When generalizing towards $\vec{s} = [1.0, 1.0, 1.0]$ it can be seen that the correct trajectory is generated.

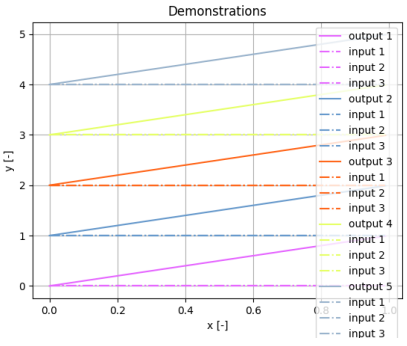


Figure A.7: Demonstrations used in the linear theoretical problem

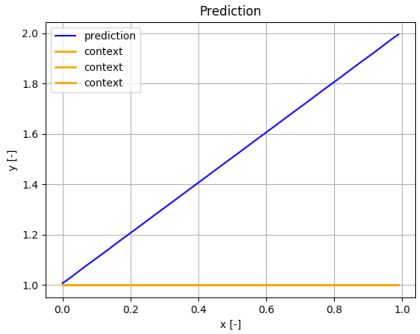


Figure A.8: Prediction with condition/context = 1.0

Linear problem: 3D output

Further evaluation of the model is done by providing single input multiple output demonstrations, as can be seen in Figure A.9. Again in Figure A.10 each of the demonstrated output is correctly generalized at $\vec{s} = 1.0$.

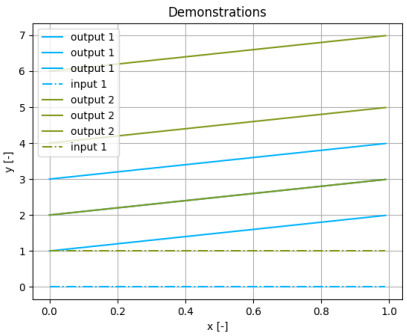


Figure A.9: 3D demonstrations used to build the probabilistic model

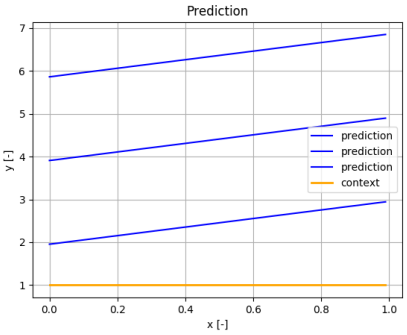


Figure A.10: Prediction with context = 1.0

Non-linear problem: sine waves

In addition to a linear problem, a non-linear model has been evaluated, using sine functions. These functions are generated using the following equation:

$$y[i] = \sin(ax + b[i]) \tag{A.15}$$

$$= \sin(10x + b[i]), \quad \text{where} \quad b = [1, 2 \ldots 5] \tag{A.16}$$

The condition corresponding to the trajectory is again equal to b, and these demonstrations are depicted in Figure A.11. When generalizing towards $\vec{s} = 1.0$, it can be seen in Figure A.12 that the correct sine function is predicted. Hence it can be concluded that the built extension of PROMPLIB to condition the trajectory on

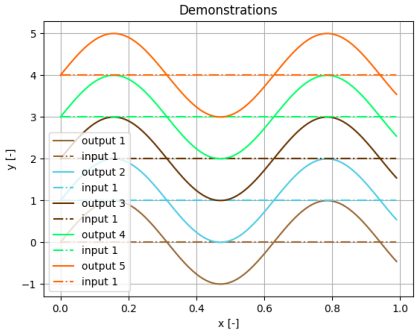


Figure A.11: Demonstrations used in the non-linear theoretical problem

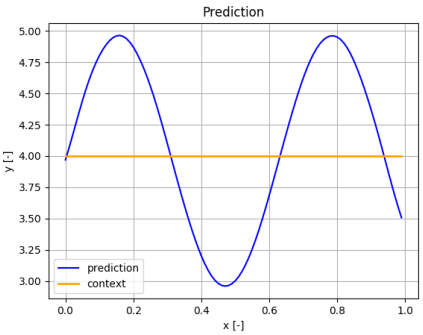


Figure A.12: Prediction with context = 1.0

an input works for basic examples, and can further be evaluated on more complex problems.

Robot simulation environment

In order to evaluate conditioned ProMP on the simulated real world, a general model has been build, which is evaluated in different planes. Demonstrations have been provided for different initial poses as well as different object positions. The raw demonstrations of the end effector position and orientation are depicted in Figure A.13, after which they are resampled to contain only 10 datapoints, shown in Figure A.14. Note that in Figure A.14, only the first and last quaternion datapoint are interpolated, since in [4] this was experienced to be good enough. In this application the opposite was true, since the orientation of the gripper as function of time is crucial to successfully avoid collision with the environment. Therefore in later versions, the complete quaternion trajectories are taken into acount when resampling. After downsampling the trajectories, they

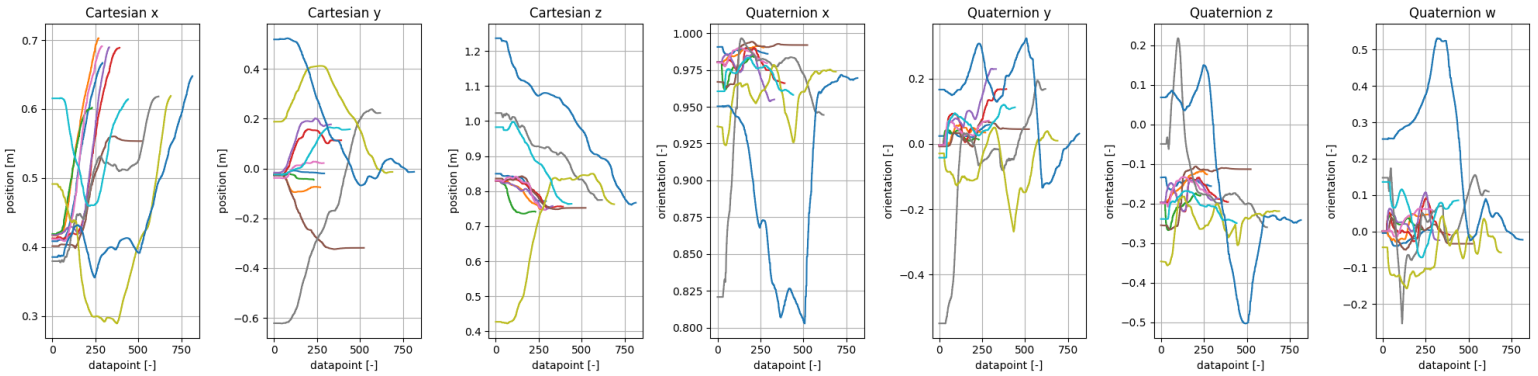


Figure A.13: Raw demonstrations used to evaluate the model

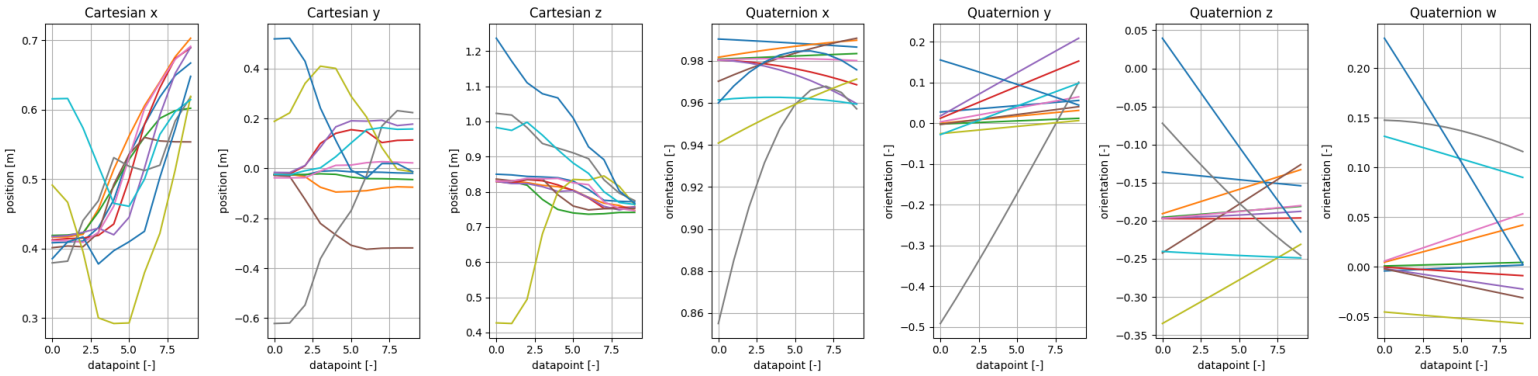


Figure A.14: Downsampled raw demonstrations to 10 datapoints

are converted to be relative to the object position. After that the reference trajectory is determined for each condition, and this trajectory is aligned with the trajectory most similar to the reference using DTW. When there is only one trajectory per condition, this DTW step is not executed. This is the final data preprocessing step before the trajectories are used as input to the conditional ProMP model. An illustration of this data is shown in Figure A.15.

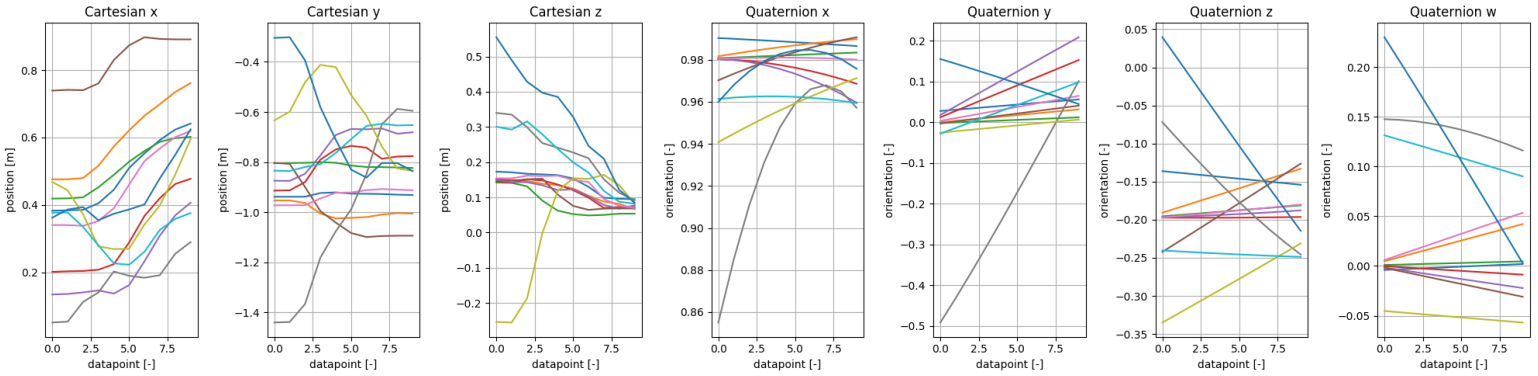


Figure A.15: Relative and aligned trajectories used as input to the probabilistic model

Using this model, the generalization towards different object positions have been evaluated. An illustration of such generalization in different planes is depicted in Figure A.16 in addition to different initial end effector poses.

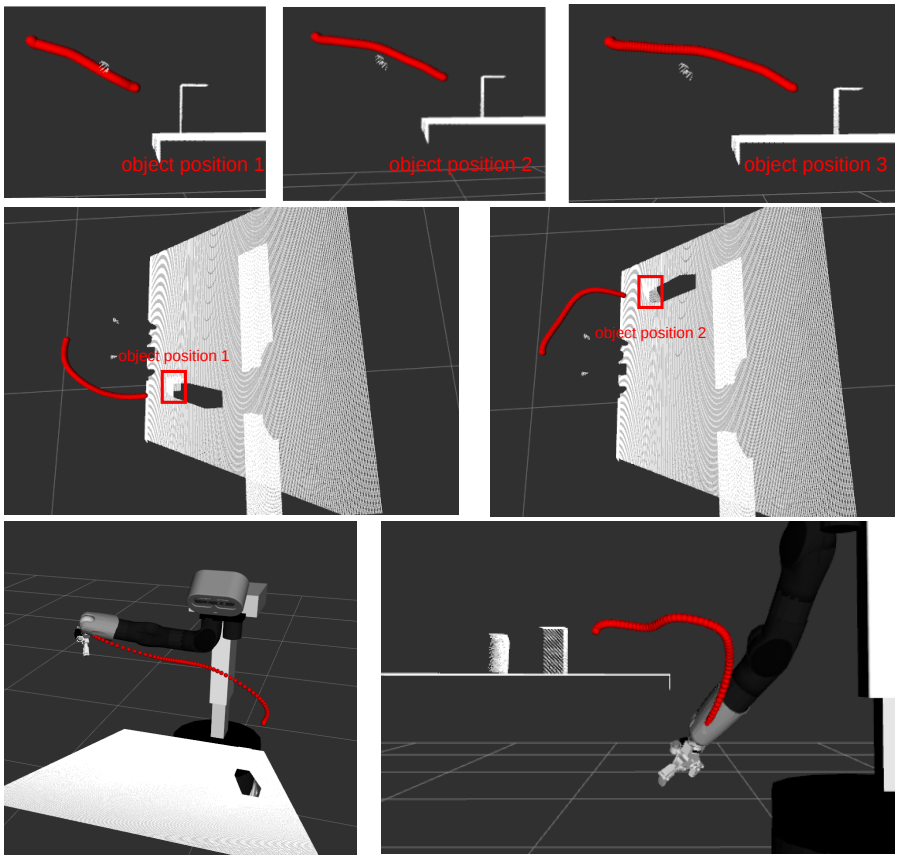


Figure A.16: Generalizations towards different object position in one plane

B

Online learning

This section shows the evaluations performed on the complete online learning framework, both on a 2D problem and on the simulated environment of the robot.

2D example

Reaching viapoints

After adapting the initial model for 4 different conditions, each condition is reevaluated on the performance of reaching the viapoints, of which the results are depicted in Figure B.1.

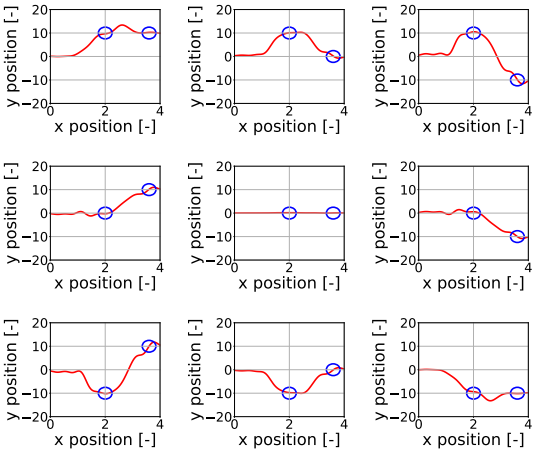


Figure B.1: Reevaluation of the adapted model for each condition, where it can be seen that for each condition the viapoints are successfully reached

Additionally an evaluation has been performed on the generalization capabilities of the adapted model that reaches the viapoints for all 9 conditions, of which an illustration is depicted in Figure B.2. In this illustration it can be seen that the conditions are varied from -20 to +20 and from -5 to +5, and that the trajectory successfully reaches each condition.

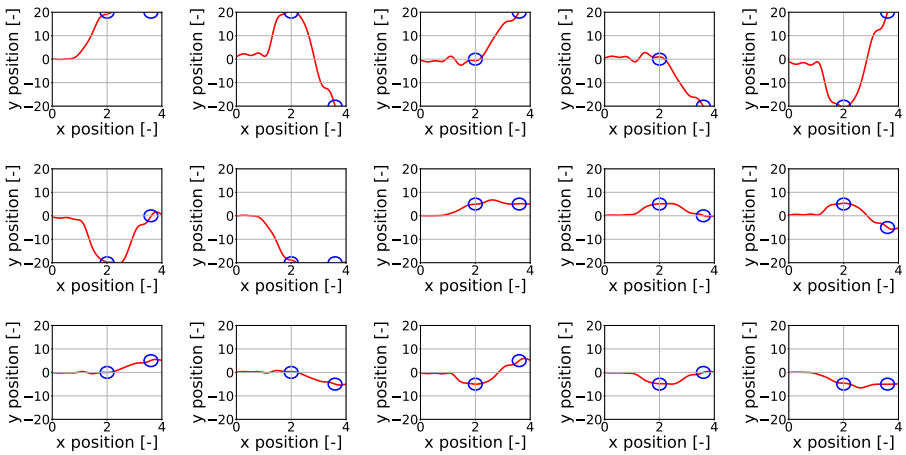


Figure B.2: Generalizations performed on previously unseen viapoints, varying from -20 to +20 and -5 to +5.

Avoiding obstacle

In Figure B.3 the demonstrations used to train the initial model are depicted. After adapting the initial model

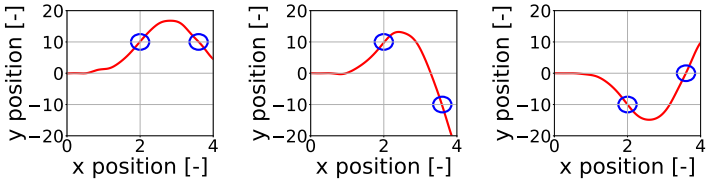


Figure B.3: Demonstrations used to train the initial model for the 2D problem with an obstacle. The conditions are [10, 10], [10, -10] and [-10, 0] respectively.

to avoid the obstacle, each condition is reevaluated on the performance of avoiding this obstacle while still reaching the viapoints, which is depicted in Figure B.4.

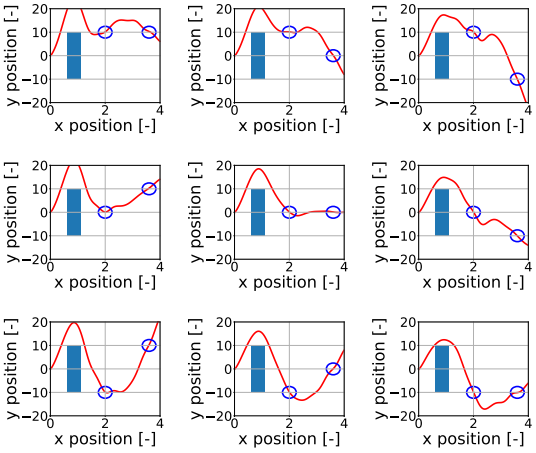


Figure B.4: Reevaluation of the adapted model for each condition, where it can be seen that for each condition the trajectory does not collide with the obstacle

Furthermore in Figure B.5 generalizations have been performed on previously unseen conditions, where it can be seen that for all conditions except [-20, -20] and [0, -20], the obstacle is successfully avoided while reaching the viapoints.

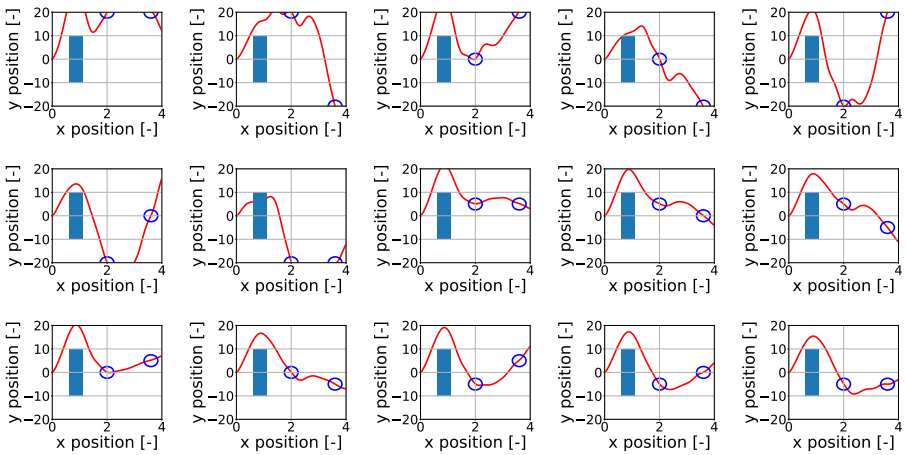


Figure B.5: Generalizations performed on previously unseen conditions, varying from -20 to +20 and -5 to +5.

Graphical User Interface

A GUI was build to control the workflow of this experiment, of which an illustration is depicted in Figure B.6.

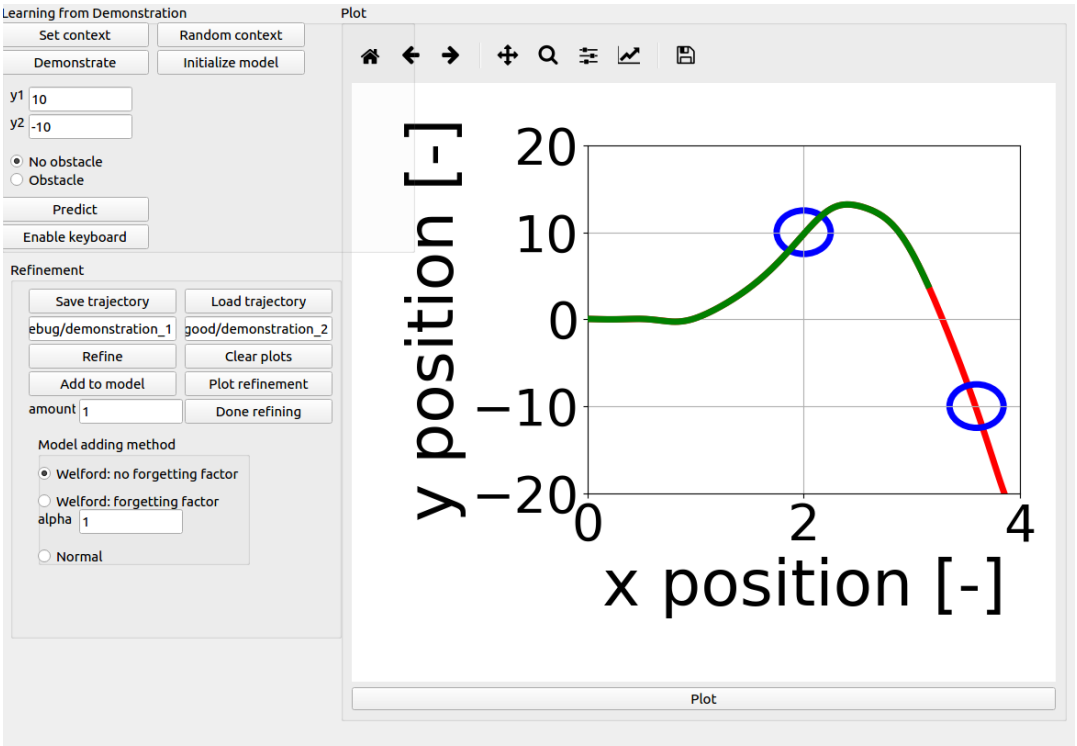


Figure B.6: GUI build to control the workflow of the 2D experiment. The current position of the refined trajectory is shown as well as the predicted trajectory.

Robot simulation environment

Furthermore the online learning framework has been evaluated on the simulated real world, where the predicted trajectory is refined such that it is successfully able to reach the object and avoid an obstacle. The plots have been generated using a build GUI, which is also used to debug the whole system (Figure B.7)

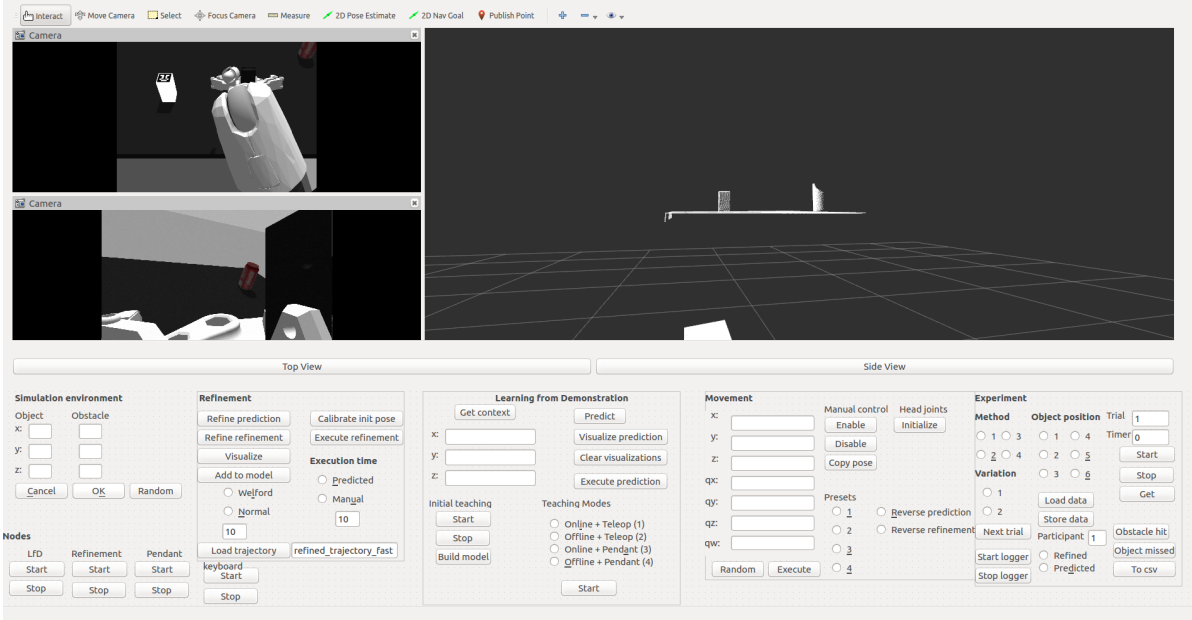


Figure B.7: GUI used to generate the plots in 3D and to debug the whole system

Furthermore the DTW step is evaluated, of which an illustration is depicted in Figure B.8. Here it can be seen that the prediction and refinement have successfully been aligned, such that we can subtract them.

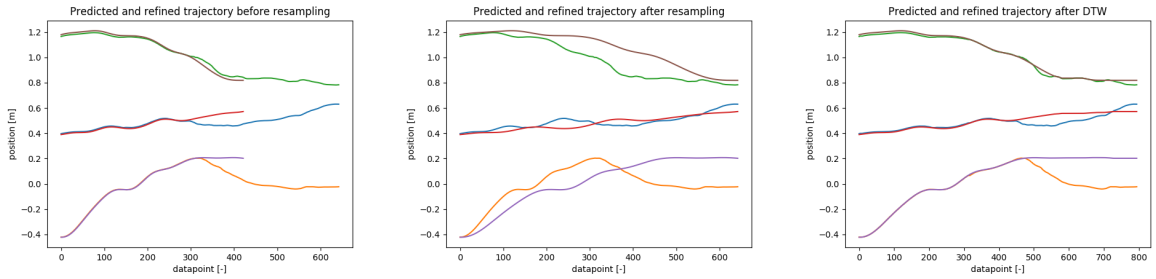


Figure B.8: Brown/Green is the z position of the prediction and refinement, Red/Blue is the x position and Purple/Yellow is the y position

Table analysis: Initial model

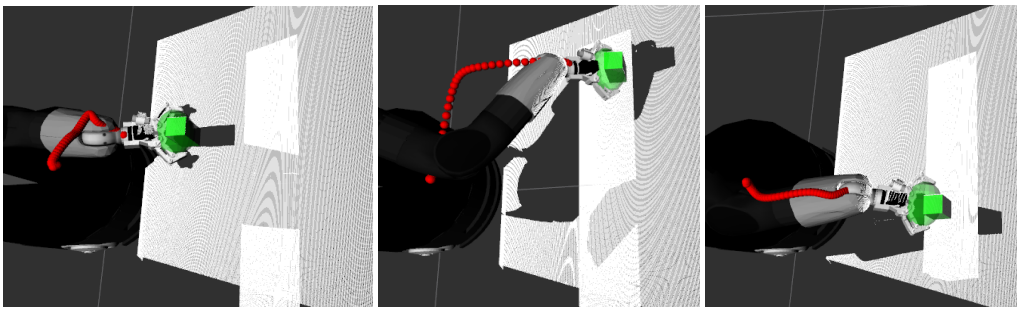


Figure B.9: Initial demonstrations for the table environment that satisfy the success criteria, for object positions ${}^{base}[0.6,\ 0.0,\ 0.9]$, ${}^{base}[0.8,\ 0.2,\ 0.9]$ and ${}^{base}[0.8, -0.2,\ 0.9]$ respectively, and the initial pose is fixed at ^{base}[0.37, -0.14, 0.88, 0.99, 0, 0.15, 0.030].

Table analysis: Second iteration

After adapting the initially trained model, each object position is evaluated again on the success criteria. In Figure B.10 it can be seen that for each object position, the object is successfully reached without kicking it over and without collision with the table. Hence the adaptation has been successful.

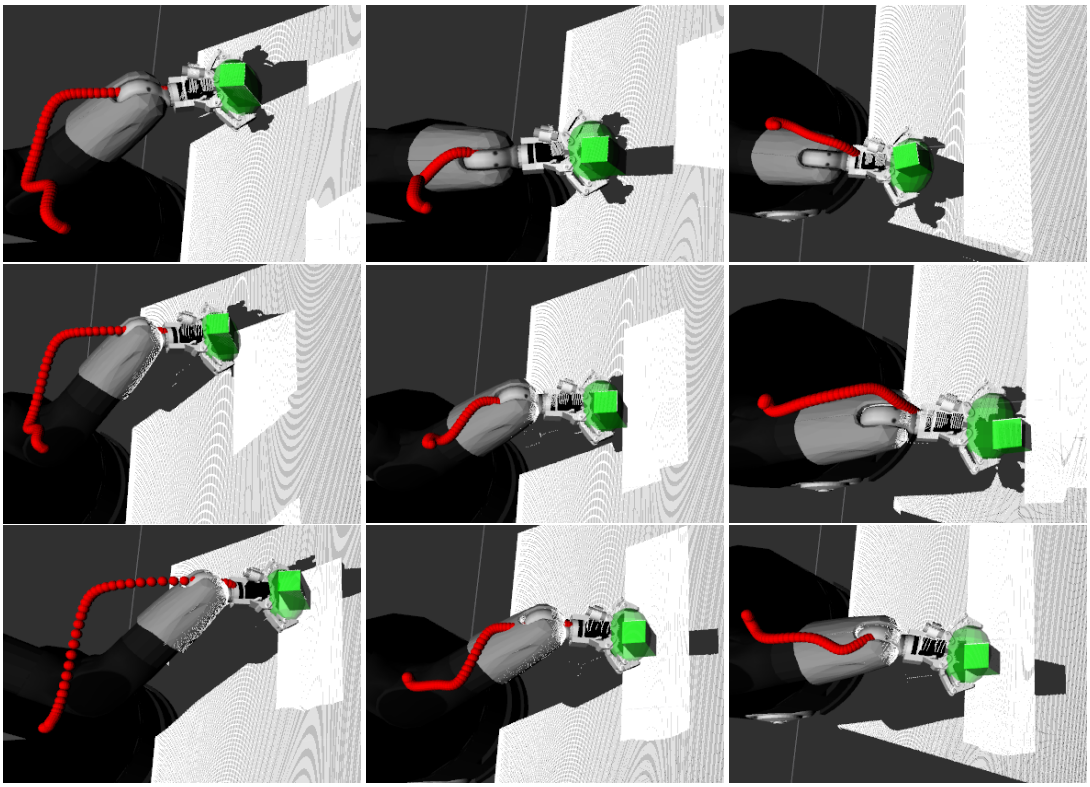


Figure B.10: Iterating a second time over the adapted model, where it can be seen that for each object position the object is reached without kicking it over and without collision with the table.

Dishwasher analysis 1: Initial model evaluation

The initial model trained on the dishwasher for basket position b_1 has been evaluated on all 4 object positions. The initial demonstrations are depicted in Figure B.11, and the generalizations across the other two conditions are shown in Figure B.12.

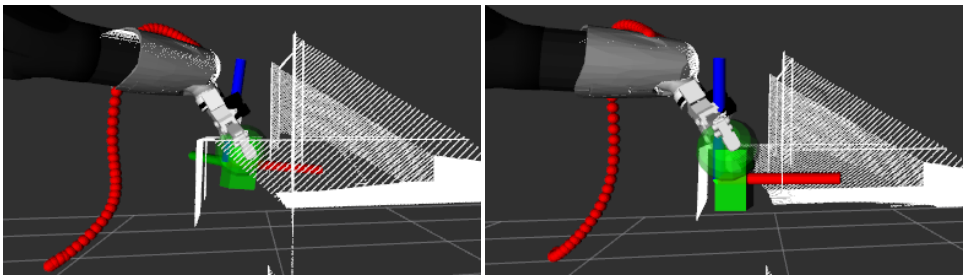


Figure B.11: Initial demonstrations used to train the model. Left: object position [0.8, 0.2, 0.9], Right: object position [0.8, -0.1, 0.9]

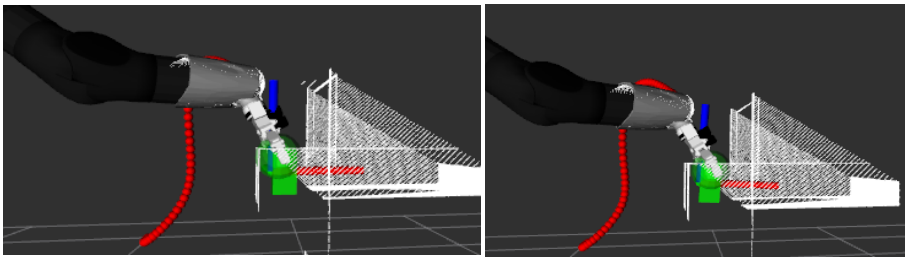


Figure B.12: Generalizations across the other two object positions of this trained model. Left: object position [0.8, 0.0, 0.9], Right: object position [0.8, 0.1, 0.9]

Dishwasher analysis 1: Second iteration

After adapting the model for basket position b_2, the adapted model is evaluated again on each of the 4 object positions. In Figure B.13 this second evaluation is depicted, where it can be seen that the model generates predictions that meet the success criteria for each of the 4 object positions.

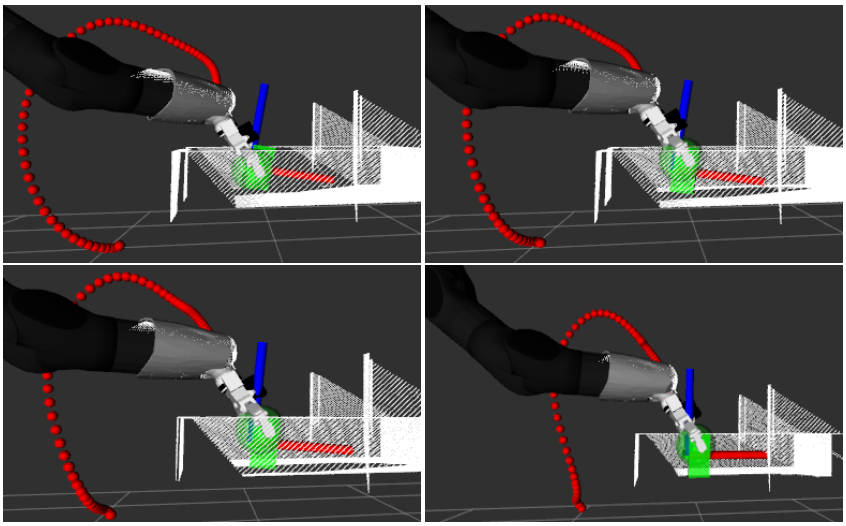


Figure B.13: Evaluations of the predictions for all object positions on the success criteria. Here it can be seen that for each object position the object is reached without kicking it over and without collision with the environment.

Dishwasher analysis 2: Initial model

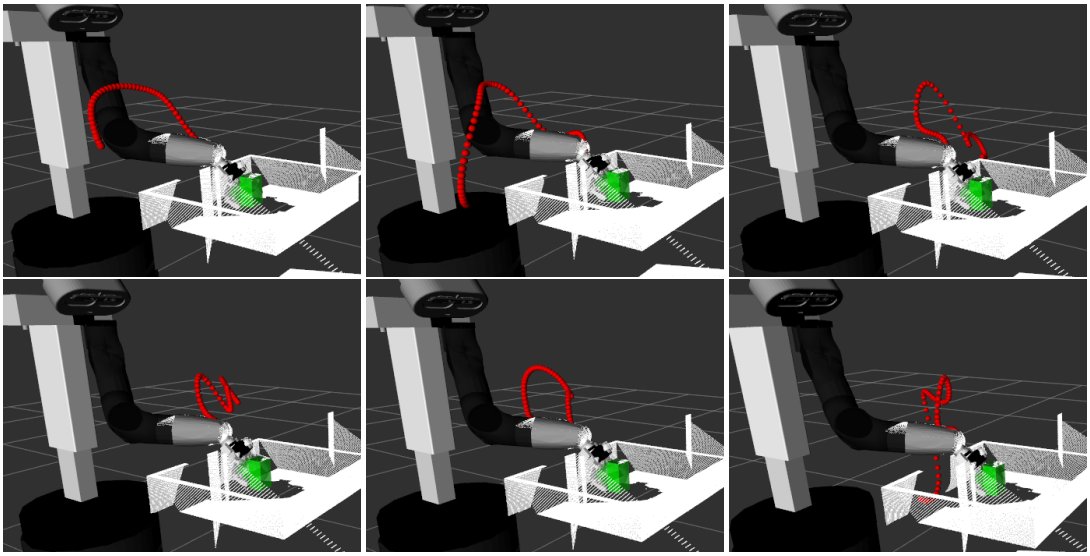


Figure B.14: Demonstrations used to train an initial model for different initial poses for basket position b_1, where the final state of the trajectory is shown to be able to see that it satisfies the success criteria. The object position is fixed at ${}^{base}[0.1,\ 0.8,\ 0.9]$.

Dishwasher analysis 2: Second iteration

After adapting the model, the initial poses where the initial model was trained on are evaluated a second time, of which an illustration is shown in Figure B.15. Here it can be seen that these predictions still satisfy the success criteria, and we did not negatively influence these predictions by adapting the model for the other initial poses.

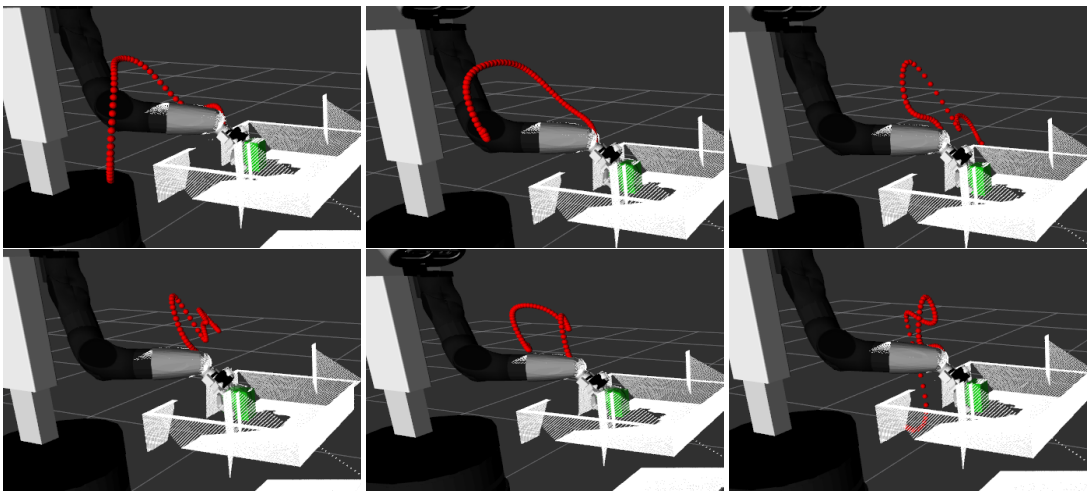


Figure B.15: Second iteration of the demonstrations that were used to train the initial model, after adapting the model. As can be seen, the predictions for these initial poses still satisfy the success criteria.

The second iteration over the other initial poses is depicted in Figure B.16. Here it can be seen that the first two predictions (top left and top middle) are successfully adapted. The rest of the predictions are the same as in the paper.

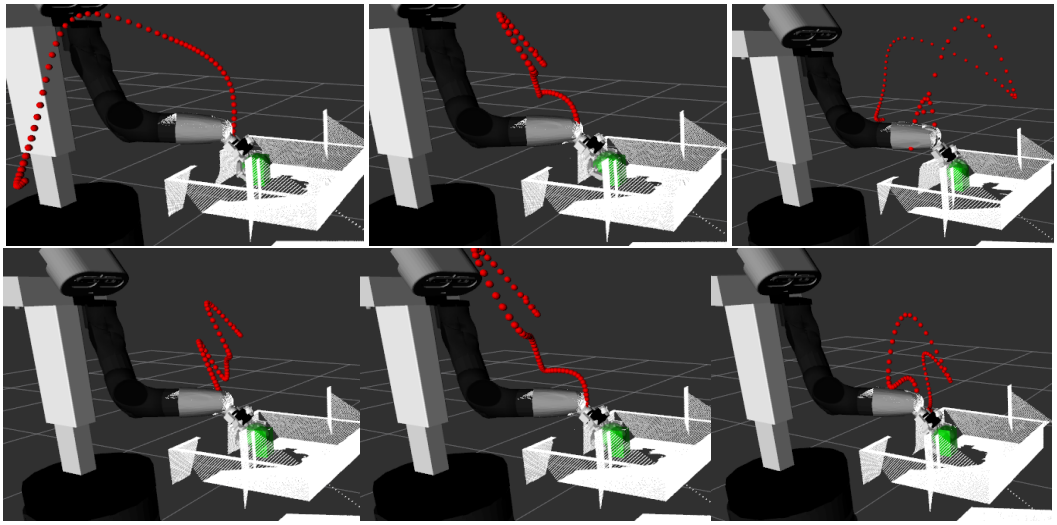


Figure B.16: Second iteration over the initial poses that were used to adapt the model. Only the top left and top middle are different from what was shown in the paper, since these are the predictions where refinements were needed.

Welford vs. Normal

When normally adding a demonstration to the conditioned-ProMP model, the weight vector $\vec{w}_M$ used to model the demonstration $\vec{\tau}_M$ is appended to the weight matrix:

$$\mathbf{W} = \begin{bmatrix} \vec{w}_1 & \vec{w}_2 & \dots & \vec{w}_M \end{bmatrix} \tag{B.1}$$

After that using their corresponding conditions, the conditional probability distribution parameters $\vec{\mu}_{\vec{\tau}|\vec{s}}$ and $\Sigma_{\vec{w}|\vec{s}}$ are recalculated using this added demonstration $\vec{\tau}_M$.

When using Welford's method, the conditional mean and covariance is changed incrementally. This means that one data point is used to update the mean and covariance, rather than using all the previous data (batch).

$$\vec{\mu}_{new} = \vec{\mu}_{old} + \frac{1}{M}(\vec{x} - \vec{\mu}_{old}) \tag{B.2}$$

$$\Psi_{new} = \Psi_{old} + \frac{(M-1)}{M}(\vec{x} - \vec{\mu}_{old})^T(\vec{x} - \vec{\mu}_{old}) \tag{B.3}$$

$$\text{with} \quad \Sigma = \frac{\Psi}{M-1}$$

A comparison between both the normal and implemented Welford computation of the conditional mean and covariance is depicted in Figure B.17. From these graphs it can be seen that small differences are present in both the mean and covariance, but these were experienced to be non-problematic in this application.

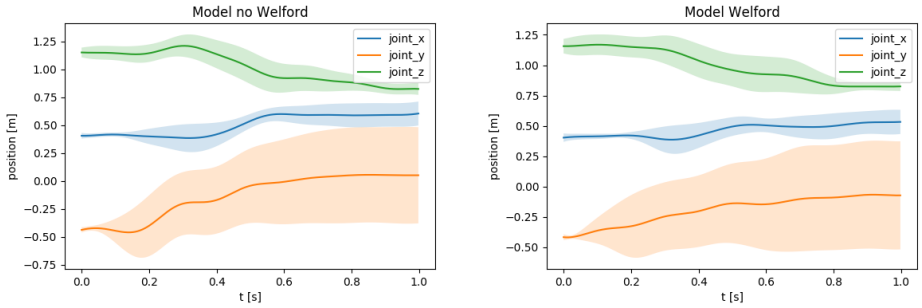


Figure B.17: Left: Computation of mean and covariance using the normal computation, right: Updating mean and covariance using Welford

Current vs. next time step adaptation

A high level overview of the difference in implementation of adapting the next and the current time step is depicted in Figure B.18.

Figure B.18: Left: Adaptation of the next time step, Right: Adaptation of the current time step. The green and red dots represent the master and slave position respectively, and when they are on the same position, the dot is indicated as a brown color. The white dot represents the reference trajectory, where the executed trajectory converges to if the operator does not apply any input.

The difference in the vectors used to calculate the adaptation is visualized in Figure B.19. As described in the paper, adaptation of the next time step is done by calculating the position vector of step $i+1$ w.r.t. step i, respectively called the *next* frame and *current* frame:

$$^{current}\vec{p}_{next} = {}^{current}R_{base}({}^{base}\vec{p}_{next} - {}^{base}\vec{p}_{current}) \tag{B.4}$$

$$\text{where } \vec{p} = [x,\ y,\ z]^T \text{ and } {}^{current}R_{base} = \mathbf{I}_{3\times3}$$

The master device position is normalized such that the zero position is approximately in the middle of the workspace, which can be seen in Figure B.18. Therefore to get the position of the human intervention (green dot in Figure B.18) combined with the executed position, we need to add this normalized master position $^{current}\vec{p}_{master}$ to $^{current}\vec{p}_{next}$:

$$^{current}\vec{p}_{next,new} = {}^{current}\vec{p}_{next} + {}^{current}\vec{p}_{master} \tag{B.5}$$

The end effector position expressed in *base* frame is then send to the inverse kinematics of the robot:

$$^{base}\vec{p}_{next,new} = {}^{base}\vec{p}_{current} + {}^{base}R_{current}\,{}^{current}\vec{p}_{next,new} \tag{B.6}$$

When adapting the current time step on the other hand, we directly change the value of $^{base}\vec{p}_{current}$, by adding to it the master position $^{base}\vec{p}_{master}$:

$$^{base}\vec{p}_{current,new} = {}^{base}\vec{p}_{current} + {}^{base}R_{current}\,{}^{current}\vec{p}_{master} \tag{B.7}$$

$$\text{where } {}^{base}R_{current} = \mathbf{I}_{3\times3} \tag{B.8}$$

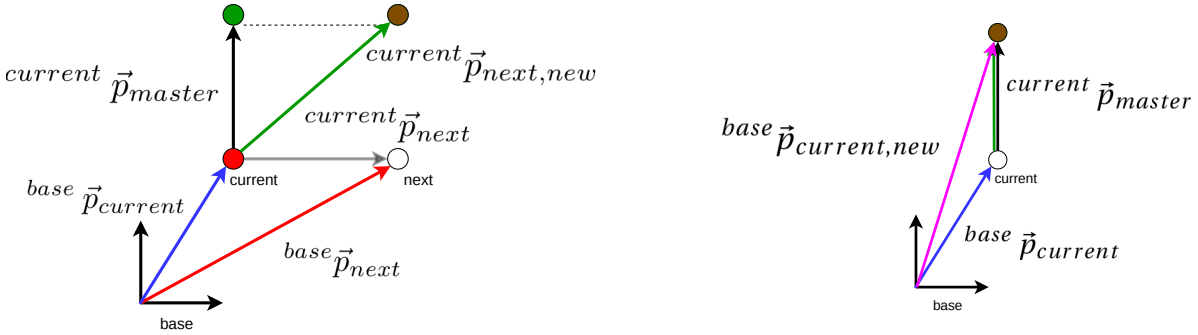


Figure B.19: Visualization of the vectors used to calculate the adaptation of the executed trajectory

Shifted initial position

When too little initial demonstrations were used to train the model, it has to extrapolate too much when either the object position or the initial pose is significantly different from the initial demonstrations. An illustration of what happens in this situation is depicted in Figure B.20, where it can be seen that the initial position of the prediction is shifted. This problem is prevented by using using enough initial demonstrations for different conditions.

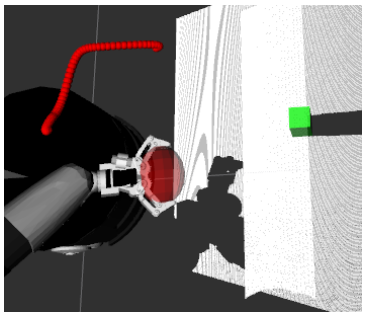


Figure B.20: Illustration of what happens when making predictions for a new object position when too little initial demonstrations are used to train the model.

Dynamic Movement Primitives

Before using ProMP, the online learning procedure was evaluated using Dynamic Movement Primitives (DMP), since a working ROS package was available [5]. In DMP, a trajectory demonstration is modeled using a second order point attractor differential equation:

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}) + f, \tag{B.9}$$

where $\vec{y}$ is the end effector state, $\vec{g}$ is the goal state, and α and β are gain terms. In order for the system to be easily solvable and generalizable, a relatively simple additional dynamic system is defined:

$$\dot{x} = -\alpha_x x, \tag{B.10}$$

which is used for a definition of the forcing function:

$$f(x, g) = \frac{\sum_{i=1}^{N} \psi_i w_i}{\sum_{i=1}^{N} \psi_i} \vec{x}(g - y_0), \tag{B.11}$$

where the $x(g - y_0)$ (y_0 is the intial position) term is necessary to be able to ensure convergence of the forcing term to zero, spatially and temporally scaling of the trajectory. The convergence of the forcing term to zero when time goes to infinity is necessary because the canonical system does also converge to zero. This means that the trajectory can take any form, but will always return to the goal point when time goes to infinity.

Zooming in on the ψ_i function from Equation (B.11), it can be seen that it represents a Gaussian centered at c_i and variance h_i:

$$\psi_i = \exp\left(-h_i(x - c_i)^2\right), \tag{B.12}$$

where w_i are the weights given to each of the Gaussians ψ_i, resulting in the forcing function being a set of Gaussians that are activated as the canonical system converges to the target. An illustration of this process is shown in Figure B.21, where different weights are given to each Gaussian. Due to these different weights, the trajectory can be shaped according to the demonstrated trajectory.

These weights are calculated using a regression method, which minimizes the difference between the desired trajectory acceleration $\ddot{y}$ and the acceleration of the base point attractor differential equation depicted in Equation (B.9):

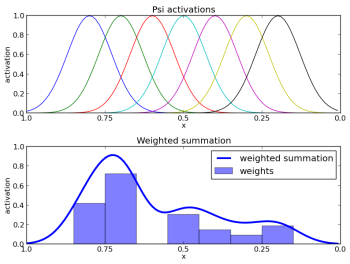


Figure B.21: Illustration of the generation of the forcing function, source: [6]

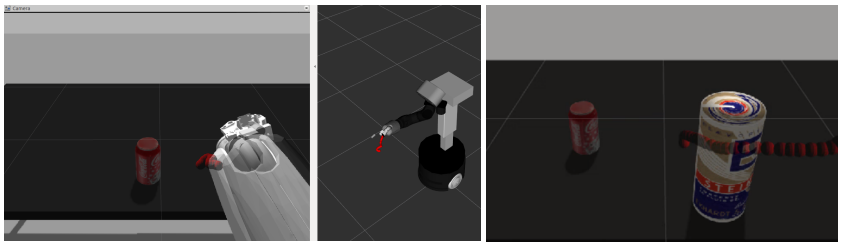


Figure B.22: Prediction made using DMP, where the goal position is hardcoded

$$\vec{f}_d = \ddot{\vec{y}} - \alpha_y(\beta_y(\vec{g} - \vec{y}) - \dot{y}) \tag{B.13}$$

$$\frac{\sum_{i=1}^{N} \psi_i w_i}{\sum_{i=1}^{N} \psi_i} \vec{x}(g - y_0) = \ddot{\vec{y}} - \alpha_y(\beta_y(\vec{g} - \vec{y}) - \dot{y}) \tag{B.14}$$

Now we can rewrite this set of equations in matrix form:

$$\frac{\vec{\psi}\,\vec{w}}{\sum_{i=1}^{N} \psi_i} \vec{x}(g - y_0) = \ddot{\vec{y}} - \alpha_y(\beta_y(\vec{g} - \vec{y}) - \dot{y}) \tag{B.15}$$

$$\mathbf{X}\vec{w} = \vec{f} \tag{B.16}$$

Here $\mathbf{X}$ and $\vec{f}$ are defined as:

$$\mathbf{X} = \begin{bmatrix} \frac{\psi_1}{\sum_{i=1}^{N} \psi_i} x_1 & \cdots & \frac{\psi_N}{\sum_{i=1}^{N} \psi_i} x_1 \\ \cdots & \cdots & \cdots \\ \frac{\psi_1}{\sum_{i=1}^{N} \psi_i} x_T & \cdots & \frac{\psi_N}{\sum_{i=1}^{N} \psi_i} x_T \end{bmatrix}, \qquad \vec{f} = \ddot{\vec{y}} - \alpha_y(\beta_y(\vec{g} - \vec{y}) - \dot{y}) \tag{B.17}$$

The linear equation depicted in Equation (B.16) is solved in the state-of-the-art DMP literature via either Locally Weighted Regression (LWR), Locally Weighted Projection Regression (LWPR) or Receptive Field Weighted Regression (RFWR). Where LWR is a batch learning method and RFWR and LWPR are based on LWR to also allow incremental updates of the parameters, which are the most commonly used incremental regression techniques [7].

Two predictions using DMP are depicted in Figure B.22, where the goal pose $\vec{g}$ is hardcoded. This goal pose can also be extracted using an object detection module, but this always requires the object to be completely known, and DMP will fail otherwise.

Slow adaptation

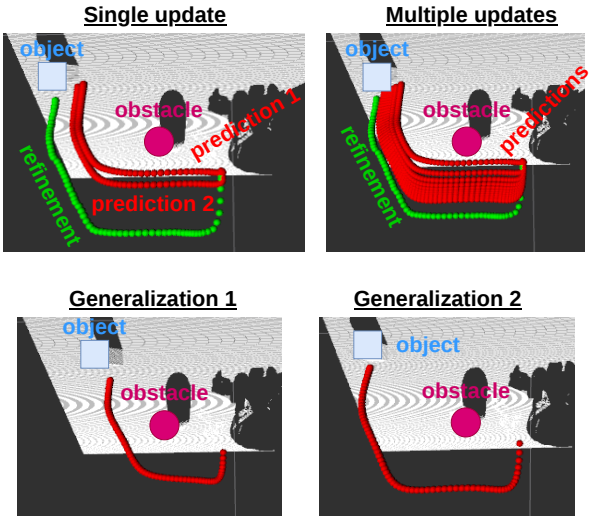


Figure B.23: Illustration of the amount of updates needed when an initial model is trained with more trajectories. Here the model needs to be updated ≈ 11 times with the same refined trajectory in order to change the prediction to avoid the obstacle

C

Initial demonstrations: Interactive coupling/decoupling

The demonstration data used as input to the model are the Cartesian end effector position and the orientation in quaternions, thus $\vec{y} = \begin{bmatrix} x & y & z & qx & qy & qz & qw \end{bmatrix}^T_{ee}$ and the condition is the object position relative to the end effector at $t = 0$: $\vec{s} = \begin{bmatrix} x & y & z \end{bmatrix}^T_{object}$, multiplied by a factor 10. To conveniently provide the initial demonstrations using the 7 degrees of freedom (DoF) manipulator of the care robot, the Phantom Omni was used as teaching device, of which an illustration is depicted in Figure C.1. This device is used to control the end effector position and orientation (6DoF), and the whole body controller (WBC) uses an optimization algorithm to determine the joint positions (7DoF) [8].

Figure C.1: Phantom Omni device used to generate the initial demonstrations

On the real robot, the master and slave are directly coupled, and force feedback moves the master device to the scaled current slave position. This means that the master and slave will always be aligned. However in simulation the force feedback did not work, which means that this alignment does not happen automatically. This results in the slave jumping to the scaled master position when the master is coupled, which produces undesirable behavior due to large overshoots. This made it challenging to properly create initial demonstrations, which is why an interactive coupling/decoupling mechanism was implemented where the slave remains at its current position when the master is coupled.

This mechanism is achieved by storing the slave pose when the decoupling button is pressed denoted by ${}^B LS$ (Lockframe Slave expressed in Base frame), after which the master device is decoupled from the end effector. When the master device is coupled again, the current master pose is stored as ${}^B LM$ (Lockframe Master expressed in Base frame). To then translate the current master position to the slave position, the relative current position w.r.t. ${}^B LM$ is calculated by subtracting ${}^B CM$ (Currentframe Master expressed in Base frame) by ${}^B LM$. Then to get the current slave position ${}^B CM$ is added to ${}^B LS$ to get ${}^B CS$ (Currentframe Slave expressed in Base frame). An overview of the vector algebra applied to achieve this is depicted in Figure C.2.

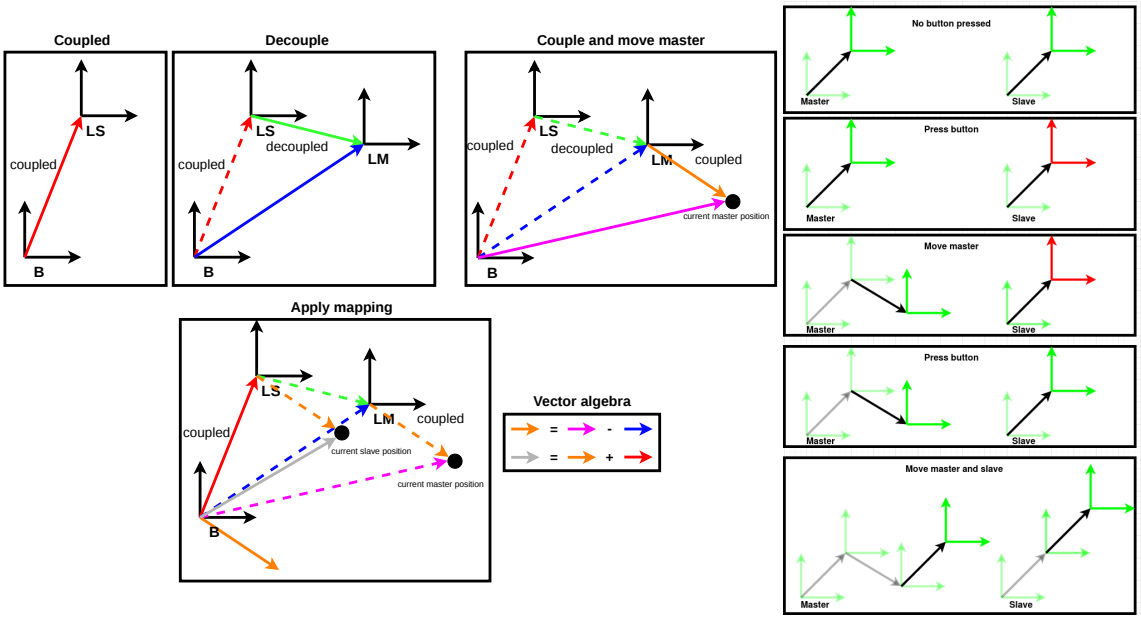


Figure C.2: On the left the vector algebra used to couple/decouple the end effector is depicted. On the right a high level description of this mechanism is shown

D

Data pre-processing

The raw demonstration data, position and orientation, are pre-processed before being input to the conditioned ProMP model. First the raw data is aligned in time using Dynamic Time Warping (DTW), after which they are resampled and converted to be relative to the object position. The setup used to evaluate the pre-processing in the dishwasher scene, where trajectories for one object position are being demonstrated. Two demonstrations are being performed going around the left and right side of the basket respectively. The third demonstration is going straight forward. Trajectory 1, 2, 3 are the blue, orange and green trajectory respectively.

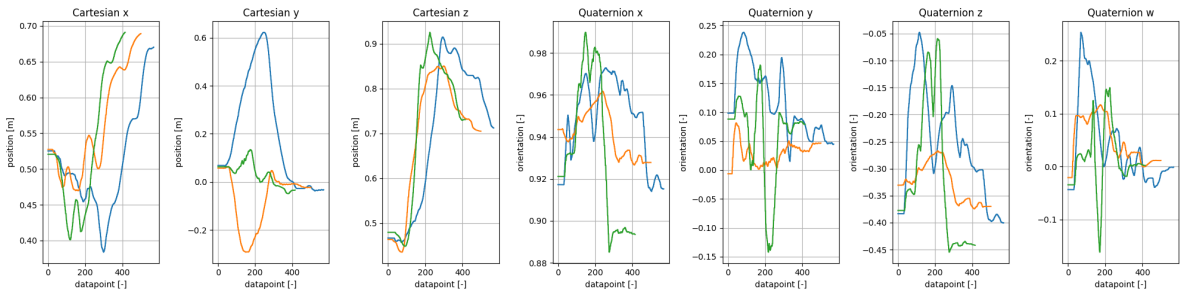


Figure D.1: Raw demonstration data. Blue/orange/green: demonstrations that go around the left side, right side and straight respectively

Dynamic Time Warping

Since the raw demonstrations are not performed at the same speed, they are first aligned in time, which is done using Dynamic Time Warping (DTW) [9]. This method calculates a distance matrix to find the best alignment between two time series, which is called the warping path. Suppose we have two time series $X = [x_1, x_2, \ldots, x_m]$ and $Y = [y_1, y_2, \ldots, y_n]$, the distance matrix is computed using Equation (D.1).

$$D(i,j) = distance(i,j) + \min\big(D(i-1,j), D(i,j-1), D(i-1,j-1)\big), \quad \text{where } distance(i,j) = |X[i] - Y[j]| \tag{D.1}$$

Suppose $X = [1,2,4,3]$ and $Y = [1,1,2,4]$, then the distance matrix is computed as shown in Table D.1.

4	4	2	**0**	**1**
2	1	**0**	2	3
1	**0**	1	4	6
1	**0**	1	4	6
	1	2	4	3

Table D.1: Distance matrix, horizontal axis is X and vertical axis is Y. The warping path is shown in bold.

The warping path is then calculated by choosing the path with the minimal distance:

$$\min D(W), \text{ where } D(W) = \sum_{k=1}^{L} distance(w_{ki}, w_{kj}) \tag{D.2}$$

Which in this example is equal to $D = 1+0+0+0+0 = 1$ as shown bolded in Table D.1. The optimal warping path has a length of 5. The following mapping is applied $(X[i], Y[j])$: (1, 1), (1, 1), (2, 2), (4, 4), (4, 3). This means that X and Y are warped onto an array with length of 5, resulting in the following arrays:

$$X = [1,1,2,4,4] \tag{D.3}$$

$$Y = [1,1,2,4,3] \tag{D.4}$$

An additional illustration of this algorithm is depicted in Figure D.2, which is adopted from [10].

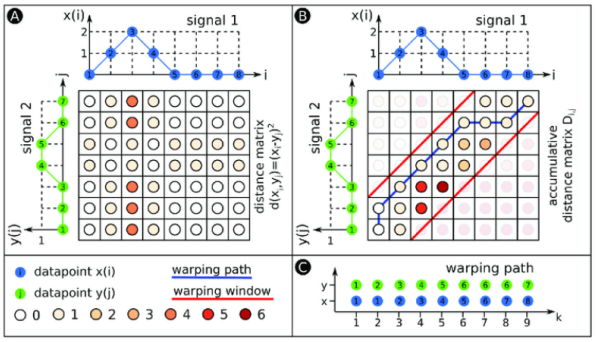


Figure D.2: Illustration of how the DTW algorithm works [10]. In A) the distance matrix is computed where each entry equals to the Euclidean distance between the datapoints from signal 1 and 2. In B) the warping path is depicted inside the distance matrix, and in C) the warping path states which datapoints of signal 1 align with which datapoints from signal 2.

To determine the optimal warping path and the corresponding distance value the FASTDTW algorithm is used [11], which is an implementation of [12].

Since only the trajectories with similar condition need to be aligned, the differences in condition between the demonstrations are calculated. If this difference is below a threshold of 0.01m, it is assumed that their condition is similar, hence they will aligned using DTW. These differences in condition are depicted in Equation (D.5), where N trajectories are being compared.

$$\begin{bmatrix} d_{11} & d_{12} & \dots & d_{1N} \\ d_{21} & d_{22} & \dots & d_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ d_{N1} & d_{N2} & \dots & d_{NN} \end{bmatrix} \tag{D.5}$$

When applying this to the example data as depicted in Figure D.1, the difference matrix becomes as shown in Equation (D.6). As expected, each of these three demonstrations have the same condition.

$$C_{difference} = \begin{bmatrix} 0 & 0.002290 & 0.009600 & 0.009600 \\ 0.002290 & 0 & 0.01096 & \\ 0.009600 & 0.01096 & 0 & \end{bmatrix} \tag{D.6}$$

Since DTW is only capable of aligning two time series, it is not possible to align more than two demonstrations per condition. When this is the case, a reference trajectory is determined which has the lowest distance [13]. The distance is determined as shown in Equation (D.7) [13].

$$distance(i) = \sum_{j=1}^{N} \text{DTW}(i,j), \forall i \in 1,2,\dots N \tag{D.7}$$

The trajectory with the second lowest distance is compared to the reference trajectory using DTW, while the other demonstrations are not used [13]. The distance of the example demonstrations in Figure D.1 are:

$$distances = [1868, 1348, 1778] \tag{D.8}$$

This means that trajectory 2 (orange in Figure D.1) is the reference trajectory and trajectory 3 (green in Figure D.1) is the most similar to the reference. Therefore trajectory 2 and 3 are being used for alignment using DTW, of which the result is shown in Figure D.3.

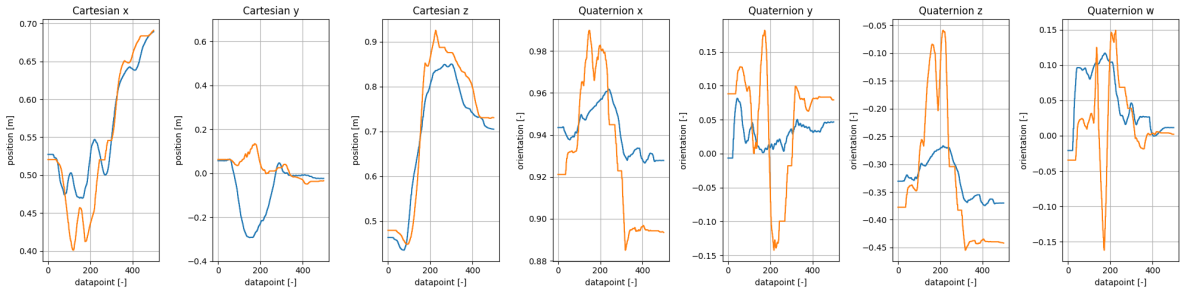


Figure D.3: Aligned demonstrations using DTW.

To better show why DTW is necessary, another dataset is used of 2 trajectory demonstrations. The raw data is shown in Figure D.4, where it can be seen that the demonstrations are similar but they have different speeds. When they are resampled to 780 datapoints, it can be seen in Figure D.5 that they are not well aligned.

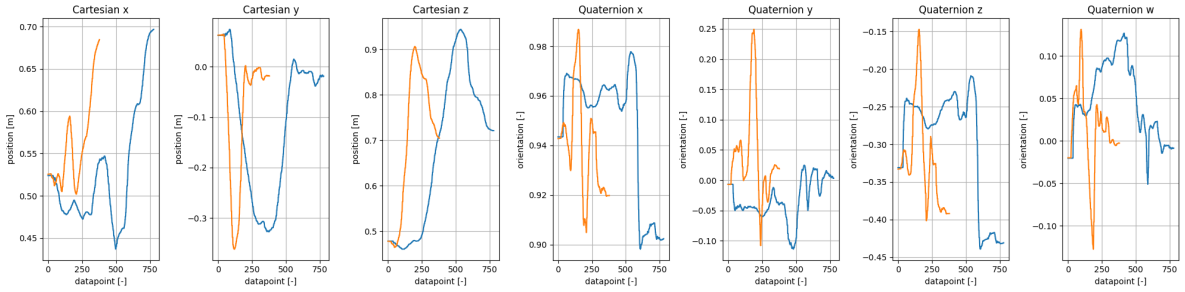


Figure D.4: Raw demonstration data to show why DTW is necessary

When instead DTW is used, of which the result is shown in Figure D.6, the demonstrations are better aligned. Nonetheless the trajectories also have 780 datapoints, which is the same as in Figure D.4.

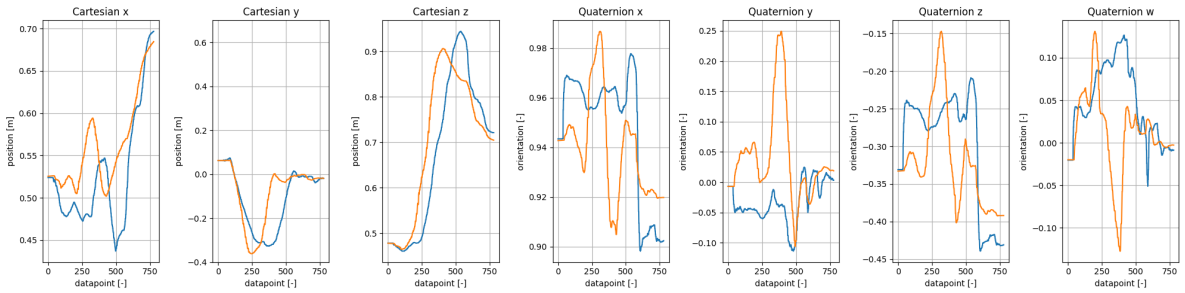


Figure D.5: Result of resampling the demonstrations from Figure D.4 to 780 datapoints

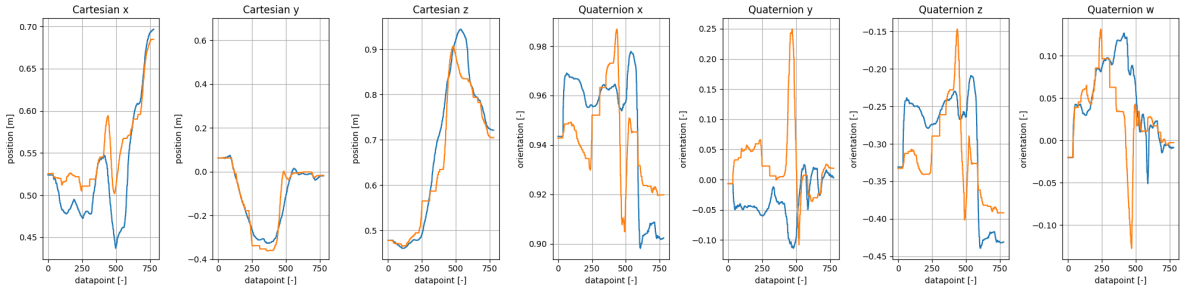


Figure D.6: Result of applying DTW to the demonstrations from Figure D.4

Resampling

After applying DTW, the trajectories are resampled to 10 datapoints, which is the amount which the ProMP model will be trained on. This amount is a trade-off between overfitting on the specific motion of the trajectories and between generalizing important motion. Too much datapoints will result in unimportant motion to be generalized, while too little datapoints result in important information to not be encoded.

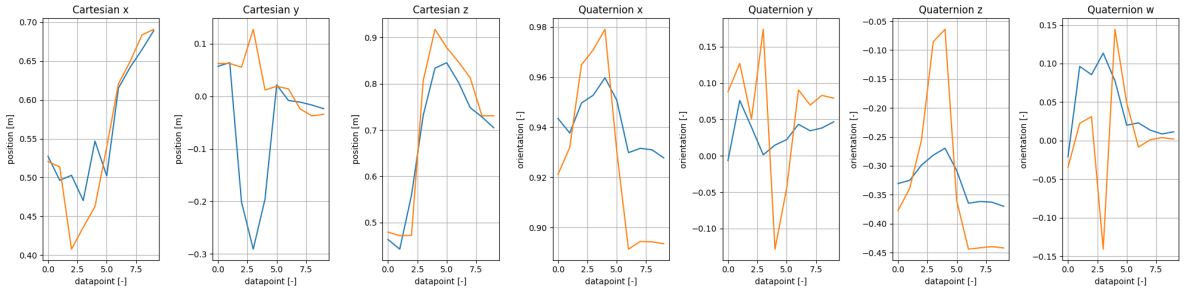


Figure D.7: Result of resampling the aligned trajectories from Figure D.3 to 10 datapoints

Trajectories relative to object and model input

The last data pre-processing step is to make the demonstrations relative to the object position and to determine the object position relative to the end effector at the first datapoint. This is necessary to be able to generalize the motion when the starting pose of the end effector is different. The first data point of the demonstration is used to determine the object position with respect to the end effector frame, which is denoted by the purple vector in Figure D.8. Furthermore the end effector motion relative to the object position is calculated by subtracting ${}^{base}ee(t)$ by ${}^{base}object(t)$, denoted by the red and purple vector in Figure D.8 respectively.

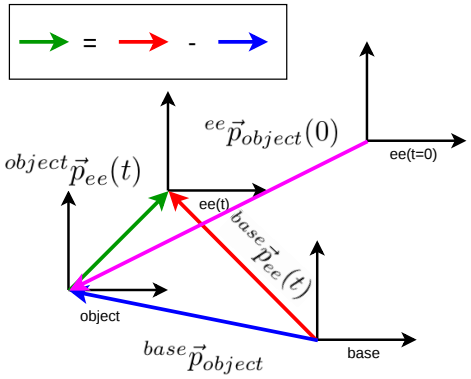


Figure D.8: Illustration of how the demonstrations are transformed to be relative to the object. Blue: object relative to base, red: end effector at time step t relative to *base* frame, green: end effector at time step t relative to object and purple: object relative to end effector frame at time step $t = 0$

In the demonstrations depicted in Figure D.3, the object position with respect to the base frame was approximately [0.024m, 0.911m, 0.558m]. The result of applying the transformation to the trajectories to make them relative to the object position is depicted in Figure D.9.

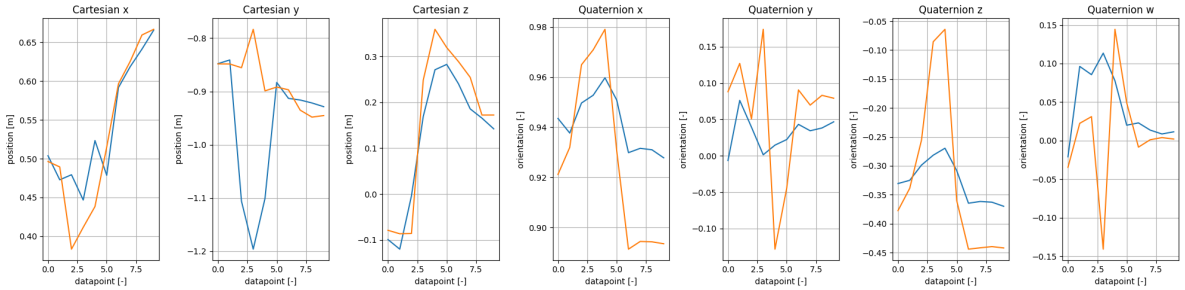


Figure D.9: Demonstrations transformed to be relative to the object position

Model input

The input or condition used is ${}^{ee}\vec{p}_{object}$ at $t = 0$, but this is multiplied by a factor of 10. This was experienced to have better prediction results. An illustration of this is shown in Figure D.10, where in the left images the demonstrations are shown and the right images show the predictions. In the images above, the input is not multiplied by 10 while the images below are. When no multiplication by 10 is done, it can be seen in the top right image that the predictions are not correct, as they should be further apart. When the input is multiplied by 10, the predictions are correct, as seen in the bottom right image.

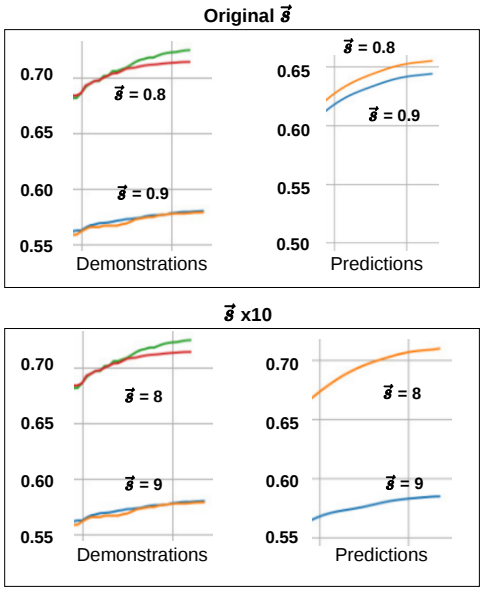


Figure D.10: Illustration of why the input is multiplied by 10 before being input to the ProMP model. The left images show the demonstrations input to the model, while the right images show the predictions using the same inputs as the demonstrations

E

Human factors experiment

This section shows the details of the performed human factors experiment.

Pipeline

The complete experimental pipeline is depicted in Figure E.1.

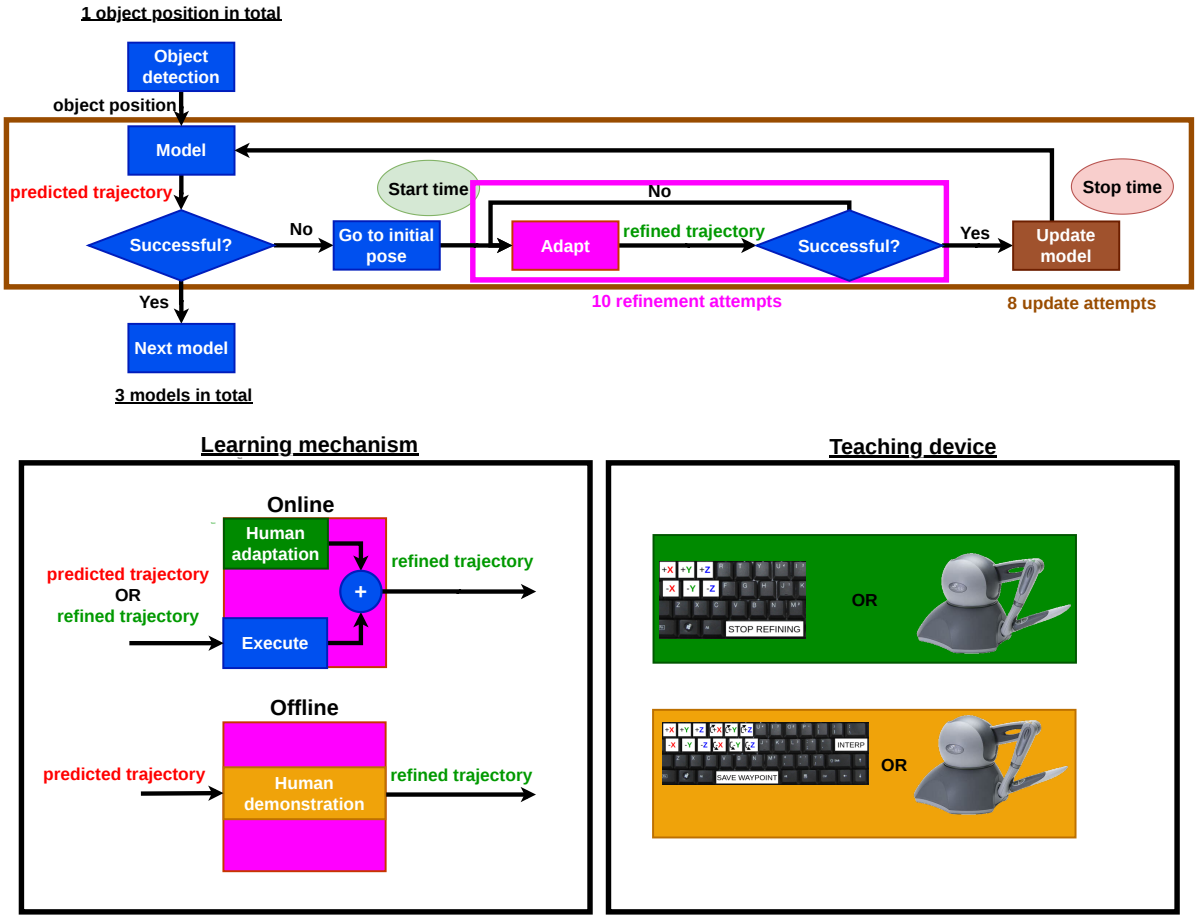


Figure E.1: Complete experimental pipeline

Success criteria

The object is reached if the center of the object coordinate frame is within the ellipsoid drawn inside the gripper (see Figure E.3 for this ellipsoid). This means that the vector between the center of the ellipsoid and the object coordinate frame, $\vec{p}_{object}$, should be smaller than or on the boundary of the ellipsoid. To calculate this, the ellipsoid equation is used, in which the x, y and z coordinates of $\vec{p}_{object}$ are substituted, and if this is smaller or equal to 1 the object is defined as reached (see Figure E.2).

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} \leqslant 1$$

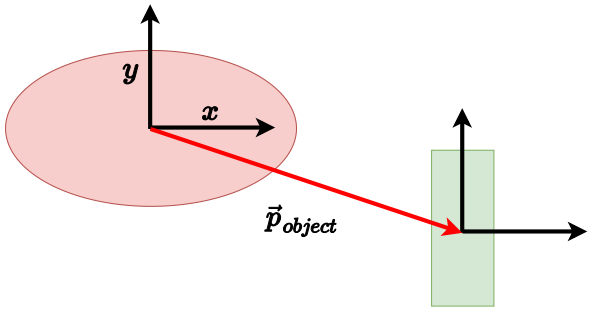


Figure E.2: Definition used for when the object is reached. The x, y and z coordinates of $\vec{p}_{object}$ are substituted in the ellipsoid equation, where a, b and c are the radii of the ellipsoid in these dimensions. If the substituted equation is smaller than or equal to 1, $\vec{p}_{object}$ is within or on the ellipsoid boundary and the object is defined as reached

Furthermore the object is kicked over if the difference between the initial and the current x, y and z coordinates of the object are larger than a certain threshold:

$$(|x_0 - x_1| \ or \ |y_0 - y_1| \ or \ |z_0 - z_1|) > threshold$$

The collision detection algorithm of the simulator is used to determine the collision between the robot and the environment, where self and floor collision have not been taken into account. Visual feedback is provided when these conditions are satisfied, of which an illustration is shown in Figure E.3.

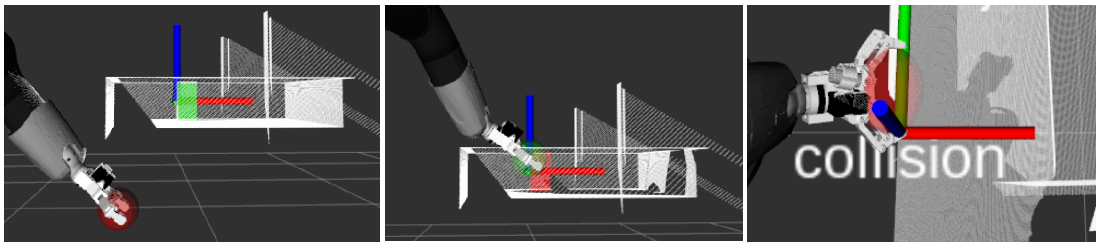


Figure E.3: Red/green ellipsoid within gripper: Object (not) reached, red/green box around object: object (not) kicked over, "collision" coordinate frame shows where collision has occurs

Normally distributed metric evaluation

Figure E.4: Distributions of refinement time and workload, depicted in the left and right graphs respectively. They are separated on teaching (stylus or keyboard) and learning mechanism (online or offline).

Counterbalancing

	Method 1	**Method 2**	**Method 3**	**Method 4**
Group 1	OnStyl	OnKey	OffKey	OffStyl
Group 2	OnKey	OffStyl	OnStyl	OffKey
Group 3	OffStyl	OffKey	OnKey	OnStyl
Group 4	OffKey	OnStyl	OffStyl	OnKey

Table E.1: Balanced Latin Square experiment design

Participants background information

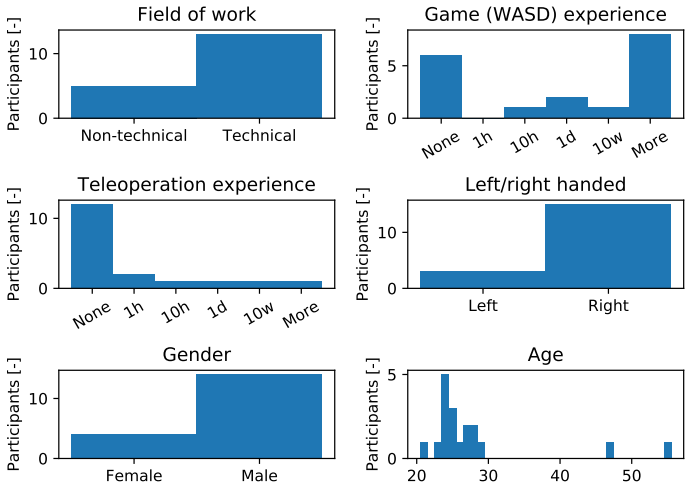


Figure E.5: Background information of the participants

Training

The operator keeps creating refinements, until the moving average of the successes over the past 4 refinements is larger than 50%. This means that if 2/4 successful refinements are created, the real experiment is started. If the operator did not succeed within 30min, the experiment for this method is not started as this method is assumed to be not workable for this participant. The distribution of the training time over the participants that passed the training is depicted in Figure E.6, because the training time of the participants that failed was not stored.

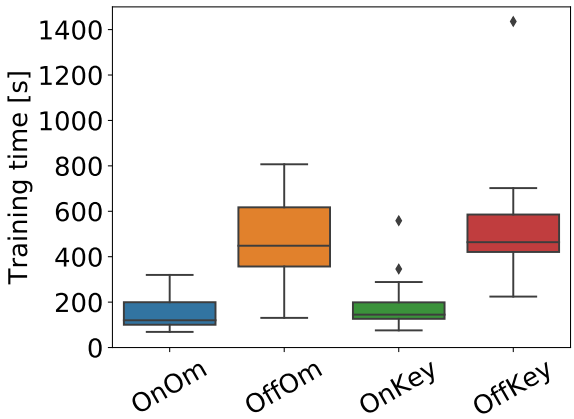


Figure E.6: Training time of participants that succeeded the training, shown per method.

Pilot

The evaluated metrics are the refinement time and workload. To be able to calculate the refinement time, every model for each method has to be successfully adapted. If this is not the case, the model is not refined and the refinement time metric cannot be calculated. This means that the experiment has to be easy enough, such that people are able to accomplish this. A pilot experiment was done on 5 participants, of which the amount of successfully adapted models have been evaluated. In Figure E.7 it can be seen that for none of the methods, all participants were able to successfully adapt all 3 models. Thus it was concluded that the experiment in this form was too difficult.

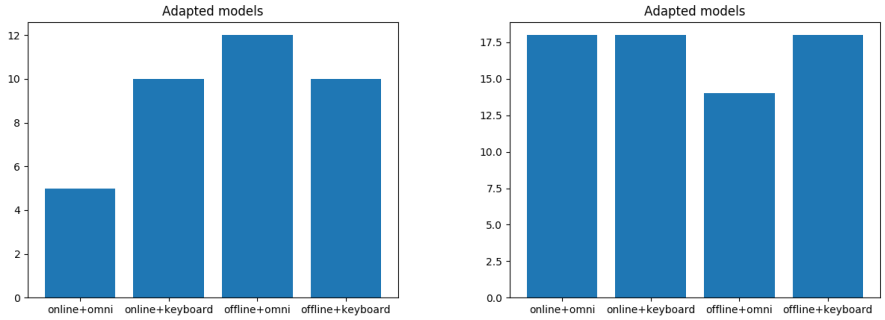


Figure E.7: Left: Amount of successfully adapted models per method in the pilot (max 15 models per method). Right: Amount of successfully adapted models per method in the final (easier) experiment (max 18 models per method)

Changes have been made to the dishwasher position, which was placed further away from the robot. In this way the prediction required less adaptation to avoid the pulled out basket. Moreover in the online methods, it was made easier to create a large refinement and the allowed refinement and update attempts were increased (4 to 8 and 5 to 10 respectively). The amount of adapted models in the final experiment are also shown in Figure E.7. It can be seen that for Offline + Omni, still some models are not successfully adapted and can thus not be used in the statistics for the refinement time.

Graphical User Interface (GUI)

Two GUIs have been build, one that is shown to the operator during the experiment, and one to fill in the background information before the experiment. The interface has 7 important regions to consider for the participant:

1. The camera view of the robot. Top is the head camera and bottom is the wrist camera.
2. Point cloud of the dishwasher, with the robot model, the predicted and refined trajectory and the failure detection visualization.

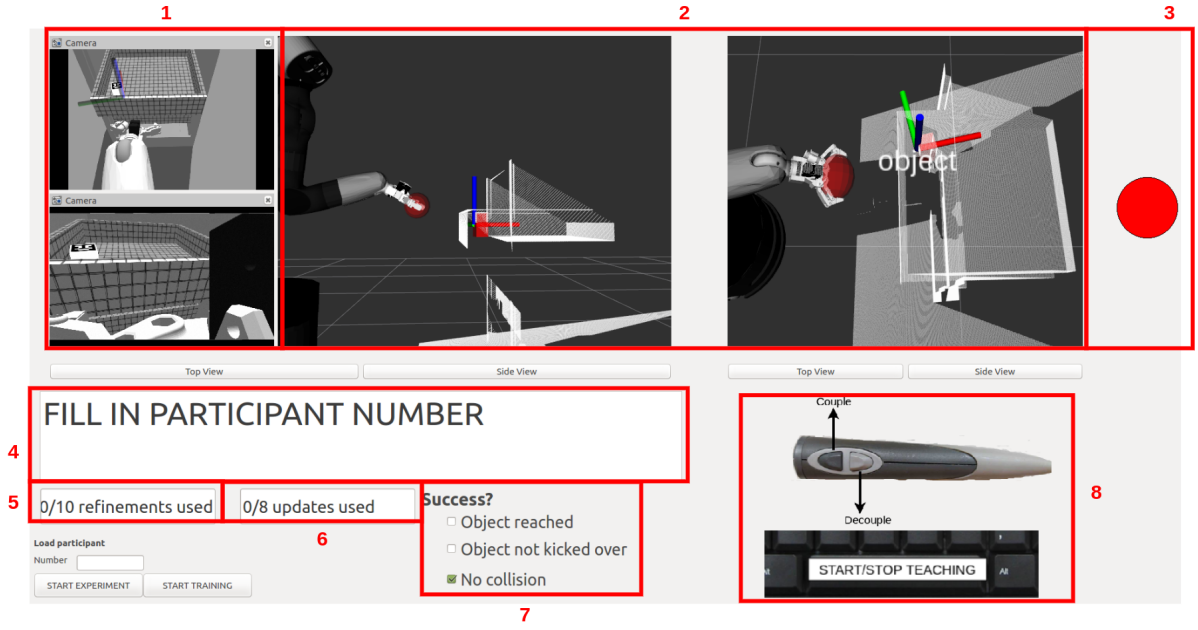


Figure E.8: Graphical User Interface shown to the operator during the experiment

3. Traffic light: Red = You should not do anything, Green = You should do something
4. Text giving details about the state of the experiment
5. How much refinements are used (max 10 per update)
6. How much updates are used (max 8 per object position)
7. Successfulness of trajectory displayed in text
8. Instructions on how to generate a trajectory using the current method

Another GUI has been build to evaluate the background information of the participants before the experiment, of which an illustration is depicted in Figure E.9.

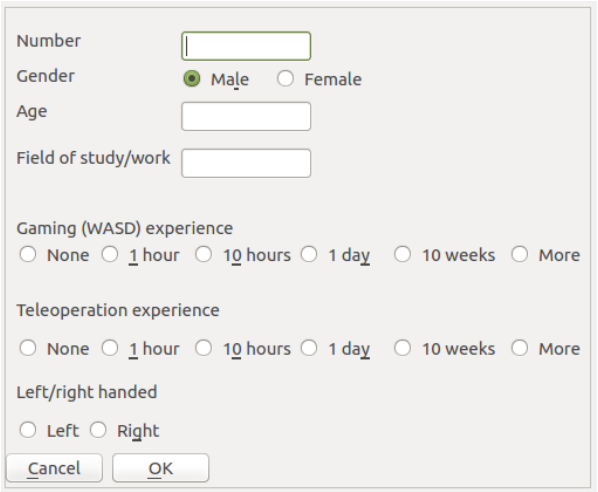


Figure E.9: GUI build to fill in background information of the participants before the experiment

Dishwasher model

The dishwasher model was downloaded from [14] and the baskets were adjusted to prevent the physics simulator becoming unstable when the arm touches the basket. This is done by making a solid basket in SolidWorks, and applying a texture of a fence over the basket in Blender. Furthermore the door was removed, such that the robot can more easily reach the object within the basket. And transparency was applied, such that it visually resembled a real basket (this did not work in the older version of Gazebo, which has to be used to control the robot). After that the model was converted to a Universal Robot Description File (URDF) such that it can be used within the simulation environment of the robot.

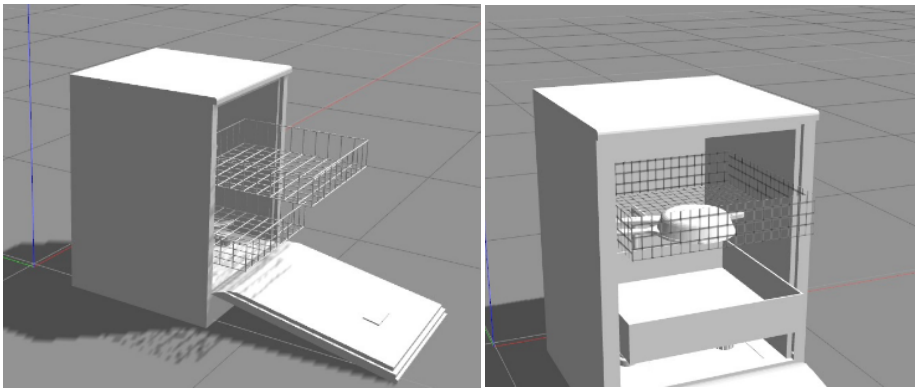


Figure E.10: Left: Original dishwasher model with instable basket/gripper collisions, Right: Solid basket with applied texture, which has more stable collisions

Additionally a ROS controller [15] was build such that the upper basket can be controlled, but this was not used in the end since having multiple robots in the same environment causes the simulator to fail. Instead a second simulator world was build with a dishwasher that has different basket position.

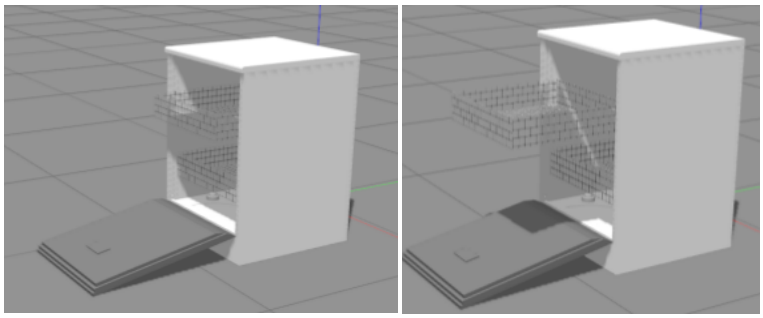


Figure E.11: ROS controller of upper basket, left: basket closed, right: basket open

Trained initial model

An initial model was trained on a dishwasher with upper basket x position 1.7 (w.r.t the base link of the dishwasher). It was trained on two object positions, which vary only in the y-direction w.r.t the base frame: ${}^{base}p_{object\,1} = [0.9177,\ 0.2592,\ 0.5621]$ and ${}^{base}p_{object\,2} = [0.0247,\ 0.9271,\ 0.5546]$. This was experienced to be a good trade-off between being robust against small changes in the object position, and a model that can easily be adapted (small amount of initial trajectories make it easier to change the mean of all the demonstrations). The raw demonstrations for these two object positions are depicted in Figure E.12, where it can be seen that the main difference is present in the y-coordinate of the trajectories.

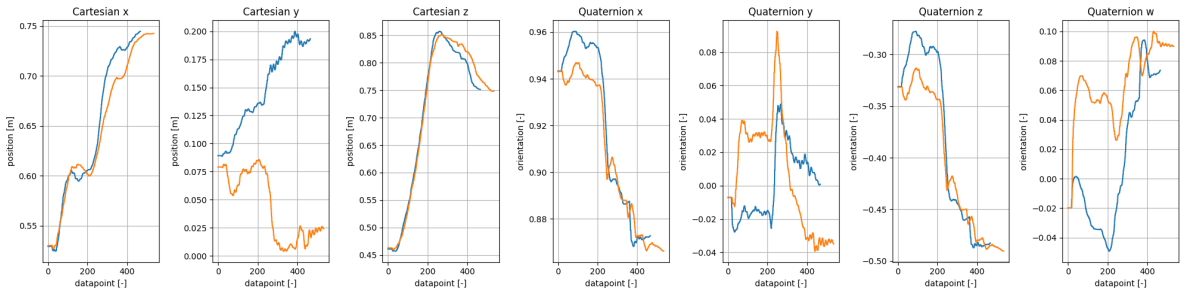


Figure E.12: Raw data generated using the Phantom Omni

These raw demonstrations are resampled, such that they contain only 10 datapoints. This is empirically determined to be a good trade-off between overfitting and generalization capabilities. Large amount of data points will result in overfitting the demonstrations, while low amount of data will result in bad generalization of environmental constraints. An illustration of these resampled trajectories is depicted in Figure E.13.

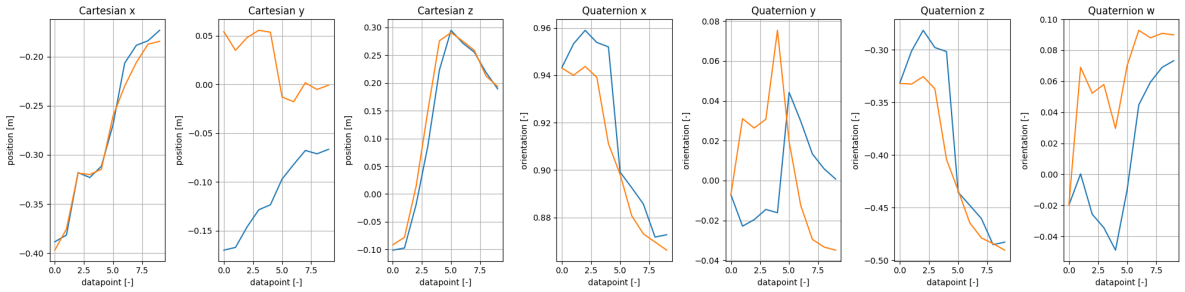


Figure E.13: Resampled trajectories to contain only 10 datapoints

Since no multiple trajectories per condition have been demonstrated, DTW is not necessary and this step is skipped. The last step is to transform the coordinates of the trajectories to be relative to object position. The result of this step is shown in Figure E.14. The trajectories in Figure E.14 are fed into the model

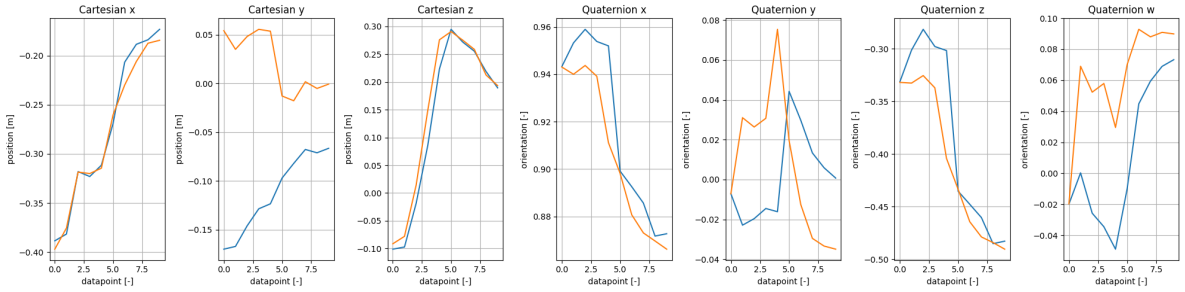
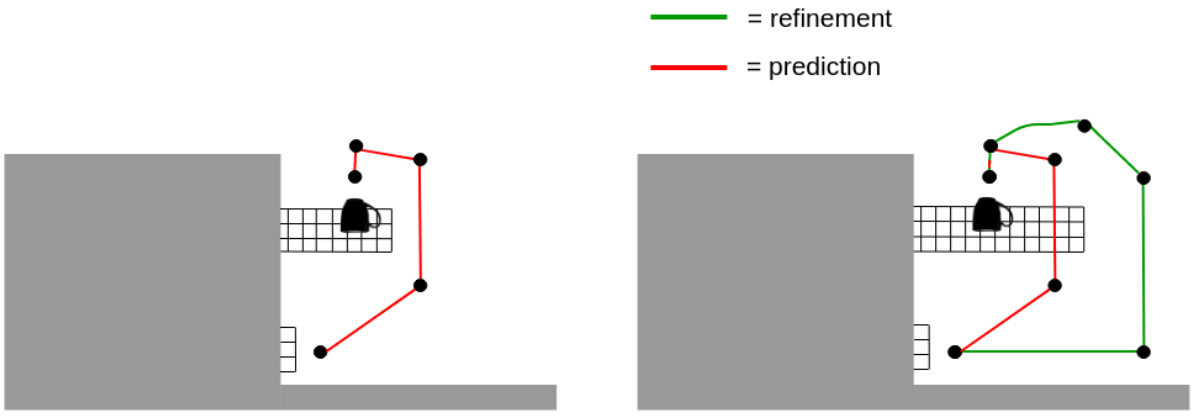


Figure E.14: Trajectories tranformed to be with respect to the object instead of the base frame

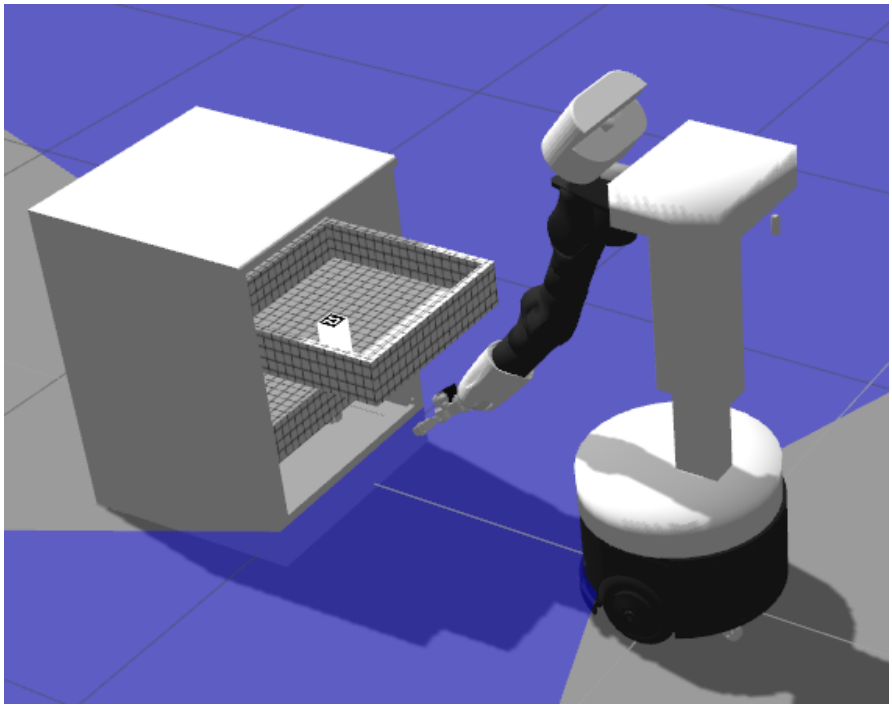
with their corresponding condition. The condition is the object w.r.t the end effector at $t = 0$, which is $condition\ 1 = [0.3886, 0.1700, 0.1010]$ and $condition\ 2 = [0.3971, -0.0543, 0.0913]$. An evaluation of the initial model on different object positions across the y-direction is depicted in Figure E.15. Here it can be seen that the constraint of the basket is successfully generalized.

Participant Instructions

Today, you are going to teach a care robot how to reach an object inside a dishwasher. This robot is trained to reach these objects for a certain basket position (left picture). In the experiment the basket is pulled outwards (right picture) and the trajectory the model generates is expected to fail, thus we need to adapt the model.



This adaptation will be done in simulation, of which the complete setup looks like this:



Experiment pipeline

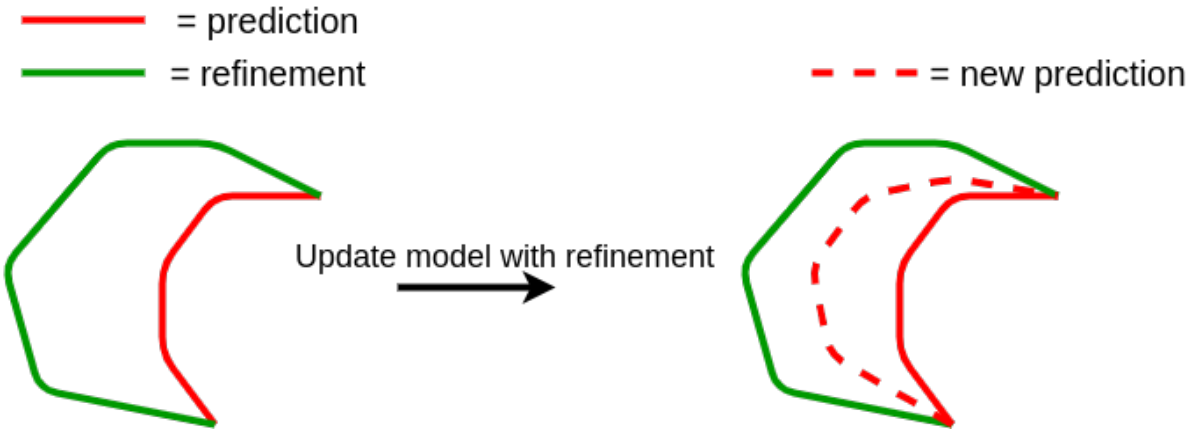
The experiment consists of 3 phases:

1) Model generates prediction

2) You refine prediction

3) Model is updated with refined prediction

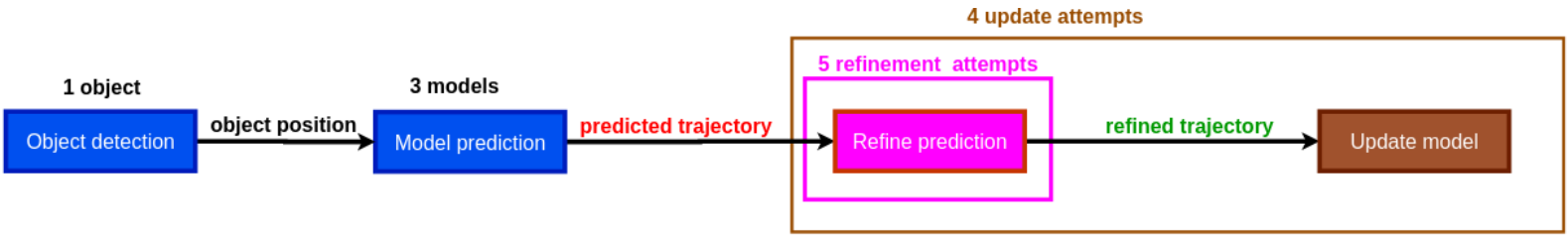
Phase 2 is where you as operator come into play. As soon as the model is not able to generate a successful prediction, you as operator are asked to refine this prediction. This is done using **4 different methods**. After you have created a successful refinement, the model is updated with this. *Hint: The model uses averaging to generate trajectories, so to adapt the model more quickly, you can update the model with an exaggerated motion (see image below).*



A rough pipeline is shown below, which will be repeated for each of the 4 methods.

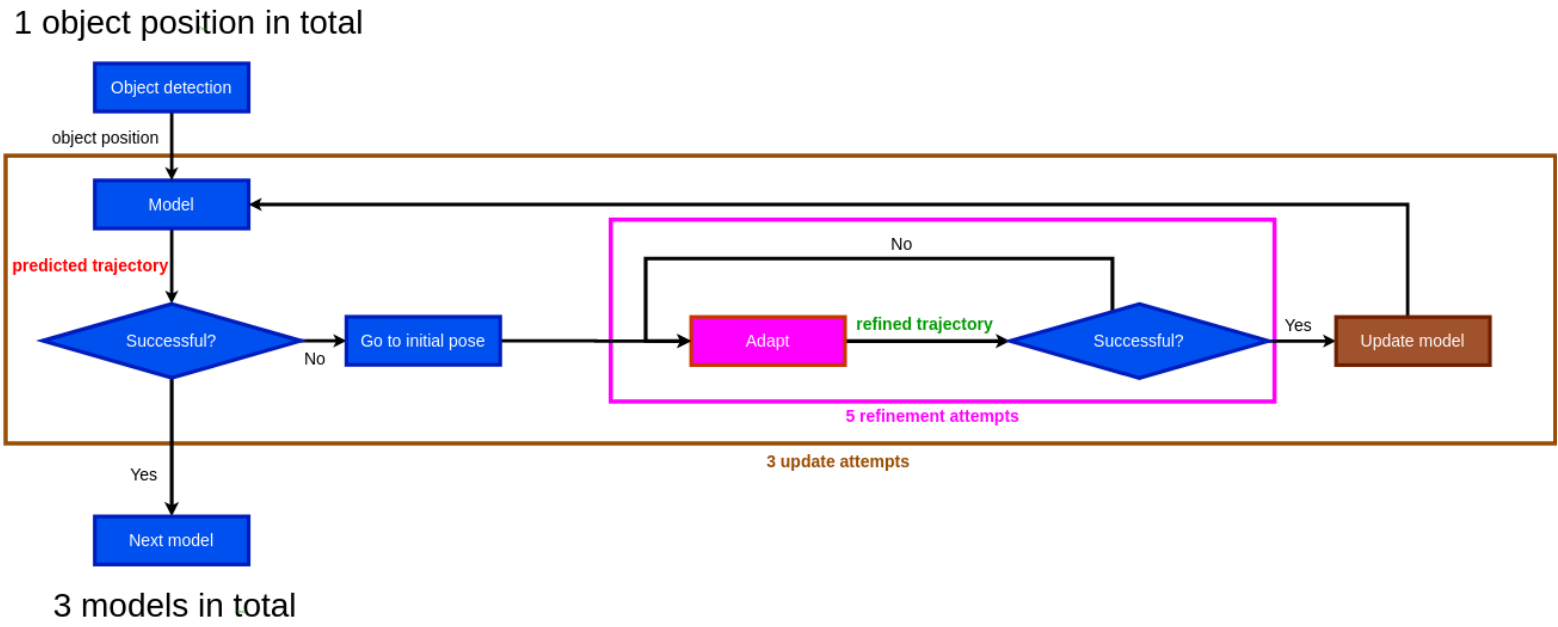
If you did not succeed within 5 refinement attempts → Go to next model.

If you did not succeed within 4 updates → Go to next model.



31/08/2020

The complete pipeline is shown below:

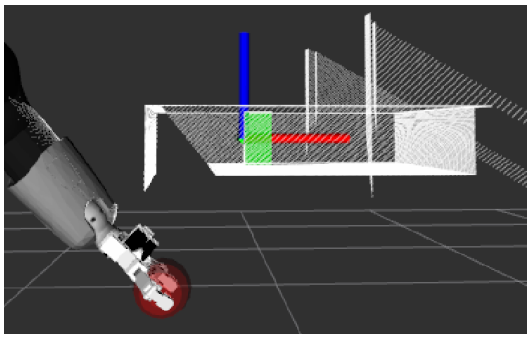


The experiment will approximately take ~2.5 hours, where the 4 methods are being evaluated on their performance in adapting models.

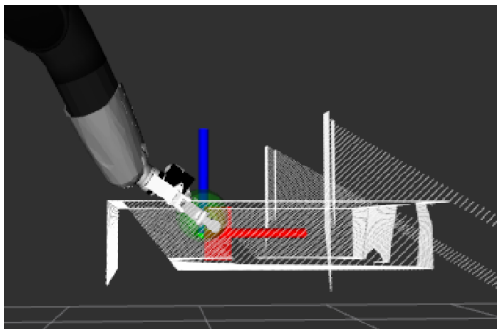
- Training phase: Score 50% (2/4 successful refinement), ~10min per method
- Experiment: 4 methods, ~30min per method

When is a trajectory successful?

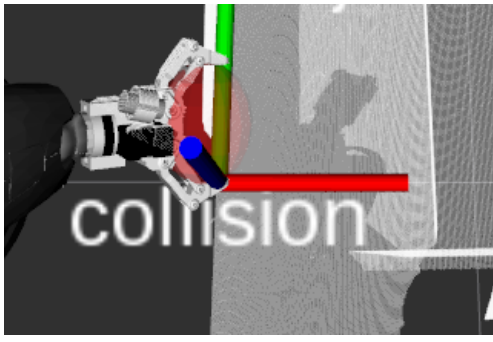
1) Object reached (center of object frame should be inside red ellipsoid), 2) object not kicked over, 3) no collision



Object not reached (red ellipsoid)
Object not kicked over (green box)



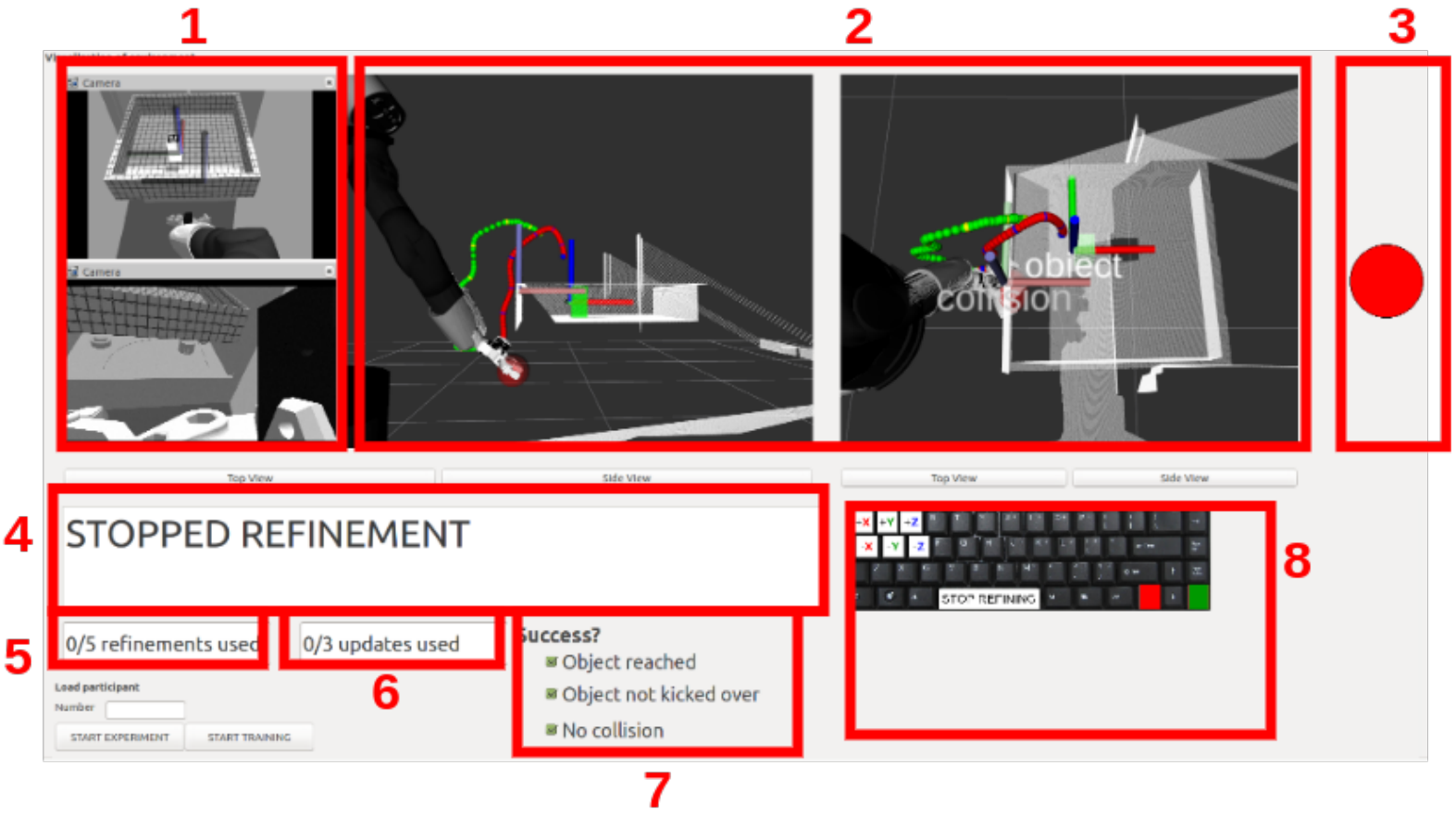
Object reached (green ellipsoid)
Object kicked over (red box)



Collision

User interface

The user interface consists of a couple of important regions:

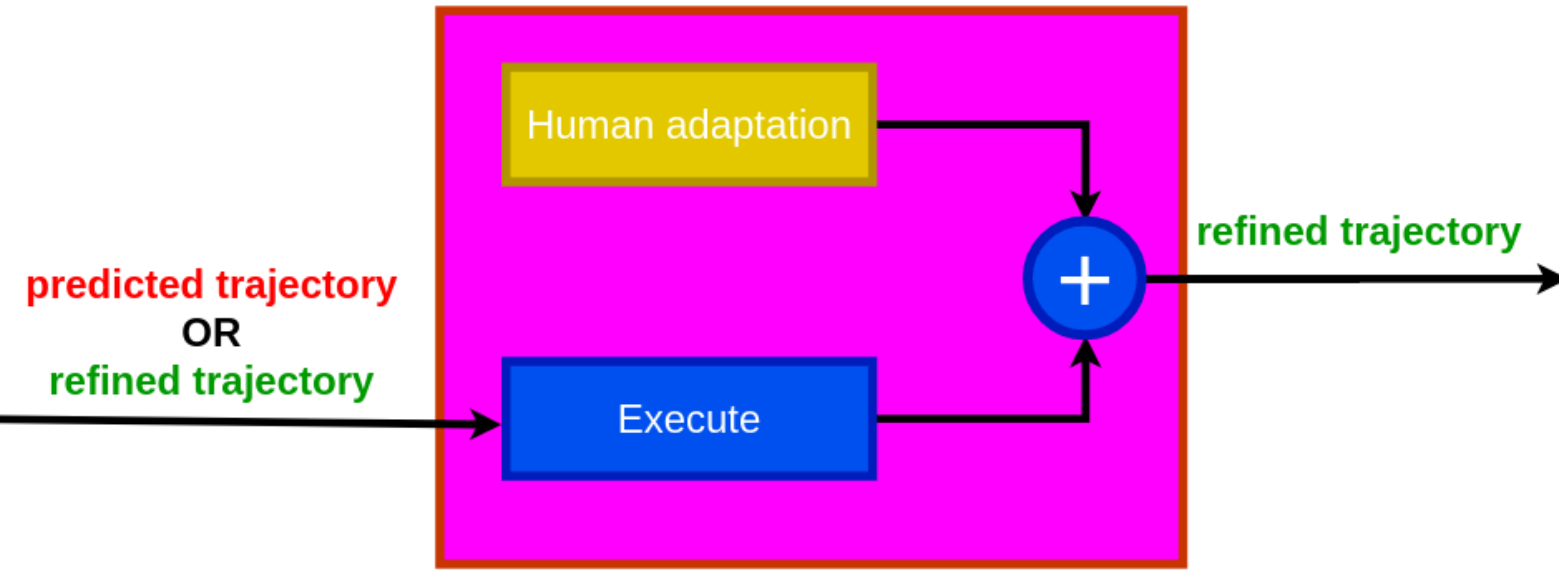


The user interface has 7 important regions to consider.

1. The camera view of the robot. Top is the head camera and bottom is the wrist camera.
2. Point cloud of the dishwasher, with the robot model, the predicted and refined trajectory and the failure detection visualization (details follow).
3. Traffic light: Red = You should not do anything, Green = You should do something
4. Text giving details about the state of the experiment
5. How much refinements are used (max 5 per update)
6. How much updates are used (max 4 per object position)
7. Successfulness of trajectory displayed in text
8. Instructions on how to generate a trajectory using the current method

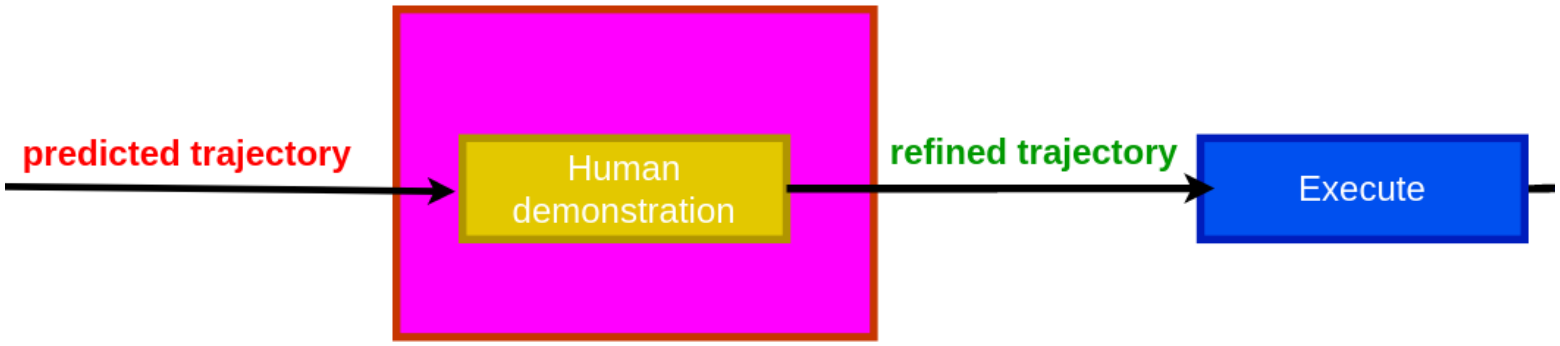
Methods

Making a **refined trajectory** (see flowchart) is done using 4 different methods. 2 of them are online and 2 are offline methods. In online methods, the predicted trajectory is adapted during execution time:



When the model prediction was unsuccessful (see flowchart), you can chose between refining the previous refinement or the initial prediction. This choice is based upon which trajectory you think will have the highest potential to be successful.

In offline methods you teach a trajectory from scratch:



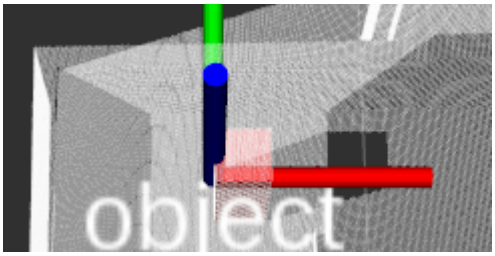
Human adaptation/demonstration is done using either a keyboard or the Geomagic Touch.

Keyboard + Offline teaching:

The keyboard is used to specify and interpolate waypoints. You can adapt both the position and orientation of the gripper in X, Y and Z direction. This is done using the keys shown in the left image below, which letters will be placed on the real keyboard. The colors represent the axis of rotation/translation as indicated in the right image.

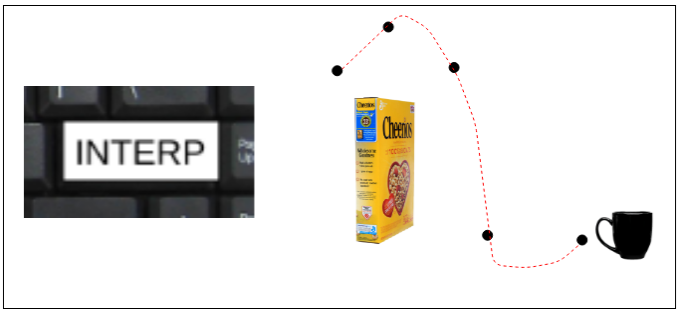
If you want to store a waypoint, you press the spacebar. *Note: At least 3 waypoints need to be specified for the interpolation to work properly.* After specifying all the waypoints as you like, you press the enter button to interpolate the waypoints.

This interpolation is then the **refined trajectory** and checked for successfulness and if this is true, it is used to update the model.



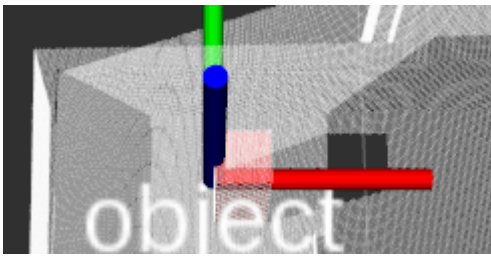
Store via points

Interpolate



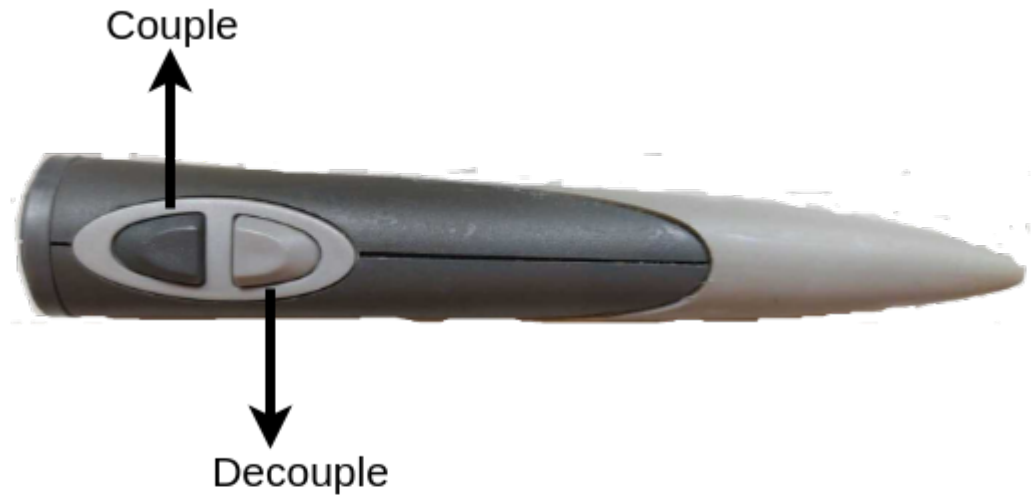
Keyboard + Online teaching:

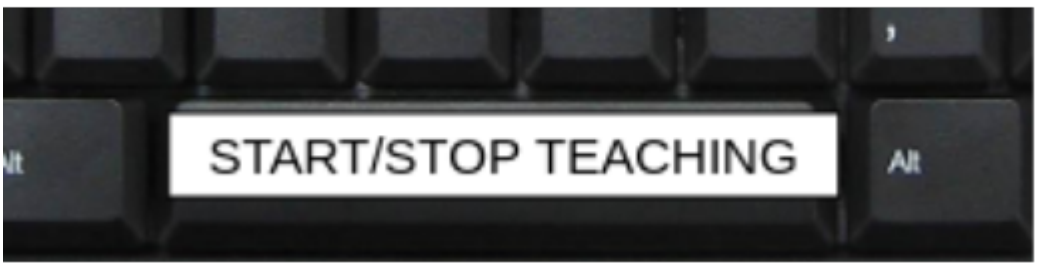
This time you adapt the position of predicted motion during execution time using the +-X, +-Y, +-Z keys on the keyboard. To stop refining, you need to press the space bar.



Omni + Offline teaching

A completely new trajectory is taught using the Geomagic Touch. Both position and orientation can be demonstrated by moving/rotating the stylus. To start/stop the teaching, the space bar should be pressed. To couple/decouple the device with the end effector, the grey/white button should be pressed.





When the model prediction is unsuccessful (see flowchart) you start teaching by pressing the space bar.

Omni + Online teaching

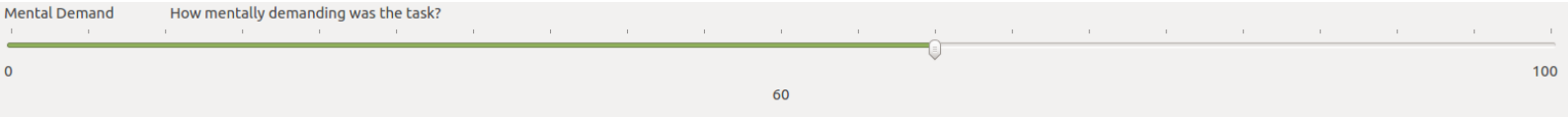
During execution time you adapt the position of the motion using the Geomagic Touch. Force feedback is provided that gives you a feel for how much you are deviating from the predicted trajectory. To stop the teaching, the space bar should be pressed.

NASA-TLX

After the complete experiment you are asked to answer questions about each of the 4 methods you used. This will give an indication of the perceived workload of the method.

Instructions: Rating scales

You are requested to rate each method based on six different criteria (shown below). This is done inside a GUI:



RATING SCALE DEFINITIONS

Title	Endpoints	Descriptions
MENTAL DEMAND	*Low/High*	How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc.)? Was the task easy or demanding, simple or complex, exacting or forgiving?
PHYSICAL DEMAND	*Low/High*	How much physical activity was required (e.g., pushing, pulling, turning, controlling, activating, etc.)? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?
TEMPORAL DEMAND	*Low/High*	How much time pressure did you feel due to the rate or pace at which the tasks or task elements occurred? Was the pace slow and leisurely or rapid and frantic?
PERFORMANCE	*good/poor*	How successful do you think you were in accomplishing the goals of the task set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?
EFFORT	*Low/High*	How hard did you have to work (mentally and physically) to accomplish your level of performance?
FRUSTRATION LEVEL	*Low/High*	How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task?

Instructions: Sources of workload evaluation

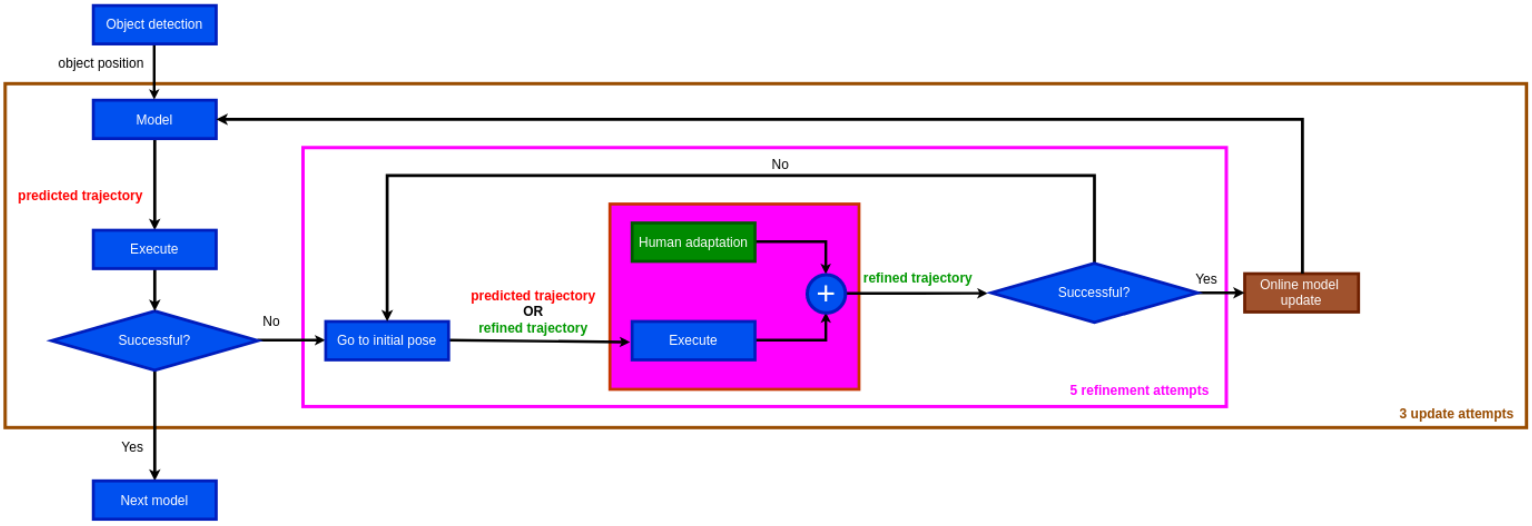
Furthermore you will be presented with a series of pairs of rating scale titles (for example Effort vs. Performance) and asked to choose which of the two was more important to your experience of workload in the task that you just performed.

○ Effort ◉ Performance

Additional material

Complete online teaching pipeline

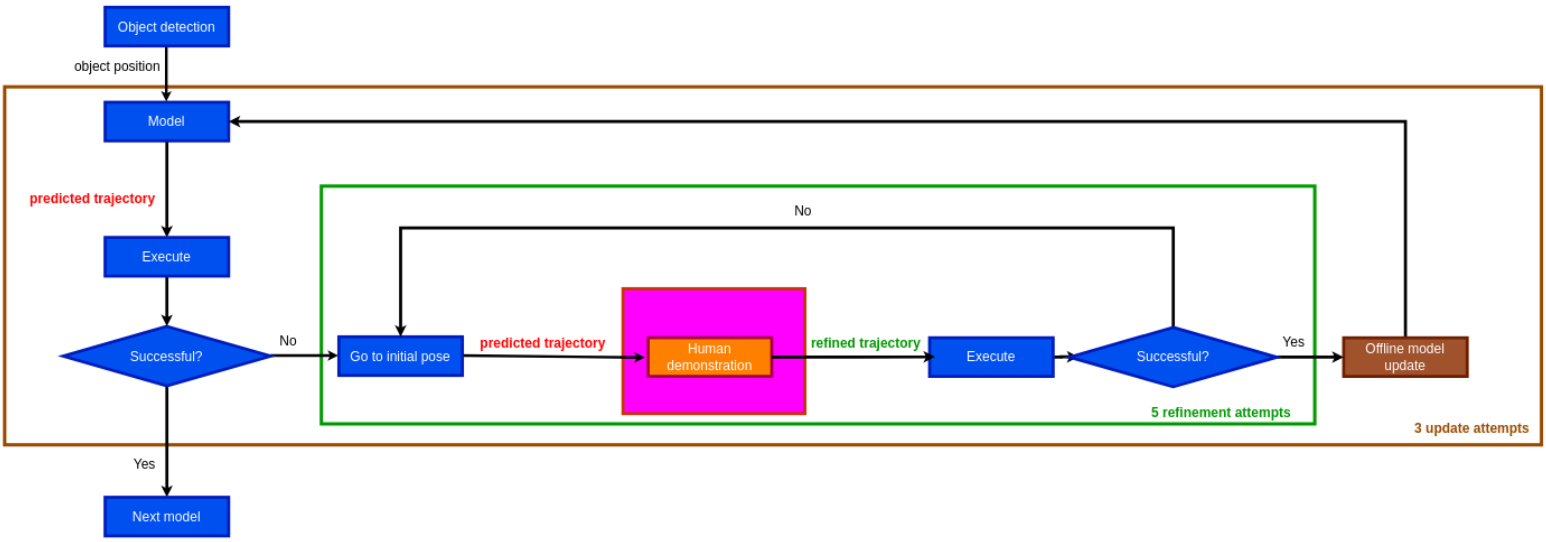
1 object position in total



3 models in total

Complete offline teaching pipeline

1 object position in total



3 models in total

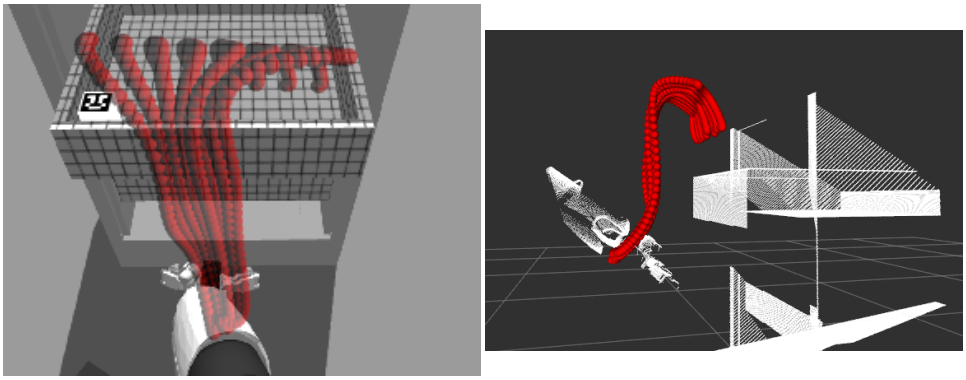


Figure E.15: Evaluation of the initial model on different object positions varied across the y-direction

NASA-TLX

The workload was determined using the NASA task load index (NASA-TLX). To conveniently determine this during the experiment, a graphical user interface was implemented depicted in Figure E.16. This user interface imitates the original NASA-TLX rating sheet and sources of workload comparison cards, which are shown in Figure E.16. In the left side of the GUI (see Figure E.16), the weighted rating sheet is implemented, while on the right sheet the sources of workload comparisons are depicted. Before calculating the workload, a check is done on how many times each source of workload occurs, which have to be equal to 15 [16]. Then to calculate the workload, the ratings per source of workload from the rating sheet are multiplied with how many times this source was chosen in the comparison cards (weight). This adjusted rating (weight * rating) is summed over all the sources of workload and divided by 15 to get the total workload (see [16] for more details).

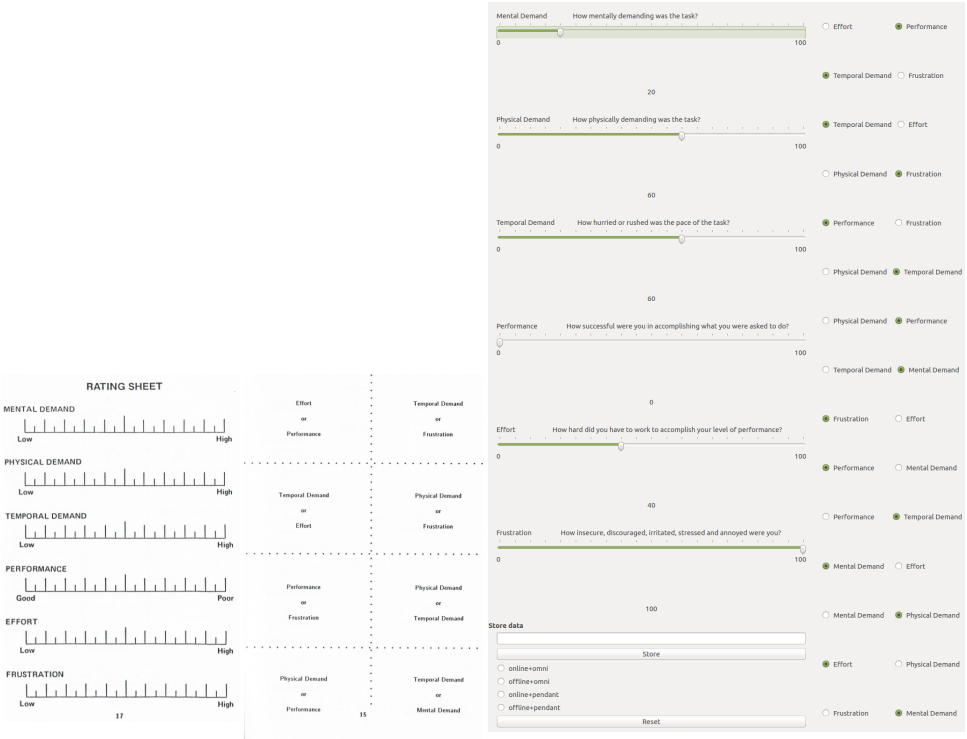
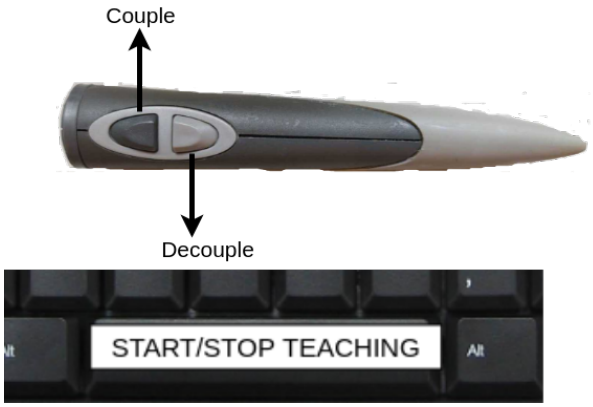


Figure E.16: Left: Original NASA-TLX [16], right: NASA-TLX graphical user interface the participants had to fill in after each method

Methods implementation

OffStyl

The light grey and dark grey button of the Phantom Omni are respectively used to couple and decouple the end effector. Furthermore the space bar is used to start and stop the recording of the trajectory. Both the position and orientation of the end effector are recorded, in addition to the total execution time T. After creating a refined trajectory, it is resampled to contain only 10 datapoints. Then the refined trajectory is executed, which means it is interpolated to contain $n = 75$ datapoints. This number is empirically determined to generate trajectories without jumps. The time step per datapoint is then calculated by $dt = \frac{T}{n}$.



OffKey

The space bar is used to specify waypoints (EE position and orientation), after which the enter button can be pressed to interpolate between these waypoints. To interpolate the position, quadratic spline interpolation is used, whereas the orientation is interpolated using spherical linear interpolation (SLERP). To achieve these interpolations SCIPY is used [17].

Furthermore a "teach loop" records the input from the operator and translates this to end effector (EE) motion. When the operator presses a translation or rotation button, the current EE position and orientation is stored and a tiny amount is added. For the translation, each button press adds 0.013m in the corresponding direction. For the rotation part, the operator can control the rotation around the x, y and z axis of the end effector. This means that by pressing a rotation button, the a new orientation will be stored, which adds 0.1rad around the corresponding axis to the current EE orientation. To convert this to quaternions, which the Whole Body Controller (WBC) [8] expects, the QUATERNIONROTATION(AXIS, ANGLE) is written:

Listing E.1: Python function used to convert an axis and angle to quaternions

```
1  def quaternionRotation(axis, angle):
2          w = np.cos(angle/2)
3          if axis == 'x':
4                  x = np.sin(angle/2)
5          elif axis == 'y':
6                  y = np.sin(angle/2)
7          elif axis == 'z':
8                  z = np.sin(angle/2)
9
10         return Quaternion(w, x, y, z).normalised
```

The new translation and rotation (in quaternions) are being send to the WBC using the ROS interface.

Quadratic spline interpolation

Quadratic spline interpolation works by fitting a spline between two consecutive data points, which are given by the following equations.

$$\begin{aligned} f(x) &= a_1x^2 + b_1x + c_1, \quad && x_0 \leqslant x \leqslant x_1 \\ &= a_2x^2 + b_2x + c_2, \quad && x_1 \leqslant x \leqslant x_2 \\ &\vdots \\ &= a_nx^2 + b_nx + c_n, \quad && x_{n-1} \leqslant x \leqslant x_n \end{aligned} \tag{E.1}$$

Filling in x_{n-1} and x_n in each of these equations gives $2n$ amount of equations:

$$\begin{aligned} a_1x_0^2 + b_1x_0 + c_1 &= f(x_0) \\ a_1x_1^2 + b_1x_1 + c_1 &= f(x_1) \\ &\vdots \\ a_1x_1^2 + b_1x_1 + c_1 &= f(x_1) \\ a_1x_2^2 + b_1x_2 + c_1 &= f(x_2) \\ &\vdots \\ a_nx_{n-1}^2 + b_nx_{n-1} + c_n &= f(x_{n-1}) \\ a_nx_n^2 + b_nx_n + c_n &= f(x_n) \end{aligned} \tag{E.2}$$

Furthermore the derivates of two quadratic splines are equal at $x = x_n$, therefore the following equations can be derived:

$$2a_1x_1 + b_1 - 2a_2x_1 - b_2 = 0 \tag{E.3}$$
$$2a_2x_2 + b_2 - 2a_3x_2 - b_3 = 0 \tag{E.4}$$
$$\vdots \tag{E.5}$$
$$2a_{n-1}x_{n-1} + b_{n-1} - 2a_nx_{n-1} - b_n = 0 \tag{E.6}$$

This are $n-1$ amount of equations, which totals to $2n+(n-1) = 3n-1$ amount of equations and $3n$ unknowns. The last equation to make the system of equations solvable is the assumption that the first spline is linear, which means $a_1 = 0$. An illustration of the splines and their derivatives are depicted in Figure E.17.

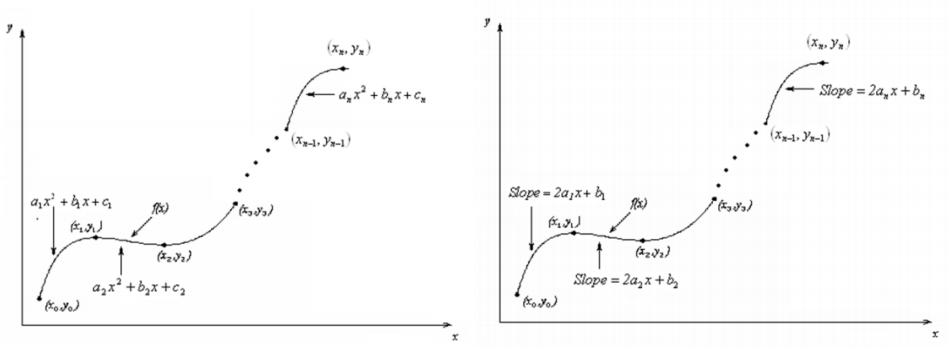


Figure E.17: Quadratic spline interpolation [18], left: Spline equations, right: Derivatives of these splines

Additional results

In Figure E.18 the amount of successfully adapted models per method are depicted, before and after excluding the 6 participants that failed to adapt at least 1 model for a method. After excluding them, we have the same amount of adapted models per method.

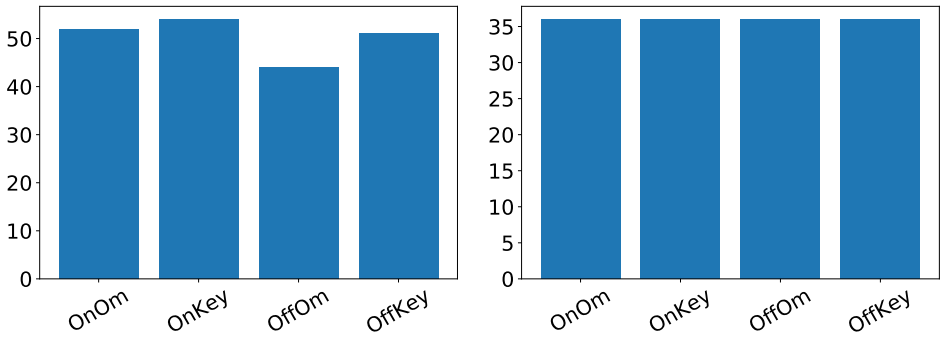


Figure E.18: The left and right images show the amount of adapted models before and after excluding the participants. We see that some participants failed at certain methods, which have to be completely excluded from the refinement time hypothesis test.

The result of teleoperation experience, gender and handedness on the refinement time and workload are depicted in Figure E.19. Table E.2 shows game/teleoperation experience and field of work and the corresponding refinement time and workload values.

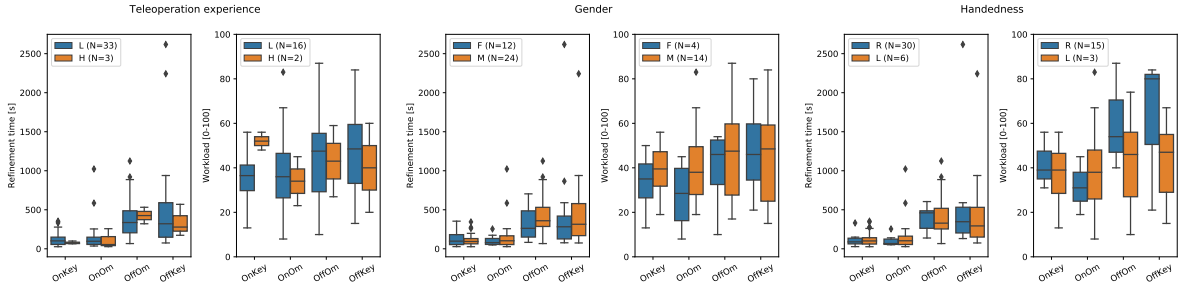


Figure E.19: Influence of teleoperation experience (High/Low), gender (Male/Female) and handedness (Right/Left) on the refinement time and workload.

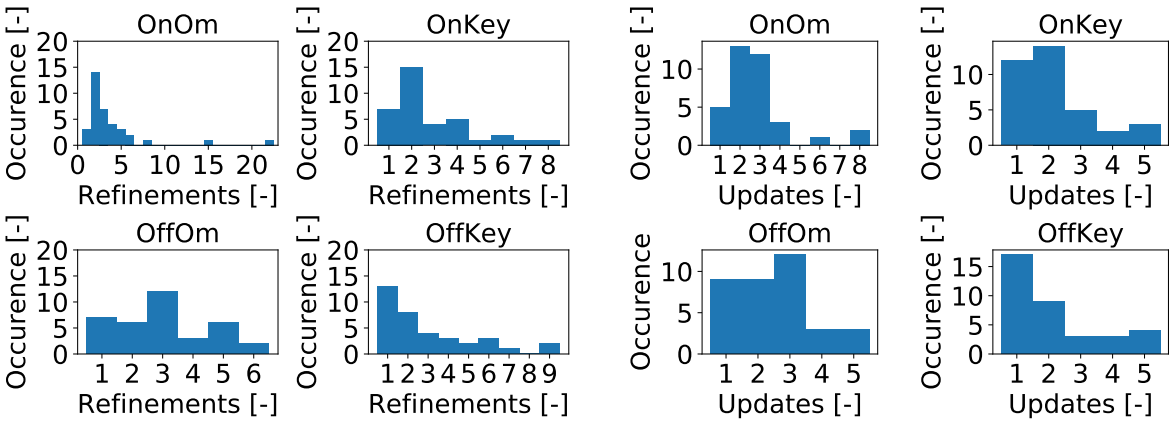


Figure E.20: Number of updates and refinements needed to successfully update one model, displayed per method.

			Gaming experience		Teleoperation experience		Field of study/work	
			L ($N = 21$)	H ($N = 15$)	L ($N = 33$)	H ($N = 3$)	T ($N = 27$)	NT ($N = 9$)
OnStyl	**RT [s]**	M	129.40	55.87	96.62	54.58	82.50	126.01
		25	76.40	51.80	55.91	41.08	53.97	76.40
		75	167.07	89.56	148.77	156.11	144.50	167.07
			L ($N = 30$)	H ($N = 24$)	L ($N = 48$)	H ($N = 6$)	T ($N = 39$)	NT ($N = 15$)
	WL [0-100]	M	32.5	42.5	36.0	34.0	34.0	38.0
		25	25.0	30.5	26.5	23.0	25.0	31.0
		75	45.0	46.5	46.5	45.0	45.0	42.0
			L ($N = 21$)	H ($N = 15$)	L ($N = 33$)	H ($N = 3$)	T ($N = 27$)	NT ($N = 9$)
OnKey	**RT [s]**	M	101.59	75.49	101.59	69.57	84.05	141.71
		25	55.65	62.13	55.65	67.21	57.53	94.27
		75	149.50	119.70	149.50	84.68	126.78	149.50
			L ($N = 30$)	H ($N = 24$)	L ($N = 48$)	H ($N = 6$)	T ($N = 39$)	NT ($N = 15$)
	WL [0-100]	M	34.0	40.0	36.5	52.0	34.0	40.0
		25	31.0	35.75	29.75	48.0	31.0	39.0
		75	50.0	45.75	41.25	56.0	45.0	50.0
			L ($N = 21$)	H ($N = 15$)	L ($N = 33$)	H ($N = 3$)	T ($N = 27$)	NT ($N = 9$)
OffStyl	**RT [s]**	M	385.71	320.83	336.59	425.88	336.58	385.71
		25	204.93	264.35	204.93	373.36	229.17	301.89
		75	591.06	441.20	487.30	478.95	506.08	487.30
			L ($N = 30$)	H ($N = 24$)	L ($N = 48$)	H ($N = 6$)	T ($N = 39$)	NT ($N = 15$)
	WL [0-100]	M	49.0	39.5	47.5	43.0	40.0	53.0
		25	40.0	27.0	29.25	27.0	27.0	52.0
		75	54.0	61.25	55.5	59.0	59.0	54.0
			L ($N = 21$)	H ($N = 15$)	L ($N = 33$)	H ($N = 3$)	T ($N = 27$)	NT ($N = 9$)
OffKey	**RT [s]**	M	335.98	278.49	320.78	278.49	252.38	590.66
		25	137.14	174.66	148.37	225.82	142.76	308.11
		75	608.28	397.00	590.66	424.08	392.66	866.16
			L ($N = 30$)	H ($N = 24$)	L ($N = 48$)	H ($N = 6$)	T ($N = 39$)	NT ($N = 15$)
	WL [0-100]	M	51.5	44.0	48.5	40.0	39.0	67.0
		25	37.0	20.75	33.0	20.0	21.0	67.0
		75	67.0	54.0	59.5	60.0	50.0	80.0

Table E.2: Refinement time (RT) and workload (WL) of the different methods as function of the field of work, gaming and teleoperation experience. Field of work is either technical (T) or non-technical (NT), gaming and teleoperation experience is either low (L) or high (H).

Bibliography

[1] trojan03, *Python implementation of probabilistic motor primitives including a ros overlay.* 2016. [Online]. Available: https://github.com/baxter-flowers/promplib.

[2] F. team, *Python implementation of contextual probabilistic motor primitives*, 2018. [Online]. Available: https://github.com/trojan03/contextual-promp.

[3] M. Ewerton, G. Maeda, G. Kollegger, J. Wiemeyer, and J. Peters, "Incremental imitation learning of context-dependent motor skills", in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, IEEE, 2016, pp. 351–358.

[4] G. Maeda, M. Ewerton, T. Osa, B. Busch, and J. Peters, "Active incremental learning of robot movement primitives", 2017.

[5] sniekum, *This ros package is a general, robot-agnostic implementation of dynamic movement primitives (dmps).* 2016. [Online]. Available: https://github.com/sniekum/dmp.

[6] *Blog explaining dmps.* [Online]. Available: https://studywolf.wordpress.com/2013/11/16/dynamic-movement-primitives-part-1-the-basics/.

[7] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration", *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[8] H. Tomé, L. Marchionni, and A. R. Tsouroukdissian, "Whole body control using robust & online hierarchical quadratic optimization", in *IROS14 International Conference on Intelligent Robots and Systems*, 2014, pp. 14–18.

[9] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series.", in *KDD workshop*, Seattle, WA, vol. 10, 1994, pp. 359–370.

[10] M. P. Nemitz, R. J. Marcotte, M. E. Sayed, G. Ferrer, A. O. Hero, E. Olson, and A. A. Stokes, "Multifunctional sensing for swarm robots using time sequence classification: Hoverbot, an example", *Frontiers in Robotics and AI*, vol. 5, p. 55, 2018, ISSN: 2296-9144. DOI: 10.3389/frobt.2018.00055. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2018.00055.

[11] slaypni, *A python implementation of fastdtw*, 2019. [Online]. Available: https://github.com/slaypni/fastdtw.

[12] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space", *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[13] M. Kyrarini, M. A. Haseeb, D. Ristić-Durrant, and A. Gräser, "Robot learning of industrial assembly task via human demonstrations", *Autonomous Robots*, vol. 43, no. 1, pp. 239–257, 2019.

[14] M. Kameric, *Dishwasher cad model*, https://grabcad.com/library/dishwasher-4, 2015.

[15] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. R. Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, *et al.*, "Ros_control: A generic and simple control framework for ros", 2017.

[16] NASA, *Nasa task load index instruction manual*, https://humansystems.arc.nasa.gov/groups/TLX/downloads/TLX_pappen_manual.pdf, (Accessed on 10/06/2020).

[17] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: fundamental algorithms for scientific computing in python", *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[18] *Spline interpolation method*, http://mathforcollege.com/nm/mws/gen/05inp/mws_gen_inp_ppt_spline.pdf, (Accessed on 10/12/2020), Autar Kaw, Jai Paul.

Glossary

DMP Dynamic Movement Primitive. 35, 36

DTW Dynamic Time Warping. 22, 28, 40, 41, 42, 43, 52

GUI Graphical User Interface. ii, 49, 50

LWPR Linearly Weighted Projection Regression. 36

LWR Linearly Weighted Regression. 36

ProMP Probabilistic Movement Primitive. 18, 19, 20, 22, 32, 40, 43, 45

RFWR Receptive Field Weighted Regression. 36