Intuitive Imitation Learning for One-Handed and Bimanual Tasks Using ProMPs

Intuitives Imitation Learning für ein- und beidhändige Aufgaben basierend auf ProMPs

Master-Thesis von Moritz Knaust
Electrical Engineering and Information Technology

1. Gutachten: Prof. Jan Peters PhD, Intelligent Autonomous Systems
2. Gutachten: Prof. Dr. Jürgen Adamy, Regelungsmethoden und Robotik
3. Gutachten: M.Sc. Dorothea Koert, Intelligent Autonomous Systems

November 2019

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Intuitive Imitation Learning for One-Handed and Bimanual Tasks Using ProMPs
Intuitives Imitation Learning für ein- und beidhändige Aufgaben basierend auf ProMPs

Vorgelegte Master-Thesis von Moritz Knaust

1. Gutachten: Prof. Jan Peters PhD, Intelligent Autonomous Systems
2. Gutachten: Prof. Dr. Jürgen Adamy, Regelungsmethoden und Robotik
3. Gutachten: M.Sc. Dorothea Koert, Intelligent Autonomous Systems

Tag der Einreichung:

Erklärung zur Master-Thesis

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

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

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

In the submitted thesis the written copies and the electronic version are identical in content.

Datum / Date: Unterschrift / Signature:

_____ _____

Abstract

Intelligent assistive robots can potentially provide assistance to elderly persons and caregivers. However, the variety of different tasks for the robots renders pre-programming of all tasks difficult and it is desirable that assistive robots can learn new tasks also directly from the caregivers. Imitation Learning is a promising and widely used approach to train robots without programming every motion manually. One big challenge in Imitation Learning is the generation of demonstrations which are critical for a successful training process. Giving demonstrations can thereby often be difficult and requires background knowledge about the used robotic system and the applied algorithms.
To this end, this thesis proposes an intuitive system that allows completely inexperienced users to teach the robot new tasks which consist of several different steps. In particular, this thesis also introduces a simple and intuitive high-level task structure and an automated teaching process to train the different actions and movements with kinesthetic teaching. Besides executing point-to-point motions the system can also be used to create generalizable trajectories based on Probabilistic Movement Primitives(ProMPs). The ProMPs can thereby be trained for both, one-handed and bimanual motions. The whole system is implemented based on Behavior Trees which allows the creation of modular and reactive processes. The framework is tested in a user study with users who are inexperienced in working with robots. The experimental evaluation shows that users can understand the task structure and teaching processes and are able to train new tasks completely on their own. The users are capable of training and successfully using new ProMPs, whereas the programming of point-to-point movements often leads to collisions with the environment. On this basis, some improvements are proposed to make the training and execution of ProMPs more reliable.

Zusammenfassung

Intelligente Assistenzroboter sind potentiell dazu in der Lage Senioren und Pfleger zu unterstützen. Die Vielzahl der verschiedenen Aufgaben der Roboter erschwert jedoch die Vorprogrammierung aller Prozesse und es ist wünschenswert, dass Assistenzroboter neue Aufgaben auch direkt von den Pflegern lernen können. Imitation Learning ist ein vielversprechender und weit verbreiteter Ansatz, um Roboter zu trainieren, ohne jede Bewegung einzeln manuell zu programmieren. Eine große Herausforderung beim Imitation Learning ist das Erzeugen von Demonstrationen, die aber für einen erfolgreichen Trainingsprozess entscheidend sind. Das Erzeugen der Demonstrationen kann dabei oft schwierig sein und benötigt Hintergrundwissen über das eingesetzte Robotersystem und die angewandten Algorithmen.
Zu diesem Zweck wird in dieser Arbeit ein intuitives System vorgestellt, mit dem völlig unerfahrene Benutzer dem Roboter neue Aufgaben beibringen können, die aus mehreren verschiedenen Schritten bestehen. Insbesondere wird in dieser Arbeit auch eine einfache und intuitive übergeordnete Aufgabenstruktur sowie ein automatisierter Trainingsprozess vorgestellt, um die verschiedenen Aktionen und Bewegungen mit Kinesthetic Teaching zu trainieren. Neben Punkt-zu-Punkt-Bewegungen können mit dem System auch generalisierbare Trajektorien auf der Basis von Probabilistic Movement Primitives (ProMPs) erstellt werden. Dabei können die ProMPs sowohl für einhändige als auch für beidhändige Bewegungen trainiert werden. Das gesamte System basiert auf Behavior Trees, die die Erstellung modularer und reaktiver Prozesse ermöglichen.
Das Framework wird in einer Nutzerstudie mit Anwendern getestet, die im Umgang mit Robotern unerfahren sind. Die experimentelle Auswertung hat gezeigt, dass Benutzer die Aufgabenstruktur und die Lehrprozesse verstehen und neue Aufgaben völlig selbstständig trainieren können. Die Benutzer können ebenfalls neue ProMPs trainieren und erfolgreich einsetzen, während die programmierten Punkt-zu-Punkt-Bewegungen häufig zu Kollisionen mit der Umgebung führen. Auf dieser Grundlage werden einige Verbesserungen vorgeschlagen, um das Training und die Ausführung von ProMPs zuverlässiger zu gestalten.

Contents

Figures and Tables

List of Figures

List of Tables

1 Introduction

1.1 Motivation

One of the most common visions on how technology could improve our everyday life imagines service robots that assist people in their households and do all the tasks we do not like to do[1, 2]. This vision was often repeated in all kinds of stories, movies, books and future discussions, but it is far away from the way robots are used nowadays. At the moment the only robots that are widespread, are industrial robots in big production processes. They can do one specific task and repeat it very often. This is only working in a well known and structured environment, because most industrial robots cannot handle new situations.
Industrial robots are programmed by highly qualified engineers and every task the robot should perform has to be implemented by experts. So the whole programming process becomes expensive and time-consuming. This working process is not applicable to future service robots, because they have to perform a lot of different tasks and will often face unknown situations. So it must be possible for users, who are no experts in robotics, to train the robot new tasks.
One possibility is that the robot trains the task on its own using advanced machine learning techniques like reinforcement learning[3]. This can result in impressive robot actions, but it also has disadvantages, like the high number of needed training runs, the high number of failed tasks in the training period and the complex algorithm itself, which often also needs tuning by experts. So this learning technique hardly can be applied to robot household applications.
Another approach to train a robot is Imitation Learning[4]: A human teacher can train the robot a new task while the robot imitates the behavior of the human. Most Imitation Learning systems use demonstrations which are created by a teacher who is moving the robot manually. Imitation Learning in general also has different challenges. Most algorithms are highly dependent on the given demonstrations[5, 6], so this is the most important step. The generation of the demonstration itself often is not easy, because it is needed to remote control the robot, which is often difficult. Another problem is that the demonstrations cannot be too similar or too different, because then the robot can only learn a very specific action or no action at all. It can be seen that Imitation Learning offers good possibilities, but also is a challenge for inexperienced users.
This challenge becomes even greater if a two-armed robot is used and most humanoid robots[7, 8], of course, have two arms. For bimanual tasks giving demonstrations becomes harder, because the user has to move two arms at the same time and has to focus on the synchronization of both arms.
But training new robotic tasks not only consists of the generation of new robotic movements, but the user also has to define the high-level task procedure. This procedure, in general, is a combination of different movements and gripper actions. The high-level task also has to be defined by inexperienced users, so an understandable and simple task structure must be applied.

Figure 1.1.: One planned scenario is that KoBo is autonomously serving drinks to the people, so they are encouraged to drink enough over the day.

Figure 1.2.: Kobo also should learn new tasks which are trained by nurses. This is the main motivation of this project.

All in all the whole teaching process of a new task must be intuitive, easy to understand and controllable by inexperienced users without robotic background knowledge.
The motivation behind this project is the KoBo robot, a robot that should assist the people and the nurses in a home for elderly people. In the beginning, KoBo should do three different tasks: Serving meals for lunch and dinner, serving drinks and snacks between the meals (Compare Figure 1.1) and buying and delivering things from a small shop. But KoBo also should learn new tasks from the nurses, like shown in Figure 1.2. The development of a framework that allows every human to teach the robot new tasks is the main goal of this project.
The used KoBo robot has two Franka Emika Panda[9] arms where each arm has seven degrees of freedom. These robotic arms are mounted on a mobile platform, so the robot can drive around in the building. For the environmental recognition, different sensors and an advanced image recognition system can be used.

1.2 Structure

This thesis is split into four important main parts and a conclusion:

Chapter 2: Foundations & Related Work
: In Chapter 2 the related work concerning Imitation Learning in general, concepts for robot learning systems and trajectory representation are discussed. Afterward, the two most important frameworks used in the project are explained: Probabilistic Movement Primitives for the generation of trajectories and Behavior Trees for the high-level task structuring.

Chapter 3: Own approach
: In Chapter 3 the developed Imitation Learning system is described. At first, the representation of high-level tasks is explained and in the second section a simple process of training a new task is developed. This also includes a special teaching process for ProMPs which should make it possible for everyone to train new ProMPs. The third section describes the development of a new robot controller used for the teaching and execution steps.

Chapter 4: Implementation
: Chapter 4 shortly describes the robot setup, the structure of the developed software and the user interface, used in the training process.

Chapter 5: Experiments
: The whole system is evaluated in Chapter 5. This is done by a user study with inexperienced users. The results are analyzed concerning the training success, common errors and the user feeling about the teaching process. Besides, the trained ProMPs are analyzed to evaluate if the training of ProMPs by inexperienced users is feasible.

Chapter 6: Conclusion & Future Work
: In the end, Chapter 6 concludes all findings and reports future work that can be done to further improve the framework itself and Imitation Learning with inexperienced users in general.

2 Foundations & Related Work

This chapter presents important related work about Imitation Learning with robots and explains two important frameworks used in this project: Probabilistic Movement Primitives and Behavior Trees.

2.1 Related Work

In this section an overview about related work in the area of Imitation Learning in general and Imitation Learning with robots is given. At first different developed teaching concepts and user studies on teaching robots are presented. The second part focuses on the representation of the trained motions with Movement Primitives.

2.1.1 Imitation Learning

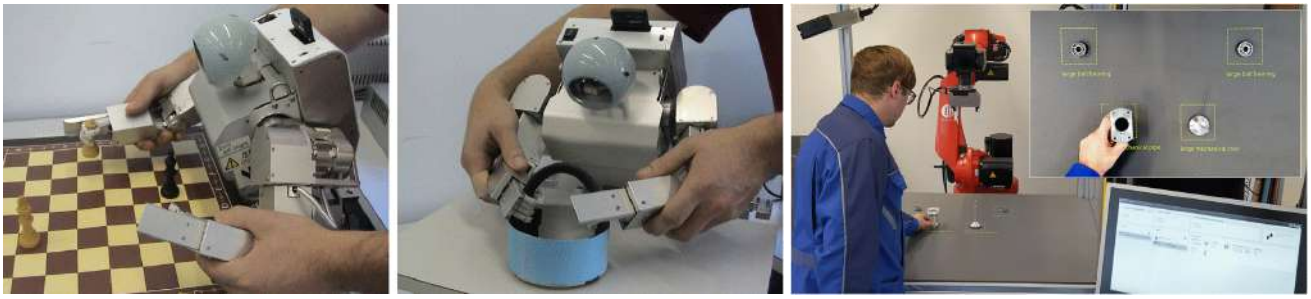
The programming of autonomous systems like robots becomes more challenging for a more complex and unstructured environment. Programming the desired behavior with classical methods is often not possible, while humans can easily perform the desired task. The main idea of Imitation Learning is to use the knowledge of the human expert who actually can perform the desired task and porting it to the robot. For humans, it also often is possible to give demonstrations on how the robot should perform the desired task.
So the knowledge of people with expert knowledge in one field can be used to train the robot, even if they have no knowledge of robot programming. In many further works[4, 10] it is proven that this kind of programming a robot is a promising approach. One of the first systems using this technique successfully is the autonomous car ALVIN[11], which learn to map sensor inputs to a steering output and was able to drive driver-less.
In general, different Imitation Learning approaches can be distinguished by the goal they should learn and the methods they use. A detailed overview of a lot of different approaches is given in [12]. The simplest Imitation Learning approach is behavior cloning which is using supervised learning to clone the demonstrated behavior[5, 13]. It is working well with applications where the whole used state-space is covered by demonstrations but normally fails if the system leaves the trained part of the state-space. So it should be only used in clear environments there the failure has no fatal consequences.
To improve the learned policy, the behavior can be trained afterward with an interactive human expert that gives feedback to the system[14, 15]. This is called interactive Imitation Learning or direct behavior cloning and can help to create policies that generalize a much better to unknown situations. The disadvantage is that the learning proves needs a human expert who can assess how well the robot executed his task.
If this is not possible, Inverse Reinforcement Learning[3] can be used to learn a reward function from the given demonstrations. Then the robot can train his policy using this reward function and reach the desired goal. These methods can achieve very impressive results, but they are also much more difficult to use than for example behavior cloning.

2.1.2 Robot Learning

Imitation Learning for robots is a complex task, especially if robot arms with a high number of degrees of freedom are used. In general, the robot has to learn a high-level procedure and additionally some low-level robotic movements. Both parts are very important for successful robot functionality. The development of the whole teaching process becomes more difficult in our case because we want to focus on an intuitive and simple to use teaching process. Section 2.1.2.1 deals with the teaching of robot behavior in general, the robot movement teaching is described in sections 2.1.2.3 and 2.1.3.

2.1.2.1 Teaching Concepts

In [16] a programming by demonstration framework is developed with the focus on generalizing the acquired knowledge to different contexts. To do this, it extracts the relevant features of the demonstrations with probabilistic methods. The framework is tested on a humanoid robot with different manipulation tasks and shows good behavior. One problem for our use case is that there is no focus on an intuitive learning process, so the generation of demonstrations may not be easy to perform.
The intuitive learning process is focused in [14]. They developed a framework to teach high-level tasks to the robot in a similar way humans teach each other. At first, the human teacher gives a demonstration, then then the robot can practice



(a) Teaching different simple manipulation tasks[16]. **(b)** Teaching industrial assembly tasks[17].

Figure 2.1.: In [16] a small humanoid robot is used to evaluate the generalization of the acquired knowledge. Kinesthetic teaching is used to give the demonstrations, but no intuitive teaching process is used.
The teaching of industrial assembly tasks is focused in [17]. The programming is done on object level and needs CAD models of all objects. This is a good and intuitive approach for industrial applications, but not applicable in an unstructured household environment.

the behavior under the supervision of the teacher. The teacher gives feedback or can generate additional demonstrations if the behavior of the robot is not good enough or it should learn a more general behavior. This system is tested on a mobile robot with a GoTo and pickup task and is working well. The approach of imitating the human to human teaching process is promising, but it is only tested with a very small set of actions. A robotic arm is much more complex because it has more degrees of freedom and can execute a lot more actions. So the development of a similar teaching process for robotic arms may be hard.
In [18] the intuitive teaching process is also focused. The robot should learn new tasks by observing humans and listening to speech commands. The robot is controlled by speech commands and also the generation of behavior networks is done by a set of speech commands. Besides, the human can interact with the robot to let the robot learn from demonstrations. This concept also is a promising approach in terms of an intuitive teaching process, because the robot can be trained only by speech and no other input is needed. The whole process uses some dialog and the robot asks questions to learn, which makes it also similar to teaching a human. An open question is if it is straightforward for inexperienced users to create the task structure by speech. Another point is that the system also is tested on a mobile robot with only a small set of actions and not on a robotic arm.
In [19] the task level learning and planning are treated. The designed system learns the goal of the task and uses a task planner to reach the learned goal. Some tasks can have underlying constraints that must be fulfilled to reach the goal. These constraints can be defined directly by the teacher or are detected automatically by the robot from multiple observations. To do this the demonstrations are abstracted to a state sequence, which is then used to learn the sequence order and some underlying constraints. The system is tested on a mobile robot with a mounted robotic arm and some pick and place tasks. So it is proven that the system can be used for more complex tasks. This approach is not focusing on an intuitive process especially, but the whole learning process is very promising.
In [17] a framework for using robots in industrial small lot production is developed with a focus on reducing the needed robot expert knowledge. Instead of teaching the task-space position the programming is done based on the object level. So the human teaches how the robot should manipulate the different objects to assemble them. This system is based on CAD models of the objects, so it is not applicable to our task. Another framework with industrial background is presented in [20, 21] which should allow novel end users to train complete robotic tasks. It uses object recognition and predefined *SmartMoves* but it is not possible to train full generalizable trajectories.

2.1.2.2 User Studies

The previously stated works explored the robot learning process in general and also worked on human-like concepts, but they were not looking at the user and which methods are preferred by him. In the case of this project, especially inexperienced users with no robot or computer science knowledge are important, because this is the target group of our system. In the following, a short overview of some user studies about this topic is given.
The study [22] explored the satisfaction of nave users teaching some simple manipulation tasks to a small humanoid robot. The main goal was to understand how the user can teach successful and also is satisfied with the result. To train the robot learning by demonstration based on direct contact interaction is used. To control the whole teaching process speech output and commands are used. The interesting result for this project is that most users were able to carry out the teaching task successful. Another important point is that the users are willed to show more demonstrations to the

(a) User-study teaching a humanoid[22]

(b) User-study on assisted kinesthetic teaching[24]

Figure 2.2.: The user satisfaction is evaluated in the user-study [22], shown in Figure 2.2a. Most users were able to carry out the teaching task successful using kinesthetic teaching. In [24] an assisted kinesthetic teaching mode is used to train complex robot trajectories with unexperienced users, like shown in Figure 2.2b. One focus is the avoidance of collisions with the environment.

robot to create better results. This is a gratifying result because giving multiple demonstrations is needed for learning trajectories that can be generalized well to new situations.
An interesting point is that most users were unsatisfied with the lack of technical background information they had. This makes sense because the users were not experienced with robots, but it also caused some fears of contact with the robot, because they are afraid of doing things wrong. So it should be helpful to give some technical background knowledge to the user before they use the robot. The open question is, which information is needed and should be given to them without overtaxing them.
In the study, a highly guided concept is used to teach the robot. This seems to perform well with inexperienced users, so it also should be considered for this project.
In [23] the teaching with teleoperation and kinesthetic teaching is compared. To do this the same learning process is used for both methods, but the demonstrations are created with different methods. The study is done with a simple manipulation task on a 7-DOF robot arm. The main results are that the users can give demonstrations with kinesthetic teaching more successful and faster than with teleoperation.
These were the objective results, but also the subjective feeling of the users is important because they must accept the learning system. The results about these are that the users think that kinesthetic teaching also is easier and that they can give more accurate demonstrations using it. So it seems like kinesthetic teaching is the more promising approach to give demonstrations.
The user study [24] focuses on kinesthetic teaching. They developed a special kinesthetic mode that assists the user during the demonstration recording phase: At first, the users train the desired null-space position for important points in the task space. After that, the user only has to move the end-effector and the null-space of the robot is controlled automated. In this scenario, this technique is used to avoid collisions. The developed system is very powerful and makes it simple for the user to generate complex trajectories. This also is proven by the user study: Most users were able to train the given task successful.
In [25] also an assisted teaching mode is developed: The user gives one demonstration, then the user repeats the movement a few times with increasing stiffness and the user can improve the robot movement. Using this system the users created much better demonstrations. Another important point is that the user is satisfied with the assistance and prefers using it.
Both studies show that kinesthetic teaching, in general, can be used successfully by most users. Using some assistance techniques to support the user can increase the quality of the demonstration a lot.

2.1.2.3 Concepts for Demonstration Generation

The generation of demonstrations is one of the most important parts of the Imitation Learning process. Except for kinesthetic teaching, there are a lot of other methods to generate demonstrations. This section gives an overview about them.
A lot of systems use teleoperation to control the robot in the teaching phase. On concept using an exoskeleton is shown in [26]. The human arm joints are recorded by an exoskeleton, which allows precise measurement and avoids a few problems concerning the joint calculation. But all in all the usage of an exoskeleton is not rational in the most use-cases. Especially if the learning process should be intuitive, the exoskeleton is not preferred.
A similar approach is used in [27]. Instead of using an exoskeleton to record the human joint angles, IMUs are used. This makes the system much more flexible and more comfortable to use. The human joints are mapped to the robot and the robot is imitating the position of the human arm. This concept also is used in [28], but now extended to two arms.

Figure 2.3.: Different teleoperation systems can be user to remote control robots. The two left images show a exoskeleton for human arm recording which is used in [26]. A similar concept is used in [28], but the human joint recording is done by IMUs. A completely different approach is the robot control in virtual reality[29], shown in the right image.

Both arms can be controlled independently with the two human arms, but it also exists a special coupled mode, where one arm follows the other with a fixed relative end-effector position. This system makes it much easier to control both robot arms synchronous.
One general problem of all methods that map the human joints to the robot arm is that this mapping cannot be done directly, because the geometries of the human and robot arm differ. Also, the geometry of a human arm is not equal for every human. This makes the teleoperation process more difficult.
A completely different approach is used in [29]. In this project the demonstration is created in virtual reality, using a headset and some hand tracking devices. The robot grippers can be controlled by the hand tracking devices while the user is seeing the objects in virtual reality. This system may work well, but in general, this setup is very complex and the usage of virtual reality is very unintuitive for new users.
To avoid all technical problems concerning robot teleoperation, the learning also can be done directly from the human movements, which is shown in [30]. The human joints are recorded, then the learning is done and the learned behavior is executed on the robot. This procedure may work for some tasks, but it seems to not be very reliable and save.
Because of the mentioned problems the most systems use kinesthetic teaching to generate the demonstrations. In [31] and [32] it is shown that kinesthetic teaching can be used to train complex tasks and the user studies [24] and [25] occupy that it can be done also by inexperienced users.

2.1.3 Trajectory Representation

After generating the demonstrations the second important part of Imitation Learning[16, 6] is the modeling of the trained movement and the generation of new trajectories. The main goal when learning from demonstrations is to reproduce and generalize the demonstrated behavior. Generalization means that the movement should deal with a new situation that is not exactly trained but is similar to the demonstrations. Often needed examples are new end-points or the definition of via-points.

2.1.3.1 Movement Primitives

An approach often used in this context are Movement Primitives[13, 33, 34], which are a Behavior Cloning method. A Movement Primitive describes one part of a more complex robotic movement and this should make it possible to compose a complex robot behavior out of a set of simple blocks. To do this it is necessary to combine two Movement Primitives, which is often done by activating both and blending them in each other to get a smooth movement. As told before, the modulation to new target, via-points and speeds must be possible. Movement Primitives are in general a time-or-phase based[35] representation of movements.
One of the first Movement Primitive frameworks that can handle all mentioned challenges were Dynamic Movement Primitives(DMP)[13]: They use simple nonlinear systems to model the movement and were used in the beginning to model the motor behavior of robots, but they also can be used for more high-level trajectories. DMPs can learn from demonstration data and support spatial and temporal coupling. Although generalization to new end-points is possible. In a lot of projects it is proven that DMPs can describe complex tasks, for example ball throwing[36] and pancake flipping[37].
A newer approach working on the same problem are Probabilistic Movement Primitives(ProMP)[5, 38]. They are a general probabilistic framework for representing and learning Movement Primitives and one ProMP is a distribution over a set of demonstration trajectories. Because they use probabilistic methods, the modulation can be done simply by

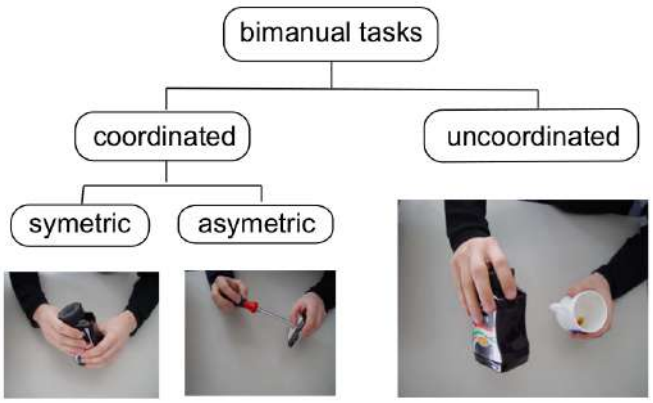


Figure 2.4.: Bimanual Tasks can be classified in three different categories[46], depending on the needed synchronization between both hands.

conditioning. For temporal modulation also a phase variable is used. One further advance of using probabilistic methods is that they can describe the variance of the movement. This can be used to evaluate how safe the execution is at different points.
The capability of ProMPs can be extended by a lot of different extensions like Interaction ProMPs[39] for interactions between robots and humans, online adaption of ProMPs[40, 41] and sequencing ProMPs[42, 43].
There also exist some newer and less common Movement Primitives like the Impedance Based Movement Primitives[44] where task-parametrized statistical dynamical systems are used. Another probabilistic framework are Kernelized Movement Primitives[45]. The structure is similar to ProMPs, but they reduced the number of open parameters by using kernel-based nonparametric methods. The main advantages are the handling of unpredicted situations like obstacles or external perturbations and the extrapolation capabilities.

2.1.3.2 Two Arm Movement Primitive Concepts

Bimanual tasks can be split into three different groups, which are described in [46] and shown in Figure 2.4: The first group includes uncoordinated two-arm manipulations which includes all tasks where both hands fulfill a manipulation but no coordination is needed. The tasks for each hand can be executed independently, so an uncoordinated two-hand manipulation can be split into two single-arm manipulations. This case can be treated by normal Movement Primitives.
If synchronization between both arms is needed, the task is called a coordinated two-arm manipulation, which can again be divided into two groups: In symmetric coordinated tasks both hands manipulate the same object and create a closed kinematic loop. In asymmetric coordinated tasks, both hands manipulate different objects.
For coordinated bimanual tasks, it is necessary to define a Movement Primitive that can describe the movements of both arms and couples them. The usage of two independent Movement Primitives will directly lead to different problems, especially with the synchronization and execution of the two arms. On this topic, much less research is done, compared to Movement Primitives in general.
One simple approach is the usage of one Movement Primitive with doubled dimension[39] and each half of the Movement Primitive describes one arm. This introduces a stochastic coupling of both arms and also allows generalization to new situations for both arms, but it is not using coupling in the execution phase.
This can be necessary if the robot interacts with objects because a slightly wrong path of one arm can lead to high tension on the carried object. To avoid this problem, two independent Movement Primitives can be used which are coupled over force feedback[47]. Other approaches are the usage of DMPs which are coupled by a formation control[48] or Movement Primitives together with symmetric control[49] to generate a system that is compilable in the absolute task but stiff in the relative position of both arms.

2.2 Probabilistic Movement Primitives

A common approach to represent a robot movement are Movement Primitives. They can be used to generate robot trajectories for one task and are adaptable to new situations, for example, a new end-position of the task. An often-used framework are Dynamic Movement Primitives(DMP)[50]. They describe the movement as a dynamic system, guarantee stability and can be adapted to other end positions or temporal scaling.
The adaption of DMPs to new situations is possible, but not very smooth. Because of this the newer framework of Probabilistic Movement Primitives(ProMP)[5] is used in this project:
ProMPs use probabilistic methods to describe the movement. At first, every trajectory is approximated by a set of basis functions and the belonging weights:

$$\boldsymbol{y}_t = \boldsymbol{\Phi}_t^T \boldsymbol{w} + \boldsymbol{\epsilon}_y \tag{2.1}$$

$\boldsymbol{y}_t$ is the trajectory, $\boldsymbol{\epsilon}_y$ $\mathcal{N}(0, \boldsymbol{\Sigma}_y)$ i.i.d. Gaussian noise and $\boldsymbol{\Phi}_t$ is the time-variant basis function vector. As basis functions a set of normalized Gaussian functions is used, shown in Figure 2.5. The number of basis functions is fixed for one ProMP, but it can be useful to change it for different ProMPs depending on the trajectory complexity. To avoid timing problems, a phase variable z is used that equals a normalization of the trajectory duration. So one Trajectory $\boldsymbol{y_t}$ can be described only by the calculated weights and the duration.

$$p(\boldsymbol{\tau}|\boldsymbol{w}) = \prod_t \mathcal{N}\left(\boldsymbol{y}_t|\boldsymbol{\Phi}_t^T\boldsymbol{w}, \boldsymbol{\Sigma}_y\right), \qquad p(\boldsymbol{\tau};\boldsymbol{\theta}) = \int_w p(\boldsymbol{\tau}|\boldsymbol{w})p(\boldsymbol{w};\boldsymbol{\theta})d\boldsymbol{w} \tag{2.2}$$

ProMPs use probabilistic methods to describe a distribution over the demonstrated trajectories . To do this, several trajectories are needed and every trajectory is described by its weights. The probability to observe a trajectory $\boldsymbol{\tau}$ under a given weight vector $\boldsymbol{w}$ is $p(\boldsymbol{\tau}|\boldsymbol{w})$ with the mean $\boldsymbol{\Phi}_t^T\boldsymbol{w}$ and the observation noise $\boldsymbol{\Sigma}_y$. In the next step, a distribution $p(\boldsymbol{\tau}|\boldsymbol{\theta})$ with the parameters $\boldsymbol{\theta}$ over the weights of all trajectories is established. It can be used to calculate the trajectory mean and variance at each time step.
The previous model is only working for one-dimensional trajectories, but real robot trajectories have a higher number of dimensions. An easy approach is to model each dimension independently and couple them by the phase variable. This is possible and provides good results for easy trajectories, but it would be better to introduce the coupling in the probabilistic model. To do this every dimension i is approximated by a weights vector $\boldsymbol{w}_i$ and the combined weight vector $\boldsymbol{w}$ is defined as $\boldsymbol{w} = [\boldsymbol{w}_1^T, ..., \boldsymbol{w}_n^T]^T$. The basis matrix $\boldsymbol{\Phi}_t$ is extended to a block-diagonal matrix $\boldsymbol{\Psi}_t$ and the observation vector $\boldsymbol{y}_t$ contains the observations for all dimensions:

$$p(\boldsymbol{y}_t|\boldsymbol{w}) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{y}_{1,t} \\ \vdots \\ \boldsymbol{y}_{n,t} \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{\Phi}_t^T & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \boldsymbol{\Phi}_t^T \end{bmatrix} \boldsymbol{w}, \boldsymbol{\Sigma}_y \right) = \mathcal{N}(\boldsymbol{y}_t|\boldsymbol{\Psi}_t\boldsymbol{w}, \boldsymbol{\Sigma}_y) \tag{2.3}$$

Because it is assumed that everything is Gaussian distributed the calculation of $p(\boldsymbol{y}_t|\boldsymbol{\theta})$ can be done easily from example trajectories. Equation (2.2) can be combined with the distribution other the weights $p(\boldsymbol{w};\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$ which also is assumed to be Gaussian.

$$p(\boldsymbol{y}_t;\boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{y}_t|\boldsymbol{\Psi}_t^T\boldsymbol{\mu}_w, \boldsymbol{\Psi}_t^T\boldsymbol{\Sigma}_w\boldsymbol{\Psi}_t + \boldsymbol{\Sigma}_y\right) \tag{2.4}$$

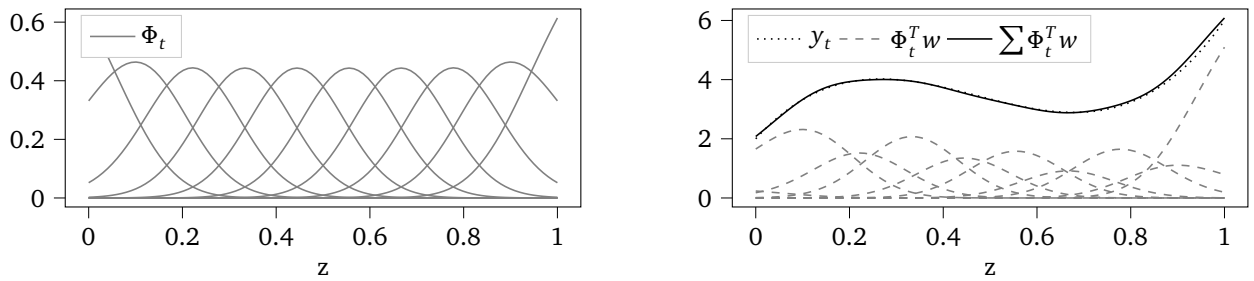


Figure 2.5.: The left figure shows the normalized Gaussian basis functions depending on the phase variable z(the normalized time). The right figure shows an example function y_t, the belonging weighted basis functions and the sum of all basis functions, which results in an reconstruction of y_t.

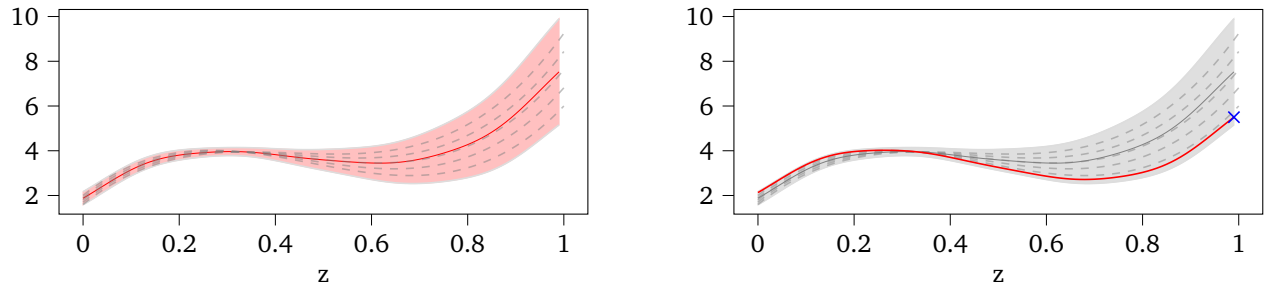


Figure 2.6.: The left figure shows an example ProMP with mean as red line, 2σ surrounding and the example trajectories as dashed lines. On the right side the same ProMP is conditioned on an observation, marked as blue cross. The red line is the mean of the determined trajectory.

In summary, the distribution over all trajectories $p(\boldsymbol{y}_t; \boldsymbol{\theta})$ which represents the ProMP and has the parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w\}$ can be determined by approximating all demonstration trajectories by their weights and then evaluate the mean and variance of the weights. A one dimensional example ProMP is shown in Figure 2.6.
It is possible to generate new trajectories from the ProMP depending on any number of via-points. The via-points are known points on the goal trajectory, often the start or endpoint. In general the observed state is called y_t^* at time t. The observation $\boldsymbol{x}_t = [\boldsymbol{y}_t^*, \boldsymbol{\Sigma}_y^*]$ is added to the probabilistic model of the ProMP and the Bayes Theorem is applied to get the distribution $p(\boldsymbol{w}|\boldsymbol{x}_t^*)$. The conditioning results in a distribution with mean $\boldsymbol{\mu}_w^{[new]}$ and $\boldsymbol{\Sigma}_w^{[new]}$:

$$\boldsymbol{\mu}_w^{[new]} = \boldsymbol{\mu}_w + \boldsymbol{\Sigma}_w \boldsymbol{\Psi}_t \left(\boldsymbol{\Sigma}_y^* + \boldsymbol{\Psi}_t^T \boldsymbol{\Sigma}_w \boldsymbol{\Psi}_t \right)^{-1} \left(\boldsymbol{y}_t^* - \boldsymbol{\Psi}_t^T \boldsymbol{\mu}_w \right) \tag{2.5}$$

$$\boldsymbol{\Sigma}_w^{[new]} = \boldsymbol{\Sigma}_w - \boldsymbol{\Sigma}_w \boldsymbol{\Psi}_t \left(\boldsymbol{\Sigma}_y^* + \boldsymbol{\Psi}_t^T \boldsymbol{\Sigma}_w \boldsymbol{\Psi}_t \right)^{-1} \boldsymbol{\Psi}_t^T \boldsymbol{\Sigma}_w \tag{2.6}$$

The desired trajectory can be calculated using $\boldsymbol{\mu}_w^{[new]}$. An example of a conditioned ProMP is shown in Figure 2.6. You can see the trained ProMP and the new determined trajectory using conditioning. The observation is matched exactly and the whole new trajectory is completely smooth and lies within the trained ProMP. This is a huge benefit across from other frameworks like ProMPs because they do not show this behavior.
You can also combine and blend two different ProMPs, but this functionality is not used in this project. The belonging methods are described in [5].

2.3 Behavior Trees

Behavior Trees are a new way to structure the behavior of autonomous agents, like robots or virtual entities in video games. They are used to control the switching between different subtasks and create a modular and reactive system.
They were developed by the game industry to increase the modularity of Non-Player-Character control structures. Modularity is a key attribute in this field because it allows parallel development and much easier testing of the very complex game software. After the introduction in games, they were also used by academia to control different robotic applications[20, 51]. This is a very similar use case because in both applications you have an agent that should perform different subtask and has to make decisions about his behavior.
At first, the structure and functionality of Behavior Trees are explained, after that, it should be compared to more classical approaches like State-Machines.

2.3.1 Classical Formulation of Behavior Trees

The classical Behavior Tree, like it is introduced in [52], only needs a very small set of components to generate a task. As the name says, the task is structured like a tree: The tree has internal nodes and leaf nodes. Internal nodes control the procedure and which subtask should be executed. So they are called *Control Flow Nodes*. The leaf nodes are used to execute any action, they are called *Execution Nodes*.
Each *Control Flow Node*, except the root node, has one *parent* and one or more *children*. In the tree graphic, the child nodes are placed below the node and the parent node is placed on top of it. In Figure 2.7 a simple tree which explains this structure is shown.
The Behavior Tree is executed following a special algorithm: The root node generates a signal called *tick* with a given frequency. This *tick* is sent to its first children and nodes are only executed if they receive the *tick* signal. The first child is the left one in the tree graphic. If a node is executed it can either return immediately *Success* or *Failure* or it can return *Running* until the action is finished.
Depending on the kind of *Control Flow Node* used, the *tick* shows a different behavior after one node has finished. In the classical formulation four types of *Control Flow Nodes* are used, they are explained in the following.
The *Sequence*-node like shown in Figure 2.7 is the most important construct and is symbolized by an arrow. It ticks the children from the left to the right, until one child returned *Running* or *Failure*. The next child only is ticked if the child before returned *Success*. If one child returns *Running* the whole *Sequence* also returns *Running* to its parent. If one node failed, the *Sequence* is stopped and returns *Failure*. The following nodes in the *Sequence* are not executed anymore.
The *Fallback* node also ticks its children from the left to the right, but it always ticks the next child until one returns *Success* or *Running*. If the child returns *Failure*, the execution is not stopped like in the *Sequence*, instead the next child is ticked. If one child returns *Success* the *Fallback* node also returns success and the following children are not executed. If all children fail, the *Fallback* node returns *Failure*. The *Fallback* node is symbolized by a question mark.
The third type is the *Parallel* node. It ticks all N children at the same time and returns *Success*, if more than M children returned *Success*. M is a predefined threshold value. The *Parallel* node symbol is the $\rightrightarrows$ arrow.
Decorator nodes describe a group of nodes that manipulate the return status of their child. They can only have one child and use a diamond as a symbol as shown in Figure 2.10. The *Decorator* can have two different functions: It can manipulate the return status by a defined rule, for example, it can invert it or can always return *Success*. The second function is that it can manipulate the control flow. An example is that the child node only is ticked n-times or can only be executed for T seconds until the *Decorator* returns *Failure*.
The *execution nodes* are split in *Actions* and *Conditions*. *Actions* can perform anything the developer wants to do and need to be developed for different tasks. They can return *Running*, *Success* or *Failure*. The *Condition* checks a custom condition and immediately returns *Success* or *Failure*.

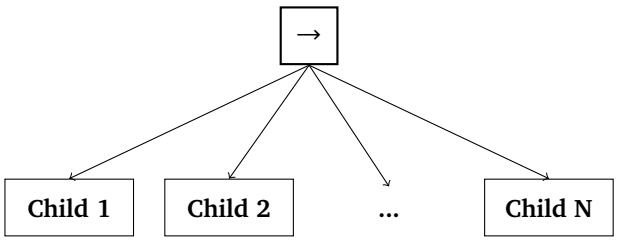


Figure 2.7.: The *Sequence* Node ticks its N children from the left to the right until one child returns *Failure*.[1]

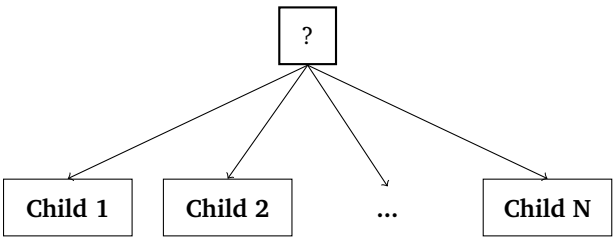


Figure 2.8.: The *Fallback* Node ticks its N children from the left to the right until one child returns *Success*.[1]

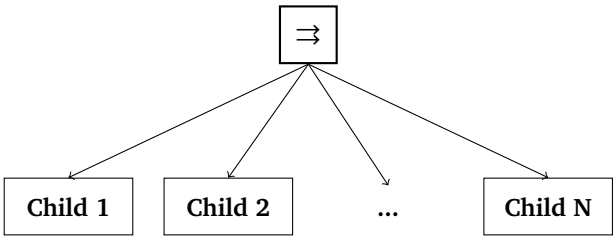


Figure 2.9.: The *Parallel* Node ticks its N children at the same time.[1]

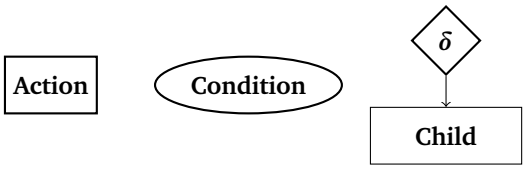


Figure 2.10.: The *Action*, *Condition* and *Decorator* node δ are nodes that can be defined by the user and execute every wished behavior.[1]

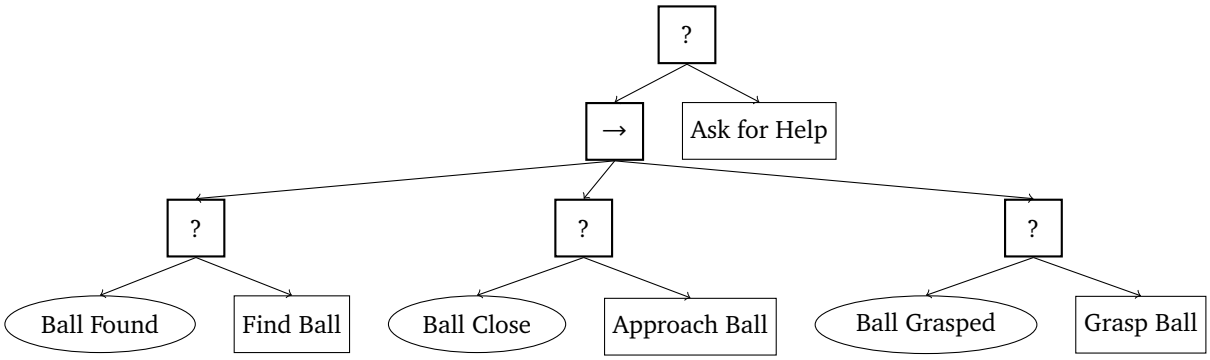


Figure 2.11.: Example Behavior Tree that is used to grasp a ball. At first the ball is searched, then it is approached and afterward grasped. If one of the steps fails, the robot asks for help.[1]

Nodes can also have a memory, for example, a *Sequence* that only ticks its children until they succeeded one time, can be used. Another important fact is that every child can be a single node or a complete subtree. This increases the modularity of the concept.
An example tree using *Sequences*, *Fallbacks*, *Actions* and *Conditions* is shown in Figure 2.11. It describes the task of grasping a ball with a robot on a high abstraction level. The execution starts on the left side, so at first, it is checked if a ball is found. If not the *Find Ball* action is executed. When the robot can find a ball the next *Fallback* is executed, if not the first *Fallback* fails and because of this the whole *Sequence* fails and the robot asks for help.

2.3.2 Comparison With Other Architectures

Next to the new Behavior Tree framework a lot of other common frameworks exist and were used in different applications. One of the standard approaches, Finite State Machines, is compared to Behavior Trees in the following.

2.3.2.1 Finite State Machines

The most common model to structure complex tasks in computer science are Finite State Machines(FSM). The general structure is very clear because only states, transitions and events are used to describe the task. This is one of the main advantages of FSMs: The are very intuitive and easy to understand[52]. Because of the clear structure, the implementation is also easy. An additional important point is that they were used for years in all parts of computer science, so it is a very well know concept.
But FSMs also have two important problems: One problem is the lack of modularity, which is especially important for larger projects like robotic tasks in this case. It is necessary to create independent modules and reuse them on the next higher organizational level. This is hard to do with FSMs, because they need a lot of transitions between the states. To integrate one module it often is necessary to connect it with different other states, which is unpractical for an increasing number of states.
The second problem is reactivity. The system should react to changes in the environment immediately. This can be reached in a FSM by using a high number of transitions which often ends in a nearly fully connected network. In general, this is no problem, but like told before it makes the modularity harder. So there is a trade-off between the two characteristics reactivity and modularity.

[1] All Behavior Tree Examples are taken from [52].

2.3.2.2 Hierarchical Finite State Machines

To overcome the modularity issue the Hierarchical Finite State Machine(HFSM) was developed. In this concept, superstates were introduced which can have a set of substates and transitions between the substates. This has the advantage that the number of transitions between the superstates can be reduced and that again increases the modularity.
Reactivity is still hard to realize and nevertheless needs a high number of transitions. In addition to the explained disadvantages, a higher number of transitions results in the creation and later editing also becomes harder. A complete comparison of Behavior Trees with FSMs and HFSMs including some examples can be found in [52].
The advantage of using Behavior Trees for robotic tasks is that they were specially developed to fulfill the modularity and reactivity requirements of an autonomous agent. FSMs are a way more general concept and are also used in completely different projects. With Behaviour Trees it is very easy to develop independent modules because they can be simply formulated as a subtree.
The Reactivity is introduced by the usage of different *Control Flow Nodes*. The usage of the *Sequence* and *Fallback* nodes is much more comfortable than the creation of a highly connected FSM. The only big disadvantage of Behavior Trees is the much higher implementation complexity. It is not possible to implement is with little code on your own, instead, it is necessary to use a libraries like [53].

3 Own approach

This chapter presents the developed intuitive Imitation Learning framework. It is structured from the high-level to the low-level: At first the designed task structure is explained, afterward the teaching processes for this task structure are shown. In the third part, the development of a new low-level model-based robot controller is explained.

3.1 Task Representation

The robot should perform different exercises, which can be created by the user. To organize all exercises, a structure is needed that contains all relevant information about the task. It also can be used to create new tasks and saves all information. This structure should be used by inexperienced users, so it needs to be easy to understand. In this project, an own structure is developed which is based on different layers and is introduced in this section.
The new structure should be explained in an example shown in Figure 3.1. In the example exercise the robot should place a glass on a tray. This high-level exercise now is called *Task* and the *Task* layer is the first layer of this structure. *Tasks* are high-level commands, they are on a human communication level. You can compare them to the commands two humans give each other, if they work together, so every user should understand what a task is doing. The example task is used to put a glass on a tray and is called *Put glass on tray*.

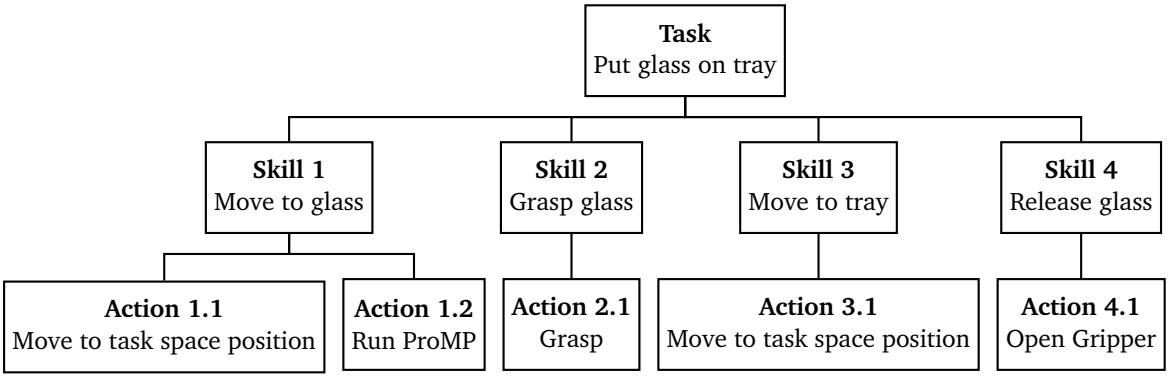


Figure 3.1.: Structure of an example task which puts a glass on a tray. The task consists of four mid-level skills and each skill has some low-level actions. The actions match commands the robot can execute directly. Skill 1 has two actions, because the robot arm should follow a special trajectory while moving to the glass.

In general, every task consists of some different steps which are executed as a sequence. These steps define the second layer of the task representation structure and are called *Skills*. In the example Task the different Skills are *Move Gripper to Glass*, *Grasp the Glass*, *Move the Glass To Tray* and *Release Glass*. The *Skills* should be created by the user so it may distinguish between different users how many Skills they use and what they are doing exactly. The Skill layer is a bit more low level orientated than the task layer, but Skills are still a more human-level description of the behavior. It can be compared to the way a human explains an exercise detailed in a manual.
To create a behavior the robot can execute the third layer is used. This layer is called the *Action*-layer and consists of commands the robot can execute directly. Every *Skill* uses one or more *Actions*. The decision how much *Actions* are used in a *Task* lies with the user. It is possible to define very general Skills with fewer Actions, for example, to open the gripper. But sometimes it also can be useful to define more specific Skills with some more Actions, for example, if a complex movement with some via-points must be executed.

3.1.1 Action Definition

Actions are the lowest level in the used representation structure and define commands that can be executed directly by the robot. The number of needed Actions should be minimized to create an easily understandable system. So every Action has parameters that allow the generalization of the robot behavior and the reuse-ability in different Skills and Tasks.

Table 3.1.: All used Actions

Action Node	Function	Usage
GoToCartAction	Moves the end-effector to a desired position in task space	Move to object
GoToJointAction	Moves the robot to a desired position in joint space	Move to fixed position
RunPrompAction	Executes a ProMP	Execute a movement
GraspAction	Executes a grasp with the gripper	Grasp
MoveGripperAction	Moves the gripper to desired width	-

Action Definition

Actions are the low-level nodes of the task representation structure. They are commands the robot can directly execute and use parameters, so they can be reused for every new exercise. The user cannot create new Actions, he only can define the parameters to adapt their behavior to the new situation.

It is not allowed and necessary for the user to define new Actions, so all Actions are predefined. A detailed overview of all Actions and how they should be used is given in the following, a short summary is given in Table 3.1.
In general, for every basic robot behavior, one Action is needed. The robot behavior can be split into moving the robot arm and using the gripper. To Move the robot arms three kinds of movements are available: The *GoToCartAction*, the *GoToJointAction* and the *RunPrompAction*.

3.1.1.1 GoToJointAction

The *GoToJointAction* is the simplest Action to move the robot, it moves the robot to a defined position in joint space. The Action is a simple GoTo command, so it is not following a special trajectory or supports via-points to precise the movement. It only is necessary to define the desired joint position, the trajectory to reach the goal state is calculated automated.
This Action always moves the robot to the exact same position and no generalization to new situations is provided. Because of this, the action should not be used for object interaction tasks. They require the adaption of the desired position to the object position. To make this clear to the user the *GoToJointAction* is called *Move to fixed position* in the user interface.

Table 3.2.: Parameters of the *GoToJointAction*

Parameter	Type	Value Range	Description
desired_joints	array[7]	-	Desired values for all joints
duration	numeric	-	Duration to reach the desired position
robot arm	enum	left, right	Which robot arm should be moved

3.1.1.2 GoToCartAction

The second type of robot movements is the *GoToCartAction*. In principle, it is similar to the *GoToJointAction*, because it also defines a GoTo command with automated trajectory calculation. The main difference is that this Action moves the robot arm to a desired position in the task space and not in the joint space. The pose in the task space is defined as position and orientation.
The desired pose can be defined in every known frame the robot is using and the transformation to the robot frame is done completely automated. This has the advantage that the pose can also be defined in the object frame, which is equal to the relative position of the gripper to the object. If the relative pose between object and gripper is known, the movement can be easily generalized to new situations, like changing object positions.

Table 3.3.: Parameters of the *GoToCartAction*

Parameter	Type	Value Range	Description
desired pose	pose	-	Desired pose (position, orientation and frame) for the gripper
duration	numeric	-	Duration to reach the desired position
robot arm	enum	left, right	Which robot arm should be moved

This is one of the most important requirements on object interaction and pick and place tasks and also the majority of exercises in our project are connected to objects. To make this understandable for the user this action is simply called *Move to object*. So the user can decide if he wants to move the robot to a fixed position, which means the usage of a joints-space command, or if he wants to interact with an object and use a task-space command. This decision can be made without any knowledge about joint and task space in general and simplifies the usage of different movement types a lot.
One point on this partition is that it is not possible to perform task-space movements in the base frame of the robot, but this is not required if you think more about it.

3.1.1.3 RunPrompAction

The two previous movement Actions are only GoTo commands, so they cannot execute a special trajectory. For some more complex tasks, it can be necessary to follow a defined way, for example, to avoid obstacles or moving to the grasping position from a defined direction. This often is easier when the whole trajectory can be defined instead of single positions. Although the generalization of the defined trajectories to new situations must be possible. That is especially important if objects at different positions should be manipulated like in this project. To fulfill these requirements the ProMP framework, which is explained in Section 2.2, can be used. ProMPs can be trained from demonstrations and allow the creation of trajectories that can be generalized to new end-positions. The training of new ProMPs and conditioning to new situations are described in Section 3.2.3, this part only explains the execution.
Every ProMP is recorded in both task- and joint space and also can be executed in both control modes. In general, it is required to reach a defined position in the task space, so the execution in task space is the default mode. Each ProMP also can be connected to an object, because they are recorded in the object frame. This allows a completely automated conditioning process.
With ProMPs it is also possible to move both robot arms at the same time that is not possible with the two different GoTo commands. So ProMPs also should be used for bimanual tasks that need synchronized movements of both robot arms. To make the whole concept of following a trajectory understandable for the user, the Action is called *Execute Movement* in the interface.

Table 3.4.: Parameters of the *RunPrompAction*

Parameter	Type	Value Range	Description
promp name	string	-	Name of the ProMP
control mode	enum	cart, joint	Control mode for execution
duration	numeric	-	Duration of the trajectory
robot arm	enum	left, right, both	Which robot arm should be moved

3.1.1.4 GraspAction

To use the gripper the robot can execute two Actions: The *GraspAction* and the *MoveGripperAction*. The *GraspAction* performs a grasp with a defined grasping force. This can be used for grasping objects with not exactly known shapes and holding them afterward. The robot closes the grippers until the specified force is applied to the object between the gripper. This procedure makes the grasping process much easier because the grasping and fixation afterward are automated. The most important parameter of this Action is the grasping force because it distinguishes between different hard and heavy objects.

The *GraspAction* can also evaluate if the performed grasp was successful. To do this, an interval for the grasping width (gripper width ± gripper epsilon) can be defined in which the object is expected. If the gripper cannot apply the desired force in the specified gripper width range, the grasp fails. This allows the detection if no object is grasped.

Table 3.5.: Parameters of the *GraspAction*

Parameter	Type	Value Range	Description
grasping force	numeric	0 - 70	Desired applied force to the object in N
gripper width	numeric	0.0 - 0.08	Expected width of the gripper in m
gripper epsilon	numeric	0.0 - 0.04	Allowed range for the gripper width in m
gripper speed	numeric	0.0 - 0.05	Value how fast the gripper should move
robot arm	enum	left, right	Which gripper should be used

3.1.1.5 MoveGripperAction

The *MoveGripperAction* is very simple and only moves the gripper to a defined opening width. The used parameters are described in Table 3.6. It is not useful for grasping exercises, because in general it is not exactly known how big the grasped object is. So this action only is needed to open the Gripper after it grasped an object. To avoid confusion a Skill is predefined to open the two grippers and the *MoveGripperAction* is not available for creating new skills.

Table 3.6.: Parameters of the *MoveGripperAction*

Parameter	Type	Value Range	Description
gripper width	numeric	0.00 - 0.08	Width of the gripper in m
gripper speed	numeric	0.00 - 0.05	Value how fast the gripper should move
robot arm	enum	left, right	Which gripper should be used

3.1.2 Skill Definition

Actions are only very general commands the robot can execute. To define the desired behavior of the robot in a specific task the user has to create new Skills. Skills are the mid-layer of the task representation structure. One Skill consists of one or more Actions that are executed in a sequence.
The Skill should define all parameters of the Actions it contains, so it can be said that a Skill is a sequence of parametrized Actions. In general, it can be useful to define Skills with some parameters, for example, if the same skill can be used on the left or the right arm. In the definition of the structure and the used implementation, this is allowed, but for the first version of the system, this functionality is blocked, because it is not clear how successful the users use the Skill and Action structure. Understanding the structure is much easier if a Skill has no parameters and defines *all* parameters of its Actions. The only disadvantage of not using Skills with parameters is that a few more Skills are needed. This is not a problem in the beginning, because the total number of skills will be still small enough.

Skill Definition

Skills contain a sequence of one or more Actions which are executed after each other. They define all parameters of the belonging Actions. Skills should define a small step of the robot behavior which is big enough to be a human-understandable step. The user should create new Skills on runtime without coding and can reuse them in different Tasks.

A Skill should describe a small step of a task and it should be possible for the user to understand intuitively what a Skill is doing. Example for Skills are *Grasp object_A* or *Move object_A* while *object_A* is a specific object, for example a glass.

Skills should be defined specifically for every object and not for a group of objects. This reduces the reuse-ability of skills a bit but makes the training process more simple. The user only has to focus on the one used object and has not to think about generalization. For the example of grasping object_A, the user can define a special grasping position and set the grasp action parameters. But at every time the robot should manipulate the same object again in a different task, of course, the skill should be reused to reduce the teaching effort.
Another important point is that the user should create new Skills on runtime and without any coding to make the teaching process intuitive. However Actions are not created on runtime, so at this point, less flexibility is needed. This must be respected while implementing the whole structure.

3.1.3 Task Definition

A Task is the highest level in the task representation structure. It describes a human-level command the robot should execute. The used structure of a Task is simple and clear to avoid confusion at the user. So a Task only is a simple sequence of Skills. At this stage of the project, no conditions and other extensions are provided and needed for the most pick and place task which should be executed. Before introducing more complex control architectures at first the reaction of users to the framework should be tested and discussed. Afterward, the addition of some more complex control flow functions to the Task structure can be considered.

Task Definition

A Task is a high-level command the robot should execute and the human user can intuitively understand. It contains a sequence of Skills and executes them one after each other.

3.1.4 Structure Robot Behavior as Behavior Trees

The tree structure explained in the last sections is implemented using Behavior Trees. This has the advantage that it is very easy to connect different actions and they also integrate the error handling if one Action fails. This makes the usage very comfortable.
Behavior Trees support a lot of different functions like described in Section 2.3, but for this application, only some basic nodes are used to keep the structure simple.
The main goal is to generate an easy to use structure that allows the generation of new tasks and skills without programming knowledge during the runtime of the program. To reach this only the absolute basic functions are coded as Actions nodes. Action nodes are coded in C++ and were compiled, so they cannot be adapted during runtime and it is not possible to add new Actions for the user. A complete list of all action nodes is shown in Table 3.1.
Because the Actions are very general, they need to be specified for every task they should do. This is done in the *Skill*. A Skill is not coded in C++, instead, the high-level Behavior Tree structure is used and every Skill is an own Behavior Tree. To keep the structure clean, the skill-tree contains only one *Sequence* with any number of actions. This structure is enough to reach all tasks in this stage of the project and it easy to create it automated by software tools.

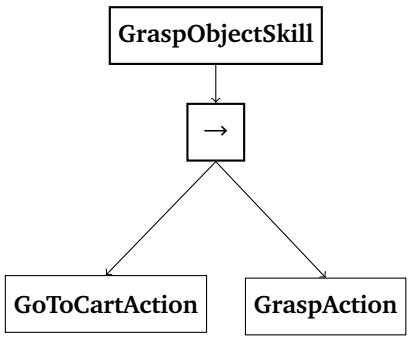


Figure 3.2.: Example of a grasping Skill as Behavior Tree. The Actions are basic functions to control the robot. The information where the robot should move and how it should grasp the object is stored in the Skill.

A good example is a grasping skill shown in Figure 3.2: At first it is necessary to move the gripper to the desired grasping position and as second to grasp the object. So the *GoToCartAction* and the *GraspAction* are used. The information where the GoTo command should move the robot and how the Gripper should grasp the object is stored in the Skill-Tree. Because these parameters are different for all objects it is necessary to create new skills for every object.
Every Skill describes only one small step, to complete a job different skills are needed in combination. This is done by the *Task* which is similar to the Skill, but instead of Actions, it has Skills as children. So it also is a Behavior Tree with a single *Sequence*, but every child is a subtree created from the different Skill-Trees.
In Figure 3.3 a example object moving task is described. It uses the GraspObjectSkill described before, then it moves the object to the desired position stored in the MoveObjectSkill. After that, the object is released with the OpenGripperSkill and the robot arm is moved back to a fixed home position.

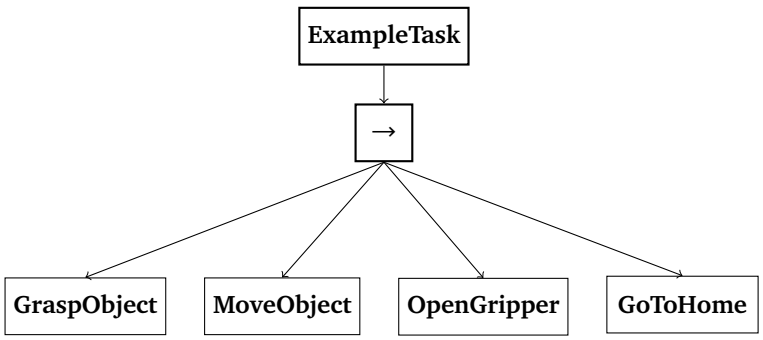


Figure 3.3.: Example of a Task described as Behavior Tree. The robot should move an object from one position to another, so a GraspObject- and MoveObject-Skill are needed. Then the gripper is opened to release the object and the robot goes back to the home position.

3.2 Teaching Process

The teaching process is one of the main aspects of this project. The aim is an intuitive process that can be used to create completely new tasks. Intuitive means in this case that the system is structured simple enough that users without background knowledge can use it successfully. From this start-point some key objectives about the teaching process can be formulated:

- Create new tasks without coding
- Train the robot with Imitation Learning
- Use an intuitive user-interface

The first and most important point is that it must be possible to create completely new tasks without coding. It can be assumed that the target group has no knowledge in computer science or programming, so coding would be a completely new concept for them. Instead, the robot should be trained using Imitation Learning and an intuitive user-interface. Of course, the user does not know what Imitation Learning is and how it is working. Because of this, the Imitation Learning process must be created very simply and the user must be guided through the process. Every technical and mathematical part of the Imitation Learning process must be done automated, to make the usage as easy as possible for the user.
To describe new tasks the structure explained in Section 3.1 is used. So it is necessary to train the complete task, which is described in Section 3.2.1, and also new skills, which is described in Section 3.2.2. Besides, the user can train new ProMPs to execute trajectories. Because the training of ProMPs is a bit more complicated, it is described in the extra Section 3.2.3.

3.2.1 Task Teaching

The creation of a new task is the highest level of teaching in this project. At first, it is assumed that all needed skills are available, so at this point, no skill teaching is needed. Starting from this, the teaching process is very simple and can be compared to a human instructor who explains something using a manual. The user only has to tell the robot the steps it should execute and the robot should execute them as a sequence.
This process is shown in Figure 3.4: At first the user selects a skill he wants to add to the task. Because the skill behavior can be hard to imagine, the robot can execute the skill directly and the user can see if the robot behavior is equal to his desired behavior. After the successful execution of the skill, the user can add the skill to the task. Because the user only has to select skills and not to set any parameters or doing other training, this process is simple and should be understandable for every user.
But for most tasks, it will be necessary to train some new skills, even if the user can reuse some other skills that were trained earlier. This makes the process more difficult for the user because he has to think about the skills he needs. An experienced user would preliminary plan how he wants to create a task, what skills he needs and afterward train them. He also will focus on the reusability of the skill and will adapt his planning on this aspect. This is, in general, the best approach, but it is not intuitive at all and would be confusing for inexperienced users. Because of this, the preliminary planning is avoided completely in this teaching structure.
To do this the task teaching process directly includes the teaching of new skills, as shown in Figure 3.5. At the point where the user selects the existing skills he also can choose to create a new skill. Afterward, he is guided through the training process of the skill, which is explained in Section 3.2.2, can execute it if it is necessary and add it to the task. This teaching process is much smoother and more interactive than the planning process, an experienced user would use. The user only has to think about the next step the robot should do, he does not need to think about steps more in the future.
One disadvantage could be that the reusability of the created skills is low because the users define very long skills. A long skill with a lot of actions performs a very specific behavior, so it hardly can be reused. This does not affect the programming of one task but increases the effort for designing multiple tasks using similar behaviors. Another possible problem could be that the designed skill all have only one action. This increases the reusability of the skills but also yields a very abstract skill that is far away from the human-understandable level a skill should have. But it is hard to predict how inexperienced users would use the framework, so this must be examined in a user-study.

3.2.2 Skill Teaching

Skills define the specific behavior of the robot, so teaching new skills is the key feature of the teaching process. The teaching of new skills should be done as an interactive process, where the user is strictly guided. This reduces the

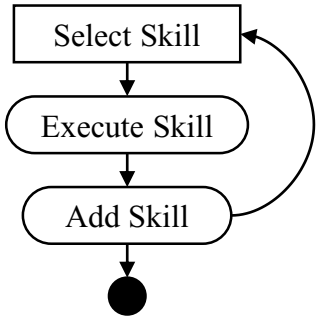


Figure 3.4.: The Task Teaching Process can add excisting Skills to a Task.

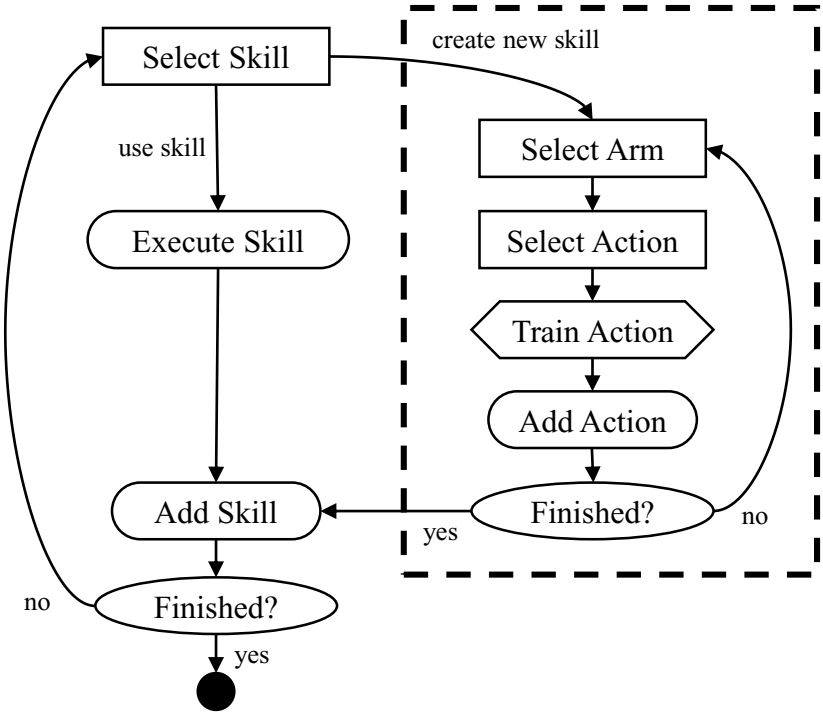


Figure 3.5.: The Extended Task Teaching Process combines the creation of the Task itself and the creation of new Skills. The Skills and used Actions can be trained directly.

amount of needed background knowledge and avoids user errors. The user studies described in Section 2.1.2.2 suggest that kinesthetic teaching is the most promising approach and is also accepted by the user. Because of this kinesthetic teaching besides of other interactive methods should be used for the robot training.

In the beginning teaching a skill is similar to teaching a task: The user now selects an action the robot should execute. The difference at this point is that the action has parameters that must be defined. To define these parameters for every action an own teaching process is needed. They are described in detail in the following.

3.2.2.1 GoToCartAction

The *GoToCartAction* is used for moving the robot gripper to a position relative to an object. At first, the user selects the robot arm that should be used, afterward he selects an object the robot should interact with. If this is done the desired position can be taught by kinesthetic teaching. The robot arm switches automated to the zero gravity mode so the user can move it without much force. When the user has moved the arm to the desired position he can confirm the pose by pressing a button.

This process is simple for the user, but using the object-relative position requires some background calculations. The robot is using different frames for the description of object positions and the execution of GoTo commands. The most important frames are shown in Figure 3.6. The *base_link* frame is the mainframe of the robot and has an orientation that is intuitive for human operators.

The robot arm task space controller cannot directly execute commands in the *base_link* frame, instead every arm uses an own frame, called *link0* frame. They are placed directly under the first joint of the robot arm. The transformation between these frames is constant over time because it only describes the body structure of the robot, which makes the transformation simple.

Using positions relative to objects makes the process a bit more complicated because the position of objects can and will change over time, so the transformation is not constant. To define the object frame the exact pose, consisting of position and orientation, of the object is needed, which can be observed by object tracking techniques. The object tracking is not directly part of this project, so it is assumed at this point that the position of all objects is known exactly.

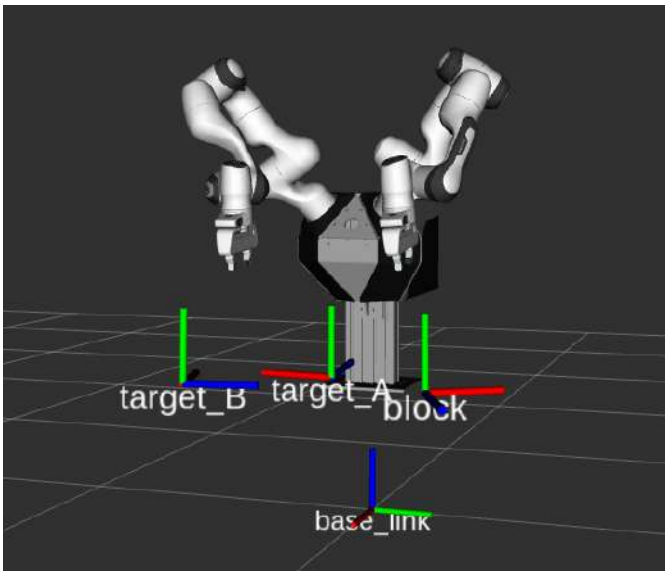


Figure 3.6.: The robot is using different frames to describe the position of objects and execute commands. The main frame is the *base_link* frame placed directly under the robot. The two robot arms use the *_link0* frames for executing commands. In addition every object has an own frame, so movements relative to objects can be recorded.

If the pose of an object is known the transformation to any other frame can be done and it is possible to move the robot to a position described relative to the object. This can be used to generalize the *GoToCartAction* to new situations, especially new object positions. In theory, the robot can reach the desired position in the object frame for every task-space position of the object and needs only one taught example position. Problems only can occur, if the robot cannot reach the position because of joint limits.
Another point while teaching GoTo Actions is that the robot will directly move to the desired position and is not following a specific way. This can generate problems if the robot should be moved to a grasping position because it must be avoided that the gripper touches and moves the object while moving to the desired position. This can be solved by defining a pre-grasp position that can be reached safely from every state and is only slightly away from the grasp position.

3.2.2.2 GoToJointAction

The *GoToJointAction* teaching process is very similar to the teaching of task space GoTos, but it is not connected to objects. So the user only selects the robot arm and uses kinesthetic teaching to define the desired joint positions. Because the joint controller of the robot can directly execute them, no transformations are needed and the GoTo in joint space can always be executed directly.

3.2.2.3 RunPrompAction

Using ProMPs is one of the key features of this framework because it allows the user to train a trajectory the robot follows and generalize it to new situations. To use ProMPs, two steps are required: The first is the training of a new ProMP and the second is the execution of the ProMP. At this point, it is assumed that the ProMP is trained and only the execution is required. The creation of a ProMP training process that is intuitive enough for unexperienced users is described in 3.2.3. Executing a ProMP requires the generation of a trajectory using the ProMP parameters. This trajectory should be adapted to the actual situation, in case of object interaction task, this is the position of the object. To adapt the ProMP in general conditioning can be used. In this project ProMPs are used mainly for object interaction tasks, so a special formulation is used to make the adaption to new object positions easier: Instead of recording the ProMP in on of the robot frames, like it is done in the most project, the demonstrations are recorded in the object frame. This is very similar to defining a GoTo command in the object frame, but now is handling complete trajectories.
In Figure 3.7 an example ProMP is shown in both, base frame and object frame. The ProMP describes the movement to grasp an object and was trained with different object positions, while the start position of the trajectory always was the same. There can be seen some important differences between the two representations: The orientation of the frames is not the same, so it is not possible to directly compare the three axes to each other. For example, the y-axis in the base

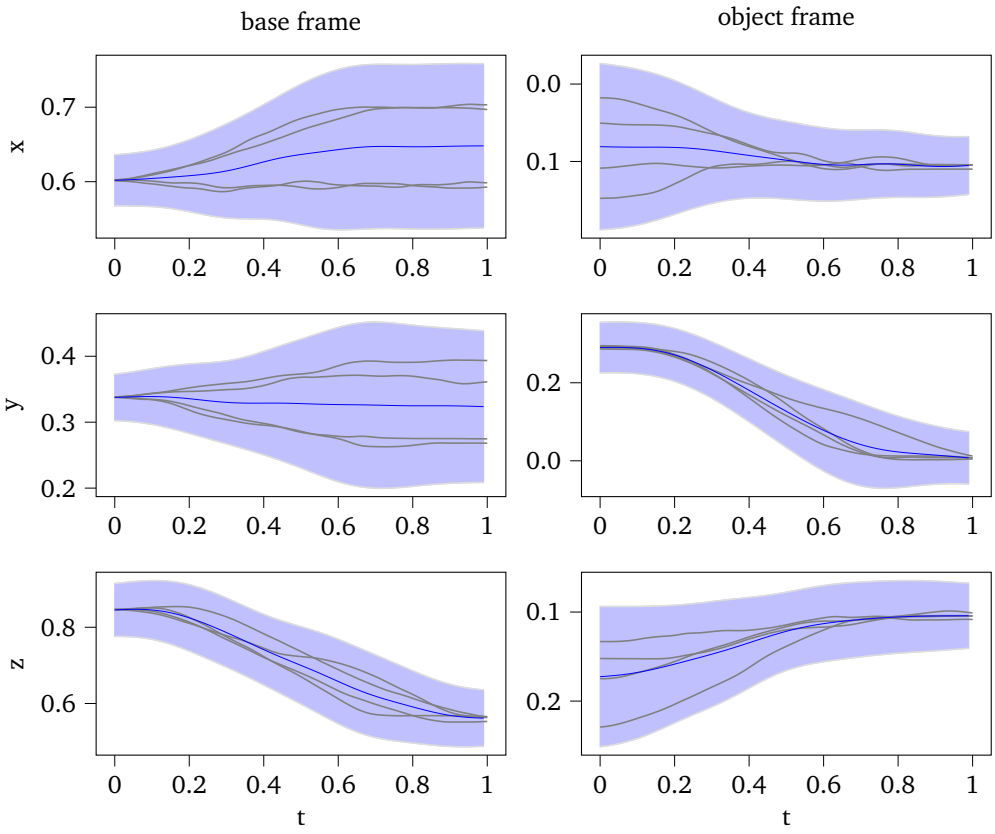


Figure 3.7.: The left side of the figure shows the position part of a ProMP in the base frame of the robot, the right part shows the same ProMP in the object frame. Grey lines are the demonstrations, the blue line is the mean with the 2σ-surrounding. It can be seen that the base frame ProMP has different end-positions and the object frame ProMP has different start positions.

Figure 3.8.: The Panda robot uses a simple two-finger gripper. It can apply a desired force on the grasped object and holds it constant to fixate the object.

frame describes the height, which is similar to the y-axis in the object frame. But this only is a problem of visualization and does not influence the usage of the ProMP framework.

The second point is that the start position of the trajectory in the base frame is, like expected, the same. The end positions differ a lot because the object was placed at four different positions. If the trajectory is recorded in the object frame, this is completely contrary, so the start positions differ and the end positions are equal. This makes sense because in object frame the grasp position has always the same pose.

The consequence of this change in the representation is that conditioning must be done on the start position, even if the final transformed trajectory has a new end position. This is confusing at first, but conditioning on the start position is simple because the start position always is known at the time where the calculation must be done. The conditioned ProMP will still has the desired end position because the variance at this point is small, so the result will be near to the mean.

Of course, the conditioning can also be done in the base frame representation and can create similar resulting trajectories, but the representation in the object frame has another advantage: The different demonstrations are more similar to each other, so they can be represented better with mean and variance. This should lead to a ProMP with better generalization capabilities.

All in all the training of a *RunPrompAction* is simple for the user: He only selects the ProMP, which should be executed and selects the arm which should be used. In principle, the ProMP could be executed with the joint or task-space controller, but conditioning is easier in the task space. So the execution also is done by default in task space. Some more advanced methods [54] also allow the conditioning in task space and the execution in joint space, but at this point, it is enough to use the task space controller to track the trajectory.

3.2.2.4 GraspAction

For using the *GraspAction* especially the gripper width parameter is needed. It defines at witch width the object is expected and is used to evaluate if the grasping was successful. The grasping force and grasping speed were set to default values to simplify the teaching process. The disadvantage using default values is that the gripper can only grasp objects with a similar surface and stiffness. In future versions of this framework it will be necessary to set the other gripper parameters automated, but grasping itself is a very complicated topic, so this would increase the effort too much at this point in the project.

Because the grasp cannot be taught by kinesthetic teaching, the *GraspAction* needs a completely new teaching process. It basically uses a test grasp: The user is asked if the object is in the right position and afterward the robot performs a test grasp. If it was successful, it can read the actual opening width of the gripper and use it for the taught Action. In Figure 3.8 this process is shown.

Training the *MoveGripperAction* is not necessary for the user and only needs to set the desired width, so it is not explained further at this point.

3.2.3 ProMP Teaching

In this project, it should be possible for the user to train trajectories the robot can execute. These trajectories must be generalizable to new end positions, for example, if the arm should move to an object and the object is placed at different positions. This can be done using ProMPs, a framework for generating Movement Primitives described in Section 2.2. The ProMP framework is based on Imitation Learning, so it learns from given demonstrations. The needed demonstrations must be generated by the user, but this part of the ProMP usage is not treated in the most ProMP applications. In general, they focus on the training and execution phases and assume that the demonstrations are simply available.
In this project, the demonstrations must be created by inexperienced users, so it must be created a concept that guides the user through the process. In former projects, the demonstration generation is done completely manual and no special teaching system was used. In the following a ProMP that is only using one arm is called *Simple ProMP* and the training process is described in Section 3.2.3.1.
Another important point is that the robot in this project has two arms and should perform bimanual actions. Executing bimanual GoTo commands has no advantage at all because the movements of the two arms are not synchronized. This means that every arm reaches his goal position but the ways they use are independent of each other. However, for some tasks, it is required to execute movements with synchronized ways on both arms. Like said before this only makes sense if the whole trajectory is trained and because of this it is necessary to train and execute ProMPs for both arms together. Training both arms together is more complicated for the user, so a special teaching process is developed for ProMP that should execute a movement that is coordinated between the two arms. These ProMPs are called *Coordinated ProMPs* in the following and the training is described in Section 3.2.3.2.
Before focusing on the teaching process, the general structure of a ProMP is explained in the following. To create ProMPs that can describe both robot arms a similar structure to [39], where one ProMP describes the behavior of a controlled and observed agent together, is used. In this project one ProMP describes the trajectory of the left and the right arm:

$$\boldsymbol{y}(t) = \begin{bmatrix} \boldsymbol{q}(t)_R^T & \boldsymbol{q}(t)_L^T \end{bmatrix}^T \tag{3.1}$$

$\boldsymbol{q}(t)$, in this case, can describe the trajectory in the joint- or task-space or both. Using both is useful in this research case because it is possible to select the representation you want later. The only disadvantage is the doubled calculation effort, but this is no problem on the used hardware. $\boldsymbol{q}(t)$ when has the following structure with the robot arm joints $q_1, ..., q_7$ and the end-effector pose described as position and orientation as quaternion:

$$\boldsymbol{q} = \left[q_1, ..., q_7, x, y, z, q_w, q_x, q_y, q_z\right] \tag{3.2}$$

Using both arms in one ProMP only leads to some changes in the structure of the weight vector, all the underlying calculations of the ProMP framework are not changed at all. The weight vector for two arms with P degrees of freedom is structured like this:

$$\boldsymbol{w} = \left\{\left[\boldsymbol{w}_1^T, ...\boldsymbol{w}_p^T, ...\boldsymbol{w}_P^T\right]^L, \left[\boldsymbol{w}_1^T, ...\boldsymbol{w}_p^T, ...\boldsymbol{w}_P^T\right]^R\right\} \tag{3.3}$$

It can be seen that a ProMP for both arms has twice the dimensionality of one arm. To use one unique structure both ProMPs for one and both arms use the two armed ProMP structure. This also increases the calculation effort, but the underlying ProMP structure is more clear. When the ProMP is executed it is necessary to select the desired DOFs, in general, the task space position and orientation of one or both arms will be used.
Some automated preprocessing steps are needed to use the demonstrations. At first, the active area of the demonstration is selected by cutting these parts at the start and end of the recorded trajectory where all trajectories are constant. The second step is the normalization of the time base, so all trajectories start at 0 and end at 1. This avoids problems with different execution speeds between the demonstrations.

3.2.3.1 Simple ProMP

A *Simple ProMP* is a ProMP that only is executed on one robot arm, so it also is only required to train one robot arm. The training process follows a strictly defined process which ensures that the user only has to move the robot arm, all other steps are automated.
The used process is shown in Figure 3.9: Like explained in Section 3.2.2.3 the ProMPs are connected to objects. So in the first step, the user selects the object that the robot should manipulate.
The second step is the recording of the demonstration. Before the recording is started the user is asked to place the object at the desired position and to confirm the start of the recording process. Afterward, the used arm is switched in

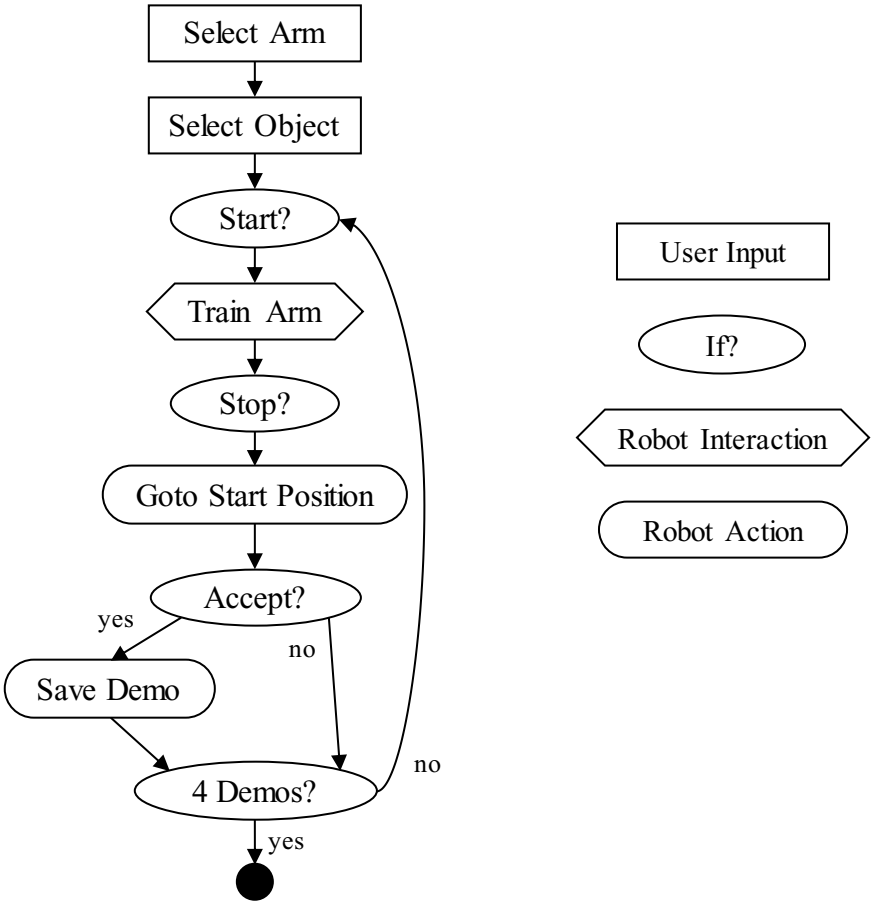


Figure 3.9.: The ProMP teaching process is used to train a Simple one-handed ProMP. At first, the user selects an object, then he gives four demonstrations. Between them, the robot is moved automatically back to the start position. The user also can discard the demonstration and create a new one.
One the right side the used symbols are explained.

the zero-g mode, so the user can easily move it and give the demonstration with kinesthetic teaching. The user can stop the recording by pressing a button. Then the robot will move back to the start position, so the user can directly give the next demonstration.

In some cases, it can happen that the user is not satisfied with the demonstrations, for example, if the movement was not smooth or the robot was moved in a joint limit. To handle this case, the user is asked if he wants to use the demonstration before it is saved.

One parameter that must be normally adapted by the operator is the number of given demonstrations. In general, it depends on the complexity of the movement and the area in which the ProMP should be possible to generate generalized movements. A ProMP of course only can generalize to new situations, in this case, new endpoints that are similar to the trained ones. If the desired area of generalization is big it is better to give more demonstrations, if the area is small a small number is enough.

For an inexperienced user, it is hard to understand this problem and he cannot assess how much demonstrations are needed. Because of this, the number of demonstrations is fixed to four in this teaching process. Four demonstrations are in most cases enough to teach a rectangle area where the generalization is successful. For very simple tasks with fewer generalization requirements four demonstrations are too much, but in that case, giving one demonstration needs not much time. All in all four demonstrations should be a good compromise between a good generalization and a fast teaching process, but this point also can be evaluated in the user study. To ensure that the ProMP can still generalize to a new position the user is requested to place the object at a new position after every iteration of the recording.

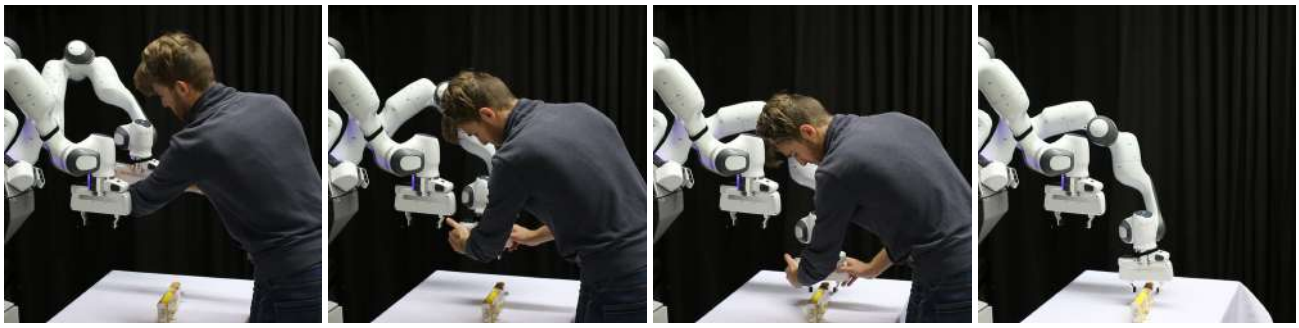


Figure 3.10.: One user is training a Simple ProMP using the teaching process in Figure 3.9. He only has to move the robot gripper in kinesthetic teaching mode on the desired trajectory to give a demonstration. This must be repeated four times.

3.2.3.2 Coordinated ProMP

The Coordinated ProMP represents a bimanual movement witch synchronization of the left and right arm. Teaching both arms at the same time is very hard for one single user because he must move both arms at the same time. If the user is trained in this it is possible, but the results are often not satisfying. One possible solution is that two users train the robot together. This makes it much easier, but in this project, the training should be done by one single person.
To make this possible a special teaching process is used, which is shown in Figure 3.11. The main idea is that the user teaches the two arms after each other. If this should be done the synchronization between the two arms is still hard. If the demonstrations for both arms have different lengths and speeds the synchronization cannot be done automated.
Instead of recording the demonstrations for both arms completely independent, the user at first gives the demonstration for one arm and this demonstration is replayed in the second step by the robot. During the replay, the user teaches the second arm of the robot and can directly adapt its movement to the first arm. During this, the movement of both arms is recorded and used as a demonstration for both arms. This process avoids the problem of synchronizing two independent recordings while using an interactive training process.
The rest of the training process is the same as in Simple ProMP training. So the user can still select an object and has to give four demonstrations. He also can select if the demonstration should be used or if he wants to create a new one.

3.2.3.3 Simple ProMP With Both Arms

The *Coordinated ProMP* can handle the most use-cases for the bimanual task, but not all: Sometimes it is not possible to move each arm independent from each other. This is especially the case if the two arms are connected, for example, because they have grasped the same object and should lift it. The advantage of two connected arms is that teaching them together is much easier when teaching two independent arms at the same time.
For this special use-case, the teaching process, shown in Figure 3.13, can be used. It is similar to the *Simple ProMP* teaching process but is using two arms. The only disadvantage is that moving two arms connected with a grasped object automated back to the start position is more difficult and so the user has to move them back on his own in the second step. To do this the robot is switched in zero-gravity mode again.

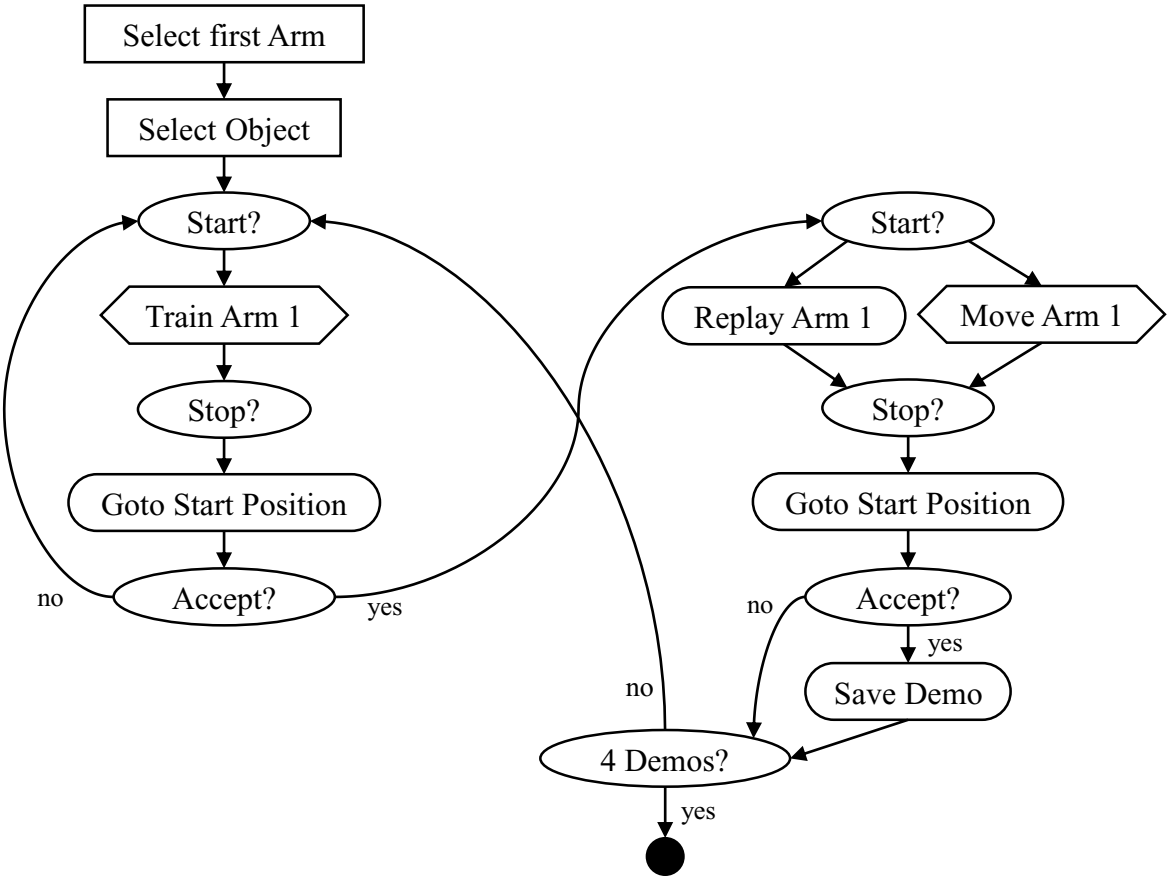


Figure 3.11.: The Coordinated ProMP Teaching Process is used to train coordinated bimanual tasks. The two arms can be trained successively. After the first arm is trained the movement is replayed and the second arm can be synchronized to the first one.

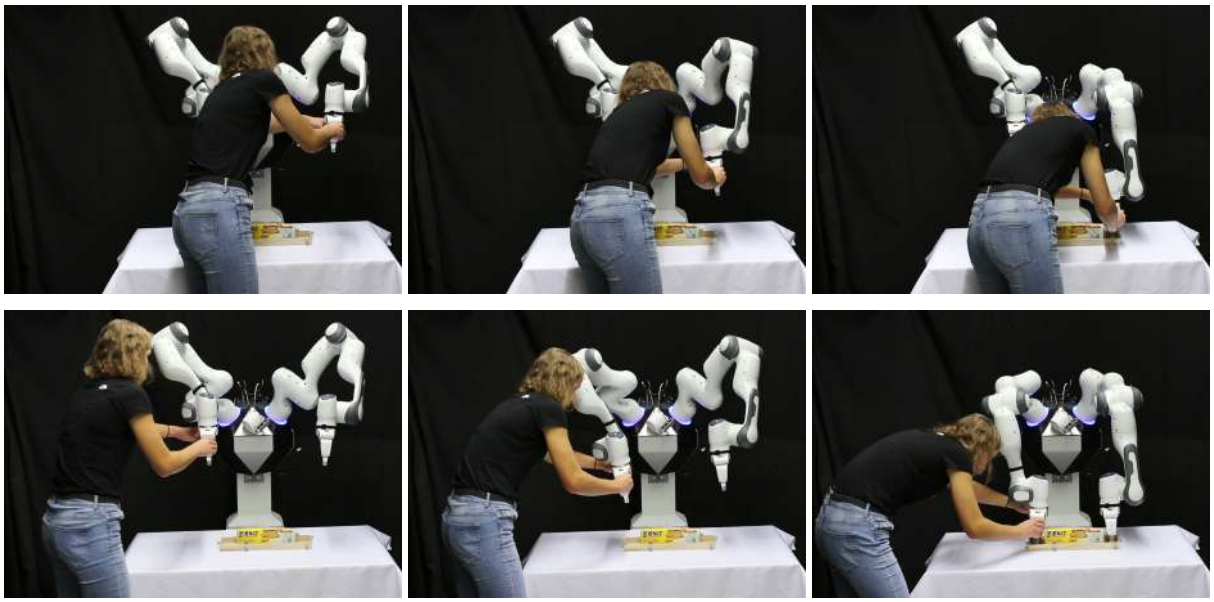


Figure 3.12.: An user is training a coordinated ProMP. At first, the left robot arm is trained, afterward it is replayed and the right arm is trained. The user can easily synchronize the movement of both arms.

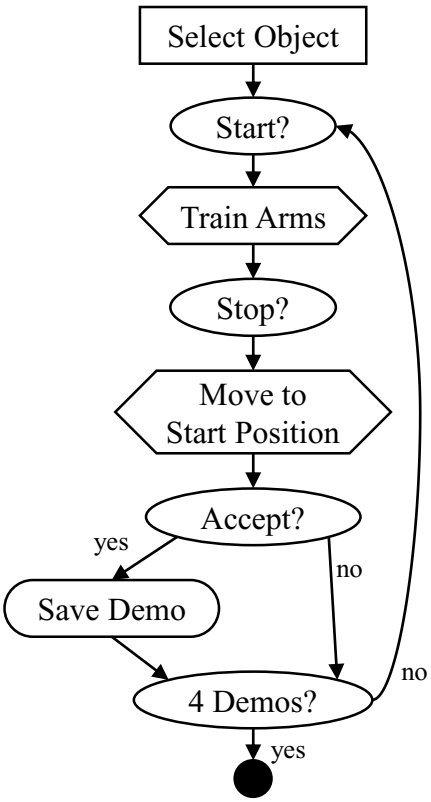


Figure 3.13.: Simple ProMP with Both Arms Teaching Process: This process is used to train both arms at the same time ad is recommended, it both arms has grasped the same object. In this process the user has to move the robot manually in the start position.

Figure 3.14.: An users teaching both arms at the same time with the Teaching Process shown in Figure 3.13. This is recommended, if the robot has grasped an object with both grippers like shown in this example.

3.3 Control Modes

The robot controllers are the lowest level we can access the robot because they directly control the torque of the joint motors. This is still one level under the actions in the tree structure from Chapter 3.
The robot needs different controllers for the teaching and execution mode. To follow a desired trajectory or reach a position joint- and task-space controllers are needed. To use the kinesthetic teaching a gravity compensation mode is necessary. The gravity compensation mode is supported by default by the Panda robot and Franka Emika also developed a good joint controller. Both can be used in this project, so no own development is needed. The Franka task-space controller has the disadvantage that it uses some null-space control methods but they are unknown and the desired null-space position cannot be submitted. Additionally, the controller is very stiff, so the development of an own task-space controller is necessary. Further, an assisted teaching controller should be developed to assist the user in the kinesthetic teaching mode.
Both tasks, the task space controller with null-space control and the assisted teaching controller can be reached with the same controller structure if the controller decouples the task-space and null-space behavior like Figure 3.15 shows.

3.3.1 Model Description

To reach this goal, a model-based controller is used, based on the standard model of an n-dimensional manipulator, which can be written as

$$\boldsymbol{M}(q)\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \tau_{ext} = \boldsymbol{\tau} \tag{3.4}$$

with the $(n \times 1)$ joint angle vector q. $\boldsymbol{M}(\boldsymbol{q})$ describes the $(n \times n)$ inertia matrix, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ the $(n \times 1)$ vector containing the centrifugal and Coriolis effects and $\boldsymbol{g}(\boldsymbol{q})$ represent the gravity effects. The model inputs are the motor-torque $\boldsymbol{\tau}$ and the external torques τ_{ext}, which appear if the robot interacts with the environment. The terms $\boldsymbol{M}(\boldsymbol{q})$, $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ and $\boldsymbol{g}(\boldsymbol{q})$ are nonlinear dependent on the joint angles and velocities, so the manipulator model is a non-linear model.

3.3.2 Task-Space Impedance Control

The knowledge of the manipulator model can be used to linearize the system using the control law

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}_c + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \tau_{ext} \tag{3.5}$$

with the new control input $\boldsymbol{q}_c$. This control law assumes that the external forces are known. Because this approach uses the knowledge of the inverse of the dynamics this controller type is also called *inverse dynamics control*. The task-space tracking control can be reached by a model based PD controller like the *Operational Space Inverse Dynamics Controller* explained in [55]:

$$\boldsymbol{q}_c = \boldsymbol{J}^{-1}(\boldsymbol{q})\left(\ddot{\boldsymbol{x}}_c - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}\right) \tag{3.6}$$

$$\ddot{\boldsymbol{x}}_c = \ddot{\boldsymbol{x}}_d + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{x}}} + \boldsymbol{K}_P\tilde{\boldsymbol{x}} \tag{3.7}$$

To get a well arranged equation $\tilde{\boldsymbol{x}} = \boldsymbol{x}_d - \boldsymbol{x}$ describes the error from the desired position $\boldsymbol{x}_d$, while $\boldsymbol{K}_P$ is the desired stiffness and $\boldsymbol{K}_D$ is the desired damping. $\boldsymbol{J}$ is the Jacobian matrix describing the relation:

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{3.8}$$

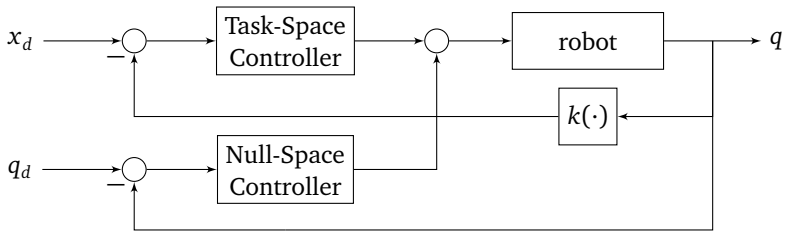


Figure 3.15.: Overview of the task- and null-space controller structure. The desired task- and null-space position can be submitted independently and use an own controller block.

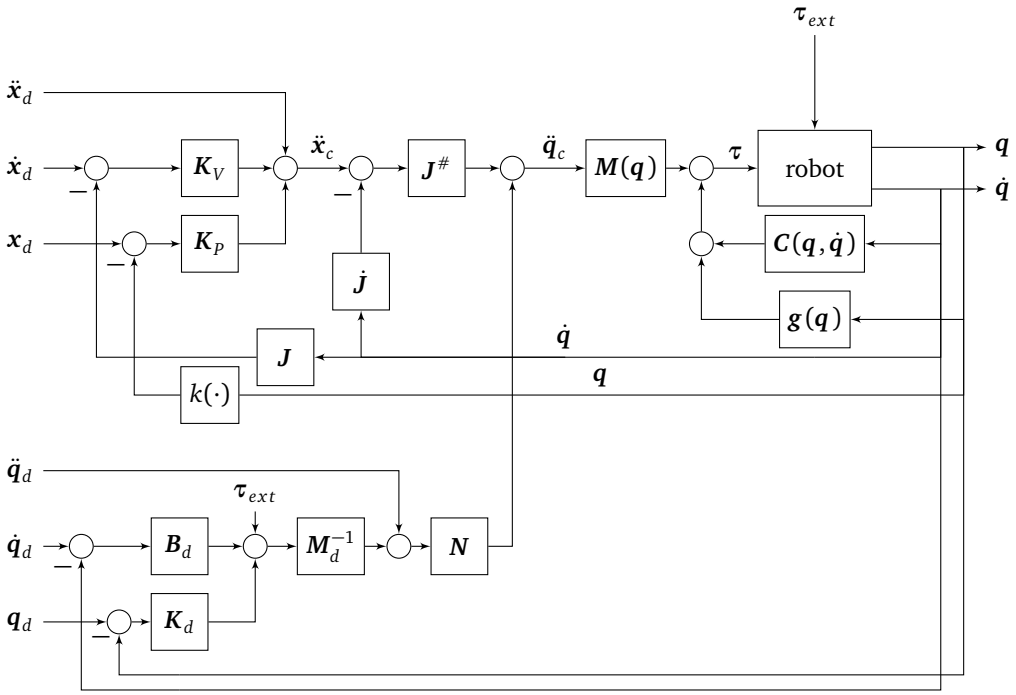


Figure 3.16.: The overall controller scheme can be split in the task- and null-space part. The upper block controls the task-space position of the end-effector, while the lower part controls the null-space position in joint space using the null-space projection N.

For a 7-DOF manipulator and a six-dimensional task space consisting of three translational and three rotational degrees of freedom the Jacobian is a (6×7) matrix, so it is not square and not invertible. To avoid this problem, the generalized inverse $\boldsymbol{J}^{\dagger}$ can be used which fulfills the condition $\boldsymbol{I} = \boldsymbol{J}\boldsymbol{J}^{\dagger}$. A common approach for $\boldsymbol{J}^{\dagger}$ is:

$$\boldsymbol{J}^{\dagger} = \boldsymbol{J}^{T}(\boldsymbol{J}\boldsymbol{J}^{T})^{-1} \tag{3.9}$$

The manipulator model in equation (3.4) does not include friction in the different robot joints. In general, this should not generate problems, because the robot has a low-level integrated friction compensation. However some own experiments with the robot show that the friction in the last joint that rotates the end-effector, is very high and not completely compensated.
Because of the resulting big model errors, the model-based controller can only hardly move the last joints, which results in big orientation errors. The end-effector position is controlled much better nevertheless because the last rotational joint is not needed for position control.
To avoid this problem, the model based task-space impedance controller is expanded with an standard *PD operational space controller*[55]:

$$\boldsymbol{\tau} = \boldsymbol{J}^{T}(\boldsymbol{K}_P\tilde{\boldsymbol{x}} + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{x}}}) + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) \tag{3.10}$$

The model-based part than only controls the position, while the PD part controls the orientation. With this approach, it is possible to control both position and orientation with satisfactory accuracy. The disadvantage of this approach is that the desired compliant impedance behavior is only fulfilled for the translation and not for the orientation. The orientation now is much stiffer, but in general, this is not a problem.

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{rot} + \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}_{trans} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) \tag{3.11}$$

$$\boldsymbol{\tau}_{rot} = \boldsymbol{J}_{rot}^{T}(\boldsymbol{K}_P\tilde{\boldsymbol{x}}_{rot} + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{x}}}_{rot}) \tag{3.12}$$

$$\boldsymbol{q}_c = \boldsymbol{J}^{-1}(\boldsymbol{q})\left(\ddot{\boldsymbol{x}}_d + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{x}}}_{trans} + \boldsymbol{K}_P\tilde{\boldsymbol{x}}_{trans} - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}\right) \tag{3.13}$$

The resulting controller scheme is shown in Figure 3.16.

3.3.3 Null-Space Impedance Control

To avoid undesired joint positions generated by the task-space controller, the null-space of the manipulator should be controlled in parallel. The main goal of the used null-space control is to push the manipulator as good as possible in a desired joint position without influencing the task-space behavior.
A controller with independent task- and null-space stiffness is published in [56] and the same structure also is used in this project. It uses a model based joint-space controller based on the linearization shown in equation (3.5):

$$\ddot{\boldsymbol{q}}_c = \ddot{\boldsymbol{q}}_d + \boldsymbol{M}_d^{-1}(\boldsymbol{B}_d\dot{\tilde{\boldsymbol{q}}} + \boldsymbol{K}_d\tilde{\boldsymbol{q}} - \boldsymbol{\tau}_{ext}) \tag{3.14}$$

Because the controller should only influence the null-space, the control-law is projected in the null-space of the Jacobian with the projection $\boldsymbol{N}$. In general, this projection can be achieved by the projection matrix

$$\boldsymbol{N} = \boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J} \tag{3.15}$$

but as described in [57], this only guarantees that the task-space position is not influenced in the equilibrium point. It does not secure that the dynamic behavior in the task space is not influenced. However, this is an important requirement in the training use case, because then the end-effector is moved dynamically by the operator while the null-space should be controlled automatically.
To also reach dynamically consistent null-space control a special generalized inverse

$$\boldsymbol{J}^{\#} = \boldsymbol{M}(q)^{-1}\boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{M}(\boldsymbol{q})^{-1}\boldsymbol{J}^T)^{-1} \tag{3.16}$$

can be used to calculate the null-space projection $\boldsymbol{N}_{\#} = \boldsymbol{I} - \boldsymbol{J}^{\#}\boldsymbol{J}$. The joint-space control (3.14) law can be combined with the null-space projection and the task-space control-law (3.6) to the combined control law:

$$\begin{aligned}
\ddot{\boldsymbol{q}}_c &= \ddot{\boldsymbol{q}}_{task} + \ddot{\boldsymbol{q}}_{null} \\
\ddot{\boldsymbol{q}}_{task} &= \boldsymbol{J}^{\#}(\ddot{\boldsymbol{x}}_d - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}) \\
\ddot{\boldsymbol{x}}_c &= \ddot{\boldsymbol{x}}_c + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{x}}} + \boldsymbol{K}_P\tilde{\boldsymbol{x}} \\
\ddot{\boldsymbol{q}}_{null} &= \boldsymbol{N}_{\#}(\ddot{\boldsymbol{q}}_d + \boldsymbol{M}_d^{-1}(\boldsymbol{B}_d\dot{\tilde{\boldsymbol{q}}} + \boldsymbol{K}_d\tilde{\boldsymbol{q}} - \boldsymbol{\tau}_{ext})
\end{aligned} \tag{3.17}$$

This control law has the desired task space trajectory $\boldsymbol{x}_d$ and the belonging null-space position $\boldsymbol{q}_d$ as input and control them independently.

3.3.4 Application of the Model Based Controller

With the previously designed controller, it should be possible to control the task-space and the null-space independently. This can be used for an assisted teaching mode in which the user can move the end-effector of the robot and the null-space is controlled automatically. A system like this is also used in [24] and has the advantage that the user has not to care about the null-space, especially the joint limits and possible collisions of the robot arm with the environment.
To reach a behavior like this the K_D and K_P parameters of the task space controller must be set to zero, so the end-effector can be moved completely free. Besides, a null-space control law that resolved the redundancy[58], for example by pushing the robot away from the joint limits or to a defined position can be used.
The second application of this controller is the execution of trajectories and especially ProMPs. It is much easier to define the desired trajectory in task space, but when the joint configuration is not defined exactly, because the robot has more degrees of freedom in joint space than the task space. Using ProMPs which describe both, the task and joint space, it would be simple to generate trajectories that lead to the exact task space position and also provide an approximated joint space position. This can be done by conditioning on the task space, the joint space is then coupled through the stochastic ProMP structure. So the controller can use the task space trajectory to perform the exact movement and the approximated joint space trajectory to control the null-space.
In theory, this concept is very advanced, but through the implementation, the biggest problem was a missing useful robot simulator. The determination of good and stable controller parameters is really hard if they only can be tested on the real system. It was possible to find parameters that show a stable behavior in most areas, but they need a lot more optimization until they can be used safely on the real system. This can only be done in simulation because tuning the parameters on the real system takes too much time.
Because the developing of new controllers is not the main task in this project, the work on the advanced controllers was stopped and instead the normal robot controllers provided by Franka were used. They do not allow advanced null-space control and are very stiff if an accurate behavior is needed, but it is possible to use them at this state of the project. If a simulator for the robot is available in the future, the development of a new model-based controller is recommended.

4 Implementation

This chapter describes the implementation of the developed Imitation Learning system. In the following the used hardware setup, the developed program structure and the user interface are described.

4.1 Hardware Setup

The used robot is a special robot configuration from Franka Emika as shown in Figure 4.1. It has two Franka Emika Panda[1] arms mounted in a human-like position. The arms use the same structure as the most modern robots, so every arm has seven rotational degrees of freedom. They can hold a maximum payload of 3 kg and also support force and torque sensing for every joint and the end-effector.
The used end-effector on both arms is a simple two-finger gripper. The gripper also supports force sensing, so it can grasp an object with the desired force and hold the force afterward. This is very useful for grasping objects whose shape is not exactly known. All in all the gripper is using a very simple structure, so grasping more complex and round objects with this gripper is difficult. It is possible to mount new extensions to the gripper and use them to expand the capabilities of the gripper.
The robot also has an advanced image recognition system using an Intel realsense depth camera[2]. This camera can be used to recognize human movements with skeleton tracking and detect object positions for object tracking. In this project, the camera is not used, because object tracking using a camera is a complicated field and requires a lot of research on its own. Because this project should not focus on the tracking part, an Optitrack system[3] is used for tracking objects instead. It uses special markers and infrared light to track the objects and delivers more reliable and accurate object poses. The disadvantage of this system is that every used object must be prepared with the special markers. The system is also not portable and must be mounted and calibrated in the room where the robot is placed. This is a big disadvantage for the final task because in the real application it is not possible to mount an Optitrack system in every room. So in the final version of the robot system, the camera must be used for object recognition, but at this stage of the whole project using the Optitrack system instead makes the development much easier.

4.2 Program Structure

An often-used framework for controlling robots and developing robot applications is ROS[59]. It is a collection of tools, libraries and conventions that should simplify the development of complex and robust robot software. It allows the creation of independent nodes that can collaborate and provides a lot of useful methods, for example for localization and frame transformation.
The Panda robot can be controlled directly using ROS because it is using the ROS Control[60] structure for all controller implementations and provides all state information through ROS interfaces. Therefore ROS also is used for all other components of the whole system. The developed structure is shown in Figure 4.2 and can be split into three parts: The low-level part, including all functionalities for the generation and execution of robot arm movements, the mid-level part which controls the teaching process and manages the usage of ProMPs and the high-level part that basically includes the user interface and the high-level task representation.

4.2.1 Low Level - Movement Generation

This is one of the first projects working with this robot setup, so a lot of basic functions must be developed to use the robot. The most important function is to move the robot arms both in joint- and task-space. To do this a controller is necessary which controls the joint torques. Franka-Emika provides a Multi-Mode Controller which can control the robot on the desired joint or task-space trajectory. They both are closed source controllers, so no adaption is possible. This is a problem because the Task-Space Controller shows a very bad trajectory following behavior if the stiffness is not very high. Because of this the controller from Section 3.3 is implemented as *TUD-Task-Space Controller*, even if it is not perfectly working.

[1] https://www.franka.de/
[2] https://www.intelrealsense.com/
[3] https://optitrack.com/

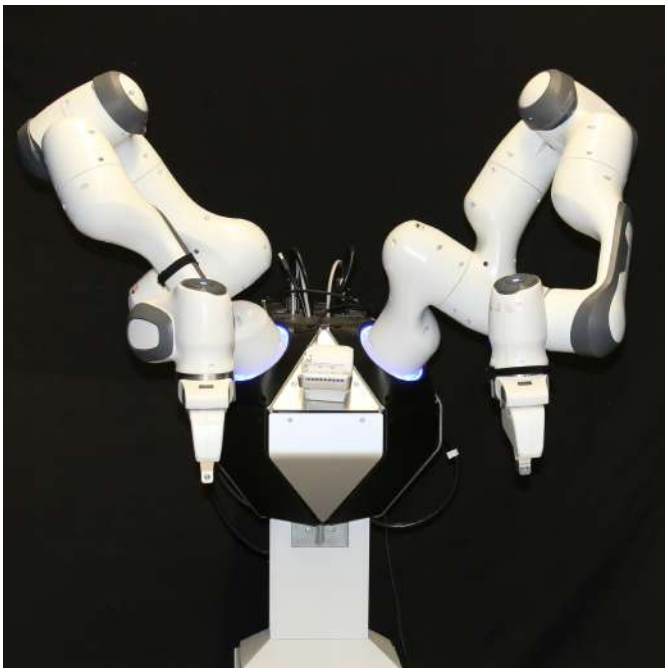


Figure 4.1.: The KoBo robot is using two Franka Emika Panda robotic arms and two grippers. They are mounted in a human-like position. Between the arms a movable camera is installed.

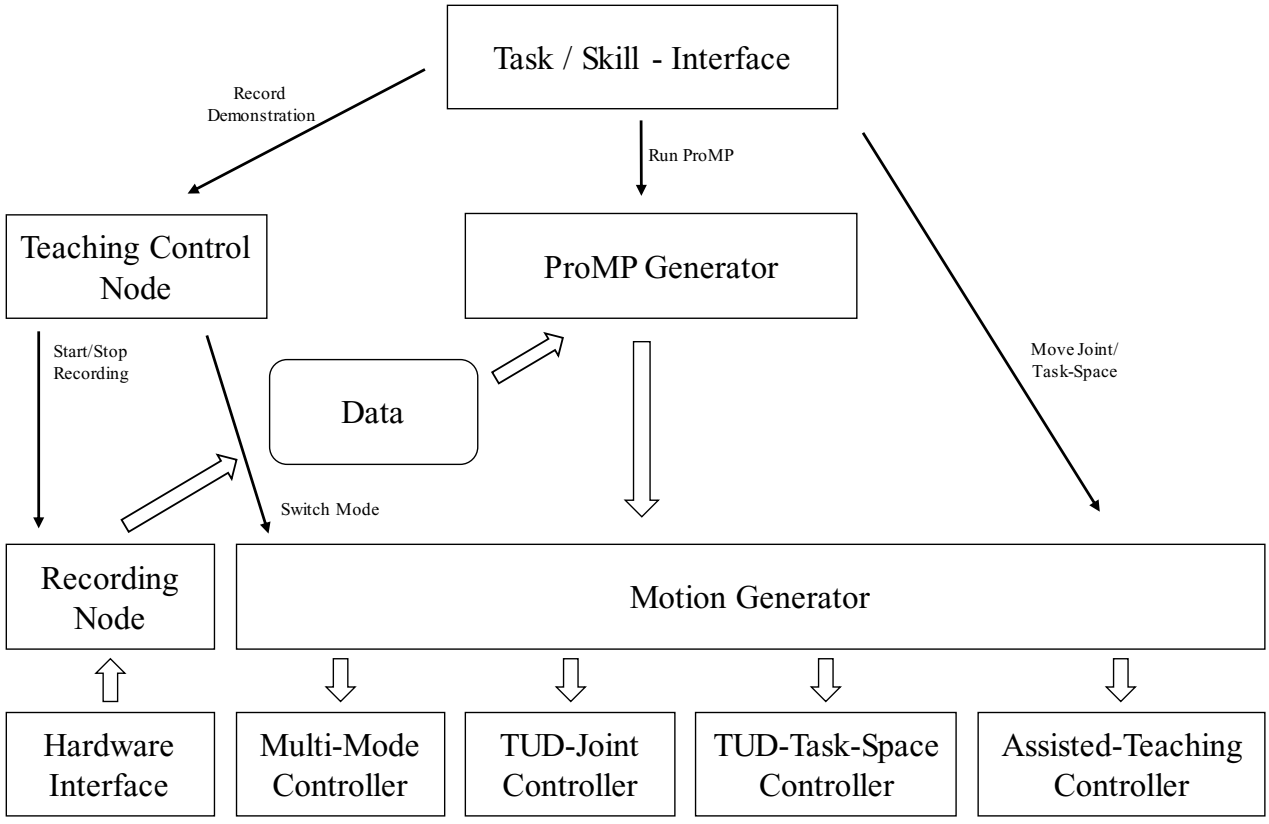


Figure 4.2.: This figure shows the structure of the low-level robot control and data recording programs. Big arrows show data flows and small arrows show commands.

Another controller type is the *Zero-Gravity* controller. It compensates the gravity and can hold the robot arm in the actual position. It also allows the user to move the arm without much force. This controller is used for kinesthetic teaching.
All controllers can only follow a given trajectory, they cannot reach a position that is far away from the start position. Because of this a *Motion Generator* is needed, which generates a trajectory from the actual starting position to the desired goal position including specified via-points. The *Motion Generator* also has some extra functions: It automatically switches the control mode running on the robot and starts and stops the different controllers. The second function is that it can recover the robot from error states like joint-limit violations or collision detections. All in all the *Motion Generator* is the most important component to interact with the robot.
Another part of the low-level functions is the *Recording Node*. It synchronizes the states of both robot arms and can record the positions over time and also save them in files. This is a basic functionality used for demonstration generation.

4.2.2 Mid Level - Teaching and ProMPs

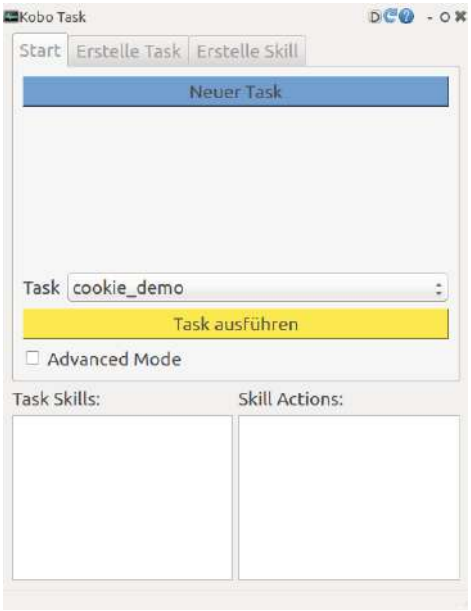
The mid-level includes the teaching functions and the management of ProMPs: To control the overall teaching process, the *Teaching Control Node* is used. It can be assessed during ROS services and starts and stops the recording of the robot state. It also automatically switches the robot control modes, so the user has not to care about this on his own. The third function is the management of the ProMP demonstration recording process like it is explained in Section 3.2.3. This means that the demonstrations are stored in the right folders and for the coordinated teaching process the movements are replayed.
A second component of the mid-layer is the *ProMP Generator*, it is used to execute ProMPs on one or both robot arms. It can load demonstrations from the storage and train a new ProMP using demonstrations or use a pre-trained ProMP. The *ProMP Generator* can condition the ProMP on a new end-position and sends the execution commands to the *Motion Generator* which can execute the ProMP in both task- or joint-space.

4.2.3 High Level - User Interface

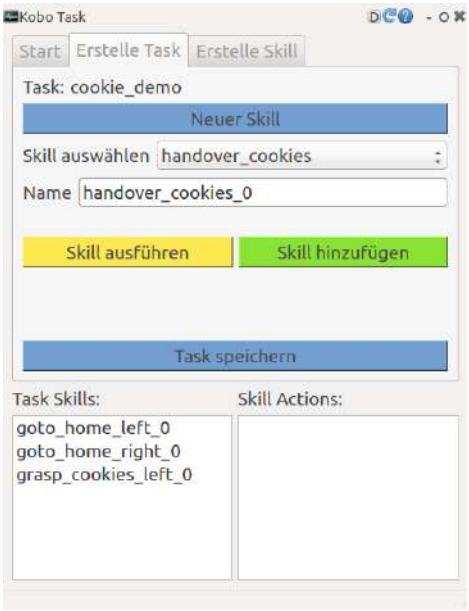
The high-level software consists of the user interface, the implementation of the task representation structure and the implementation of all teaching processes. Because the user interface is an important part on its own, it is described in Section 4.3.
The tasks and skills all are structured as behavior trees. Because behavior trees are a complex concept, the Behaviortree.CPP[53] implementation is used, instead of developing an own one. It has some advantages over other implementations, for example, it can execute asynchronous actions, it allows the creation of new trees on runtime and allows a simple generation of custom treenodes. All in all the implementation fulfills all the requirements of this project. In this library, the trees are stored as XML files. To create them a special Graphical Editor can be used or the XML syntax can be implemented manually. Both ways are not simple enough to use them with inexperienced users, so an own custom system is developed to create the tree XML-files automatically from the defined Task-Skill-structure. It is using two custom classes for Tasks and Skills and directly represents the task representation structure.
All teaching processes explained in Section 3.2 are also implemented with behavior trees. This allows a simple and safe creation of new processes that can be adapted easily in the future.

4.3 User Interface

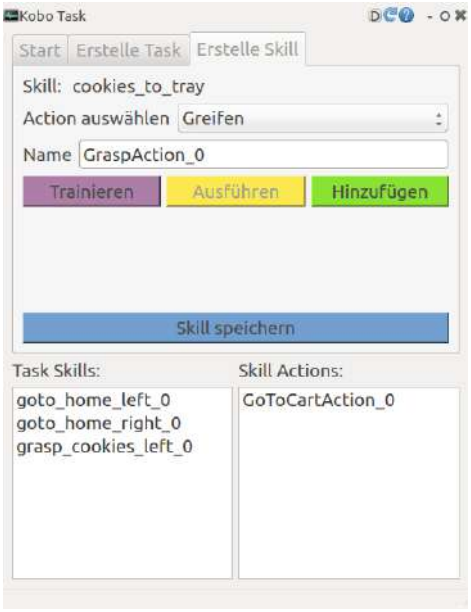
The teaching process should be executed completely alone by the user, so a user interface is needed to control both, the robot and the teaching process itself. Using terminal commands and outputs like it is often done for research robots is no option, because this is very confusing for all users without a computer science background. So a graphical user interface is used, which can be controlled on a touchscreen. The user interface is German because all users in the user study are German.
The most important windows of the user interface are shown in Figure 4.3: At first the start tab is shown, the user can select to create a new Task or execute an existing one. If he wants to create a new one, the *Create Task* tab is opened and the user can select existing Skills to execute and add them to the task. In addition the user can create a new Skill, using the *New Skill* button and the *Create Skill* tab. In this window, the user can select an action he wants to add to the Skill. At first, the action must be trained. Afterward, some actions can be executed, while others are executed during the training process. The training is done with different windows, similar to the shown example. The training process used in the training windows exactly follows the developed process from Section 3.2.2. After training the action the user has to add the Action to the Skill.
All Skills from the opened task are shown in the box on the left bottom side, while all Actions in the actual Skill are shown on the right side. So the user can get a short overview of the steps he had finished yet.



(a) Start Tab



(b) Create Task Tab



(c) Create Skill Tab

(d) Train Action Tab

Figure 4.3.: The user interface has three main windows, the *start*-, *create task*- and *create skill*-tabs. The user is guided through the complete process, so he only has to use *New...* and *Save...* buttons, the window is changed automated afterwards. To train the Actions, different windows are used, one example is shown in Figure 4.3d.

5 Experiments

In this section, the developed teaching process should be evaluated. To do this it is necessary to test the teaching with inexperienced users and to assess the generated ProMPs. To do this a user study was carried out, which is explained in Section 5.1 and evaluated in Section 5.2.
The trained ProMPs are examined in Section 5.4.

5.1 User Study Description

The main goal of this user study is to evaluate if users, which are inexperienced in training and programming robots, can teach new tasks to a robot if the teaching interface is intuitive enough. To do this the user should use the user interface and teaching processes shown in Chapter 3, so these processes also should be evaluated.
Another important point in this user study is the comparison of GoTo and trajectory following commands. To execute and train trajectories that can be generalized to new positions ProMPs are used. The most known projects only had evaluated ProMPs with professional operators and more difficult tasks, but not with simple tasks and inexperienced users. So it should be evaluated if successful training of ProMPs is possible and accepted by users without a robot or computer science background.

5.1.1 Setup

In the user study, the user interacts with the KoBo robot described in Section 4.1. In the future, the robot should support the people and the staff in a home for elderly people and therefore it also should perform simple household tasks in this user study.
To create an attractive and well-arranged environment two different objects, the robot can interact with, are used: A simple tray and a pack of cookies, both shown in Figure 5.1. Because the gripper of the robot is a very simple construction, it was necessary to select objects that are very simple to grasp. This is fulfilled by the two selected objects. Another important point is the mass of the object: If they are too heavy the zero-gravity control mode is not working as requested anymore.
Both objects are prepared with OptiTrack markers, so the object tracking system can easily measure the position and orientation.

5.1.2 Tasks

The user study is split into three different experiments every user should take part in. The explanation of the three experiments with a short manual of all Actions that are given to the users can be found in appendix A.2, an overview is given in the following.

5.1.2.1 Experiment 1: GoTo

In Experiment 1 the robot should grasp the cookies and lay them on the new position. The exact desired procedure is the following:

- The robot grasps the cookies with the left arm
- The robot hands over the cookies to the right arm
- The robot places the cookies on the tray
- In the end both arms are in the start position again

To train this behavior the user can use the Actions: *Grasp*, *GoTo Object* and *GoTo Fixed Position*. So the user can only use GoTo commands and no trajectory following commands. This makes the teaching simpler, but the user also has to define via points at the right positions to ensure that the robot is moving on an allowed trajectory and not colliding with objects or the environment.
The robot has no predefined skills in the beginning, so the user has to create completely new tasks with new skills. So this experiment can be used to examine both, the high-level task structure and the low-level GoTo Position teaching.

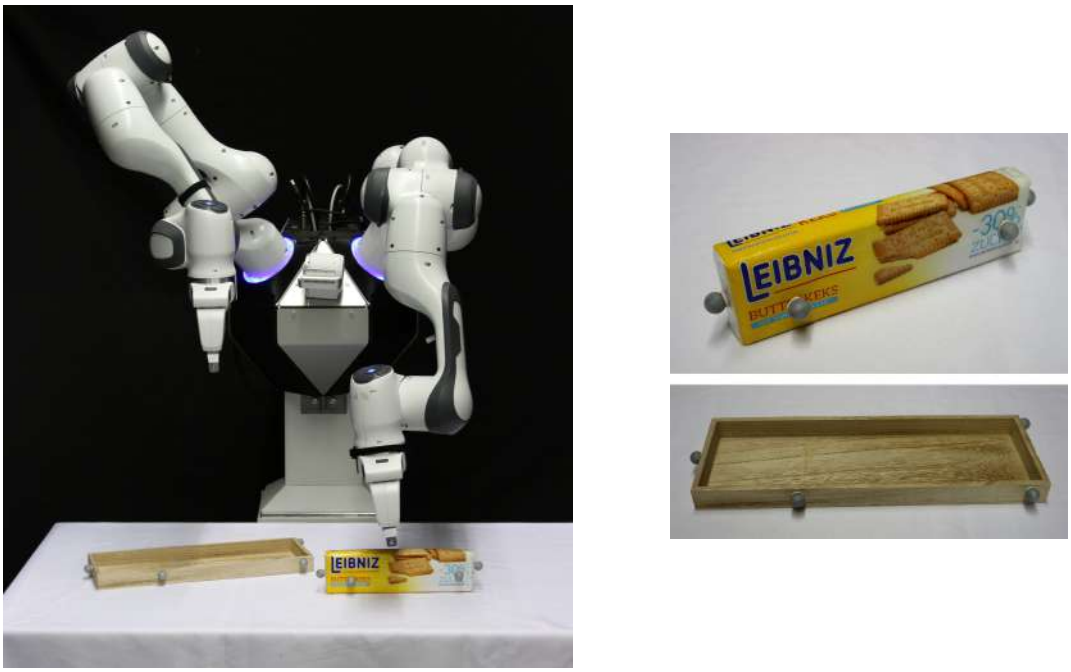


Figure 5.1.: In the user study the KoBo robot is used with two objects, a simple tray and a package of cookies. Both objects are prepared with Optitrack markers to allow an exact object tracking.

5.1.2.2 Experiment 2: ProMP

Experiment 2 is using the same assignment like Experiment 1, but instead of using GoTo commands the user now must use the *Execute Movement* Action which is executing a ProMP. To do this the user has to give four demonstrations following the ProMP teaching process.
This experiment can be used to evaluate how successful the users can train ProMPs, how good the generalization capabilities are and if the training of a ProMP is accepted in general or even preferred over using a GoTo command. To get unbiased information about this, the users were not given background information about the way ProMPs are calculating the trajectory. They only know that the robot will execute a trajectory similar to the demonstrated ones. The user also is asked to move the object before giving a demonstration to ensure some generalization.

5.1.2.3 Experiment 3: Both Arms

In Experiment 3 the robot should use both arms to grasp the tray and lift it. The desired procedure is the following:

- The robot grasps the cookies
- The robot places the cookies on the tray
- The robot grasps the tray with both arms at the same time and grasps it
- The robot lifts the tray

In this experiment, the users can use both movement types, GoTos and ProMPs, and also all Skills they have created before. To train the tray grasping step the *Coordinated* ProMP teaching mode should be used, which also is described to the user. For the training of the *Lift Tray* step it is necessary to train a *Simple ProMP* with both arms, so the user has to move both arms at the same time. This becomes more simple than normal because both arms grasp the tray and are connected over it.
This experiment can be used to evaluate if the user can perform the two different both arm teaching modes. A second interesting point is that it is necessary to define a via point or use a ProMP if the robot should move the cookies on the tray without colliding with the tray before. So it can be evaluated if the user remembers this recommendation or is trying to execute this step with only one GoTo command.

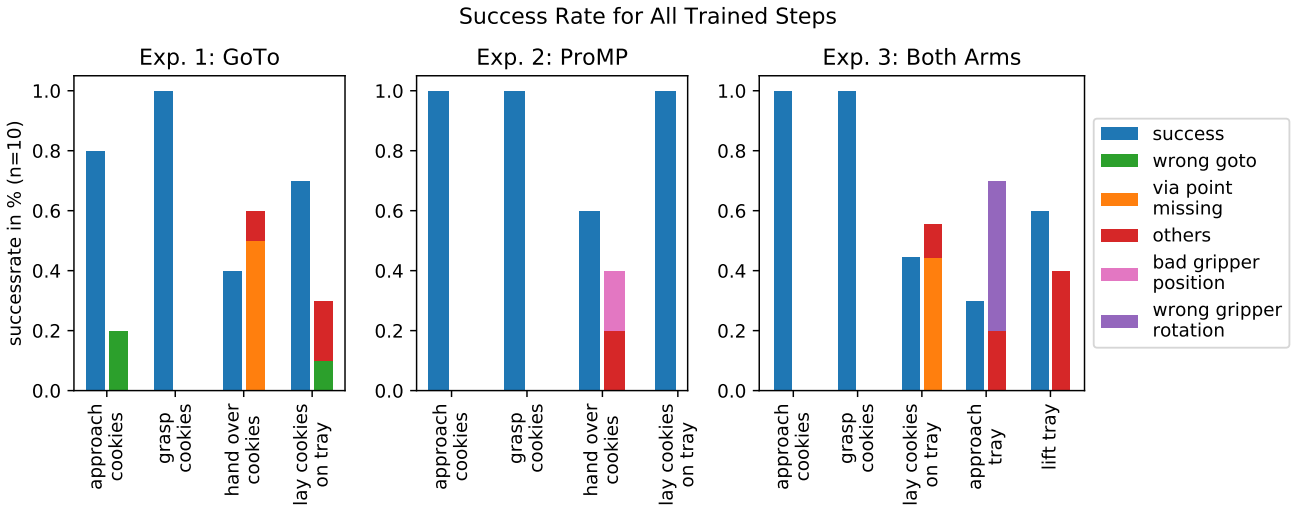


Figure 5.2.: This figure is showing the success rate for all three experiments. Every experiment is split into small steps and each step is rated on its own. The left blue bar shows the proportion of users that trained the step in a way that it can be executed completely successful. The right bar is showing the proportion where the execution failed. Some common errors are shown with different colors: The *Wrong GoTo* error means that the user used a *GoTo Fixed Position* instead of *GoTo Object*. *Via Point Missing* means that the user forgot to define via points for the robot trajectory and so the robot collided with the object instead of approaching it the right way. The *Bad Gripper Position* error occurs, if the gripper is not positioned sufficiently accurate and failed to grasp the object.

5.1.3 Users

The user study is carried out with ten users, who have no background in robotics. One-half of the users were male the other female and the age of the users was between 18 and 55, while half of the users were between 21 and 29 years old. All except one user were students, but they all are studying subjects not directly related to computer science and robotics. This also can be seen in the overview questions in appendix A.1: Only two users answered the question about the robot knowledge with "I have basic knowledge about the construction and functionality of robots". The rest reported that they have no experience or only have seen robots in action.
The knowledge about programming languages also was asked and only two user reported that they use them occasionally(At least once per month) or regularly(At least once per week). The rest only had basic knowledge or no experience at all. So it can be said that the tested user group has less background knowledge like it is needed to evaluate a system for inexperienced users.

5.2 User Study Results

In this section, the results of the user study are reported. At first, it is evaluated in Section 5.2.1 how successful the training was and if the task could be executed without errors. In the second step the feelings of the users about the training in general and the different movement types are evaluated in Section 5.2.2.

5.2.1 Training Success and Common Errors

To assess the training success every experiment is split into small steps that are analyzed on their own. An overview of the analysis is shown in Figure 5.2. It shows the success rate for each step of all three experiments as bars. The bars are split in the two parts *Successful Execution* and *Execution with Errors*.
The first step of Experiment 1 and 2 is *approach cookies*, so the robot should move the gripper in a grasping position near to the cookies. Using GoTo Actions 80 % trained a successful skill, the other 20 % used a *GoTo Fixed Position* instead of *GoTo Object* Action, so the Skill cannot generalize to new cookie positions. Using ProMPs all users selected the right object and trained a ProMP that was able to grasp the cookies at a position similar to the training position. The generalization capabilities are analyzed in Section 5.4.2 and at this point, a ProMP step count as successful, even if the generalization capabilities are small.

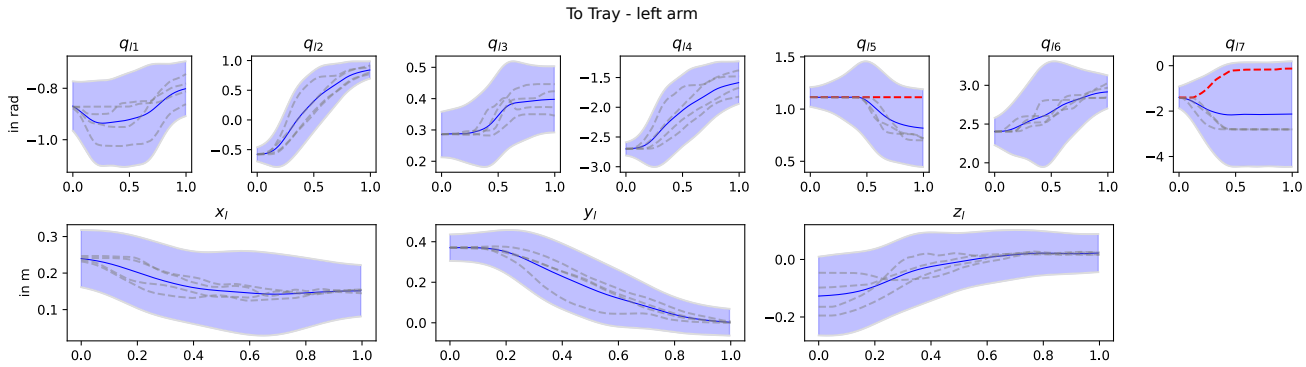


Figure 5.3.: This is the left arm part of an example ProMP trained to approach the tray. This ProMP cannot be executed successfully, because the operator gave one demonstration where the gripper was rotated in a different direction. So the demonstrations are not similar enough. This can be directly seen in the gripper joint q_7 and also in q_5, where the wrong demonstration is highlighted in red.

After approaching the cookies the robot should grasp them. The *Grasp* action was trained successfully by every user in all experiments and no problems occurred. The users also understood that it is necessary to train the *Grasp* Action again for different objects.

The third part of the task is the handover from the left to the right arm. This is more difficult because the robot has to lift the cookies first and grasp them with the other hand. Using GoTo Actions only 40 % of the trained Skills were successful. The main error, done by 50 % of the users, was that no via points were defined. When the user tries to move the cookies from the start position directly in the right gripper, the cookies often collide with the gripper and the robot stops. Using ProMPs this happened less often and 60 % of the users trained a successful Skill. But in this step, a new error occurred: The cookies were not positioned well in the second gripper and so the gripper lost them during the handover. This is an issue that is mainly related to the gripper itself and not the training, because the gripper is very simple and using it for grasping objects sometimes is hard.

The last step is the placement of the cookies on the tray. This was successful for 70 % of the GoTo skills and all ProMPs. Again some users used the wrong GoTo action and some others grasped the cookies at such a bad position during the handover that the placement failed.

In Experiment 3 the first two steps again are approaching and grasping the cookies. Because the users were allowed to reuse the skills they trained before, all users succeeded in these two steps. The third step is the placement of the cookies on the tray and is much more interesting because only 40 % trained a successful skill. In this experiment, the user could decide to use GoTos or ProMPs on their own, but 80 % of the users using GoTos forgot to define the necessary via point and the cookies collided with the tray. This leads to an overall failure rate of 40 % only because of missing via points. In contrast, all users using a ProMP(30 %) were able to train a successful skill. This could be a first indicator for the point that GoTos are unintuitive to use and training whole trajectories is more intuitive.

In the *approach tray* step the *Coordinated ProMP* teaching process is used to train a trajectory that can be executed on both arms at the same time. The success rate for this step is only 30 %, but the main errors are not related to the both arm training: The most users tried to grasp the tray from the side, which needs a 90 ° rotation of both grippers. The grippers are symmetric, so the rotation can be done in both directions, but for a successful ProMP training it is necessary to give all demonstrations with rotation in the same direction. An example of a not working ProMP is given in Figure 5.3, where the wrong demonstration can be seen directly. This problem was not foreseen in the planning of the teaching process, but in the future, an algorithm to automatically find demonstrations that are not similar enough must be used.

Compared to this 75 % of all users grasping the tray from the backside were successful, so the teaching process itself seems to be usable for most users. They also had no problems with the synchronization of the movement of both arms, so the *Coordinated ProMP* seems to be a good solution.

The last step is using a *Simple ProMP* for both arms to lift the tray. The training for this step is a bit difficult because the grippers cannot hold the tray very secure, so it can happen that the tray is falling. Yet 60 % of all users trained a successful skill, the rest had problems with moving the grippers and the tray. An improvement of the grippers so that they can hold the tray stronger and more stable would help at this point.

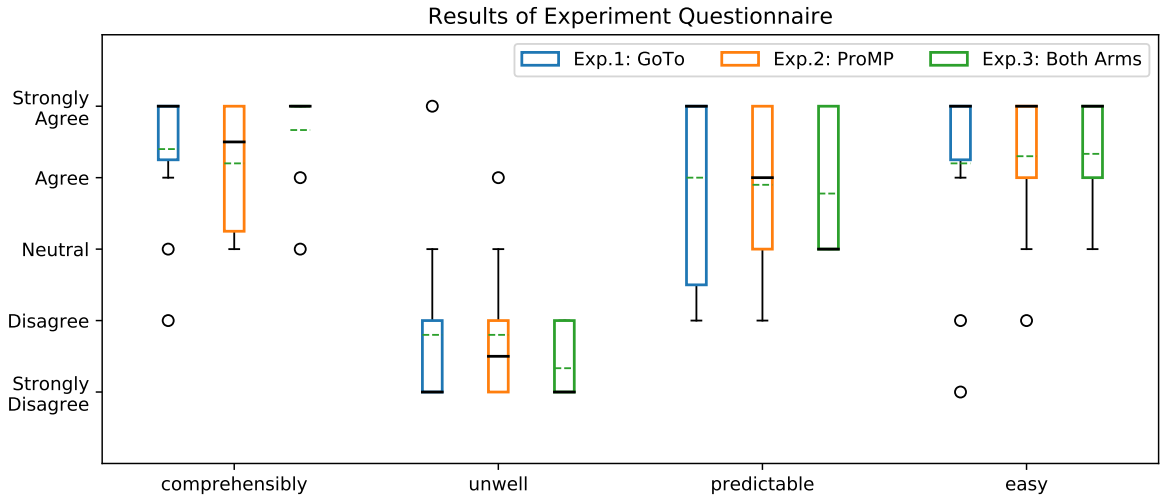


Figure 5.4.: The box plots show the feelings of all users about the robot teaching process and compare the three different experiments. A normal box plot with the median as a black line and mean as a green stashed line is used to represent the results. The users were asked after each experiment if the teaching process is comprehensible if they unwell using the robot, if the robot behavior is predictable and if it was easy to move the robot at the desired position.

5.2.2 User Feelings

In this chapter, the user feeling about the three experiments and the different movement types are analyzed. To do this the users had to answer some questions after each experiment, the results are shown in Section 5.2.2.1. After all experiments the users were asked to compare the two movement types (Section 5.2.2.2) and to rate the workload (Section 5.2.2.3).

5.2.2.1 Experiment Comparison

To analyze the user feeling about the experiment and the movement types, the users were asked the following four question, which should be answered using the 5-point Likert Scale(Strongly Disagree to Strongly Agree):

- The training process is comprehensible
- I felt uncomfortable when using the robot
- The robot behavior is predictable
- The movement of the robot to the desired position is easy for me

The full questionnaire can be found in appendix A.3 as *Questionnaire 1*. The result is shown in Figure 5.4, where the three experiments are compared for every question. The results were analyzed using a nonparametric ANOVA with the Kruskal Wallis test to find statistically relevant differences. A significance level of $\alpha = 0.05$ is used, which is often used in the literature[41] in similar studies. The results are shown in Table 5.1, but none of the differences is statistically relevant, so the results are analyzed using the median and quartiles.

Table 5.1.: ANOVA User Feelings

	p_{E1E2}	p_{E1E3}	p_{E2E3}
Comprehensible	0.492	0.635	0.216
Unwell	0.770	0.567	0.331
Predictable	0.714	0.631	0.760
Easy	0.824	0.741	0.927

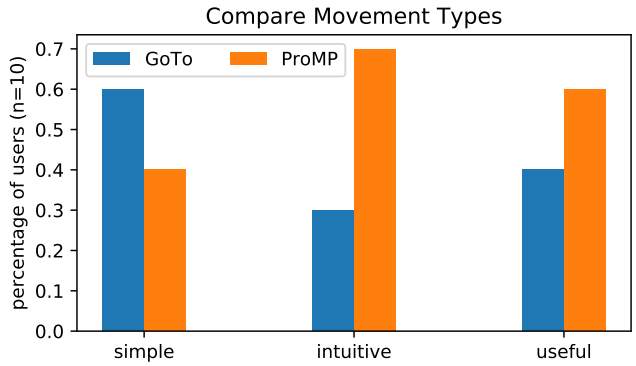


Figure 5.5.: The bars compare the user feelings about the movement types GoTo and ProMP. The user was asked for which type the usage was simple, more intuitive and more useful.

The first question about the *comprehensibility* can be used to rate the teaching process itself. High approval is wished because the teaching process should be clear and not confusing for the user. For the GoTo experiment 80 % agree to the statement and the median is *Strongly Agree*. This is the expected result because teaching a GoTo is very easy with this framework.
In comparison to the first experiment, the ProMP experiment leads to a bit worse results. This also is expected, because training a ProMP is much more complicated. But all in all the results are still good because no users disagree with the comprehensibility statement. The result for the both arm experiment is surprising, because nearly all users *Strongly Agree* to the statement, even if training a both arm ProMP is much more complicated. One reason could be that this is the last experiment and the users get used to the whole teaching process.
The opinion of the users on the predictable robot behavior differs a lot between the three experiments: For the GoTo experiment, the median is *Strongly Agree*, so more than half of the users think that the behavior is predictable. Compared to the ProMP experiments this is a much higher value. This result makes only little sense if it is compared to the fact that most users forget to define via-points for the GoTo command and so the trained Skills often failed. It seems that most users think that the GoTo behavior is easy to predict, but they predict the wrong behavior. This is a fundamental problem because it results in that the user often uses GoTos, but they use them in the wrong way.
The users also think that the ProMP experiment is harder to predict, the median now only is *Agree*. This result could be explained by the fact that the users have to train the ProMP four times and do not completely understand why they have to do it that often. Another point is that ProMPs show completely unpredictable behavior if the applied condition is outside of the trained area. To avoid this it could be useful to use an algorithm that can detect if the condition is outside of the trained area and rejects the execution.
The both arm experiment seems to be much harder to predict for the users, but this could be related to the high number of failures because of the wrong demonstrations as explained in Section 5.2.1.
The question if users feel unwell during the robot usage was contradicted by nearly all users. Both mean and median are in the area of disagreement and there is no big difference between the three experiments. The results are a bit worse for the ProMP experiment, but this also is expected.
In all three experiments, the users have the opinion that moving the robot to the goal position is easy: For all three experiments, the median is *Strongly Agree* while 75 % of the users agree to the statement and only a few outliers disagree. The good results on the *unwell* and *easy* questions could be an indicator for the point that kinesthetic teaching is the right method to teach the robot and is in general accepted by the users.

5.2.2.2 Compare Movement Types

After all three experiments, the users were asked to compare the training process of the two different movement types, GoTos and ProMPs. They were asked the following questions in Questionnaire 2 (Appendix A.4):

- For which movement type the training is easier?
- For which movement type the training is more intuitive?
- Which movement type is more useful?

The results of this questionnaire are shown in Figure 5.5: For the first question 60 % of the users have the opinion that teaching GoTos is easier. This is the expected result because the training itself is much faster and less complex. It is

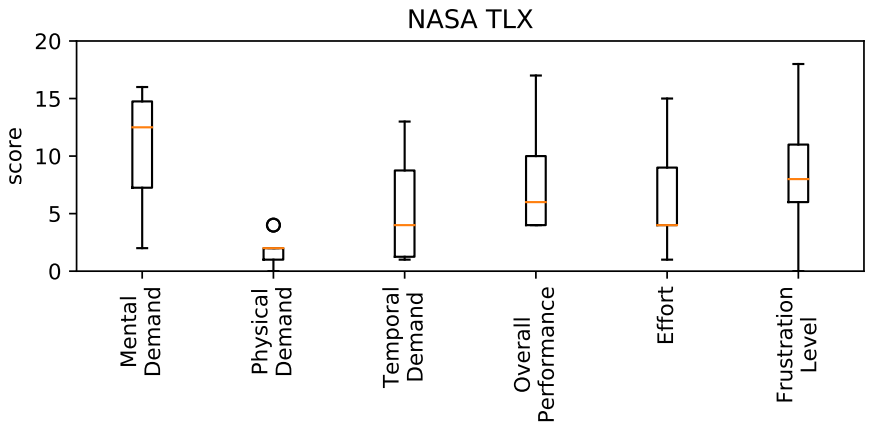


Figure 5.6.: This Boxplot shows the user workload using the widely used NASA Task Load Index. Low scores describe a low load or a good performance.

interesting that still 40 % of all users think that ProMP teaching is easier even if it much bigger effort. It seems that they take into account that the GoTo action needs the definition of via-points to work correctly.
In terms of intuitiveness 70 % of all users prefer ProMP teaching and 60 % think that they are more useful. Together this is a clear result for the acceptance of ProMPs and it seems like the users even prefer training ProMPs over using GoTos. This speaks for the usage of ProMPs in an intuitive Imitation Learning systems, especially concerning the high failure rates using GoTos.

5.2.2.3 Workload

At the end of the user study, the users should rate the workload. This is done with the widely used NASA-TLX[61] assessment tool which splits the workload into six subscales: Mental Demand, Physical Demand, Temporal Demand, Performance, Effort and Frustration. The user can rate each subscale on a scale from zero to twenty, while low scores describe a low workload or a good performance.
The results are reported with box plots in Figure 5.6. At first, it can be said that the opinions on the workload strew a lot, only at the physical demand the users agree that it is low, but this is not surprising, because the users had not to do any physical exhausting work. The opinions on the mental demand provide no clear picture: More than 50 % of all users think that the mental demand is higher, but some also rate it as very low. Also, no clear context between the mental demand and the users working background and previous knowledge can be found.
The temporal demand is rated as lower by 75 % of all users, but there were no time limits for the users and so this is not meaningful. The rating of the overall performance is similar to the results of the examination of the training success in Section 5.2.1: 50 % of all users think that they had a good performance and 25 % rate it with neutral. The rest of the users has the opinion that their performance was bad, which could be related to the high number of failures in experiment 3. Related to the performance the frustration level also strews a lot, from zero values to values near 20. So the frustration allows no general statement, it is highly depending on the user. In the future, it should be focused to minimize the frustration level, because high frustration will lead to users not using the system.
The effort the user has to apply is rated as low by 50 % of the users, the rest thinks that it is a bit higher (scores from 7 to 15). All in all the results are better than expected: For all users training a robot was a completely new task, so a high workload could be expected. Instead, the median of the users thinks that the workload is lower in all subscales, except for the mental demand.

5.3 Analyze of Used Task and Skill Structure

The analyze of the used task and skill structure applied by the users should give an overview of the reusability of all skills created by the users. If the defined skill has a high number of actions this is in an indicator that the skill is executing a very specific procedure, so it hardly can be reused in other Tasks. If the number of actions per skill is low, for example only one action in every skill, the skills execute a very abstract behavior. This is harder to imagine for the operator, so the reusability also is not optimal. But the reusability of very small Skills is still higher compared to very large skills because in theory the Skill can be reused very often, even if the application is harder for the user.

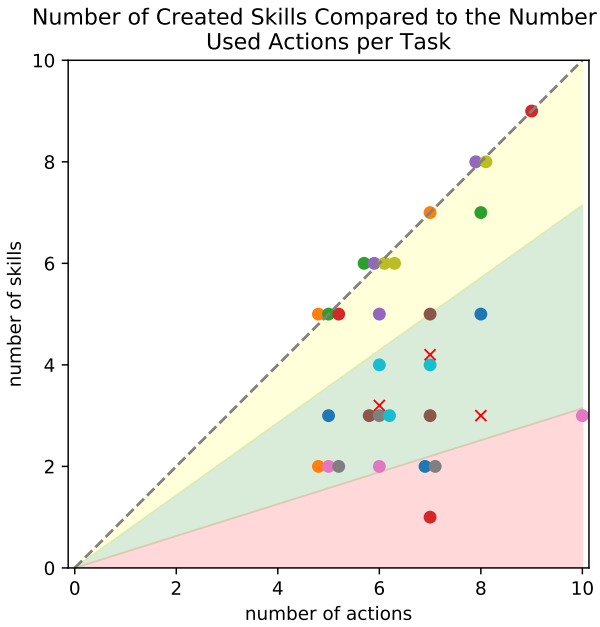


Figure 5.7.: This figure shows the number of used Skills compared to the number of used Actions in a Task. Every dot marks a Task created by the users and every user is represented by their own color. The red crosses are examples of well-structured tasks. The dashed line marks the border at which the number of skills is equal to the number of actions. In the green area, the tasks are structured similar to the recommended structure while the tasks in the yellow area have a lot of skills with a little number of tasks. In the red area, the tasks use only a few skills with a lot of actions.

In Figure 5.7 the number of skills and actions for each trained task in the user study is shown. Besides, three example tasks are added which use a skill architecture like it is recommended in terms of reusability. Starting from this an area is marked in green which includes all tasks with a similar number of actions per skill, so these skills also have a structure that be reused easily. The red area marks tasks that have a too high number of actions per skill. So they cannot be reused for other tasks, because their behavior is very specific. The yellow area describes the tasks where the most skills only use one action.

It can be seen that one-third of all tasks only use one action per skill and only a few users define very long skills. But only one-third of the created tasks are in the recommended area, so it seems that the users were not using the Task-Skill structure in the way it is thought. One reason for this could be the fact that the users were not instructed to define Skills with a useful structure and because the whole theme was completely new for them, they were not thinking a lot about this.

Another problem for a lot of users was the naming of the skills they created. It seems to be difficult to find useful names for people without a background in programming because they are not used to work like this. Often the skills were named similar to "move arm down" or "grasp" instead of easily understandable names like "move right arm to cookies" or "grasp cookies". The bad naming of the created skills confuses the users later if they want to reuse a skill they trained before. This was allowed in Experiment 3, but ~25 % of all users were not able to reuse their skills, even if they wanted to because they could not remember the right name.

This simple problem was not predicted during the development of the framework, but it causes a lot of issues in the real application. It is difficult to provide a technical solution for this problem, but it could help to explicitly remember the users to give a meaningful name and giving some examples when they create a new Skill.

5.4 ProMP Evaluation

In this section, the usage of ProMPs in general and the training of ProMPs by inexperienced users should be evaluated. To analyze the ProMP behavior in general on these tasks, at first the reconstruction error is considered in Section 5.4.1. To rate the training of the ProMPs, the generalization capabilities are analyzed in Section 5.4.2.

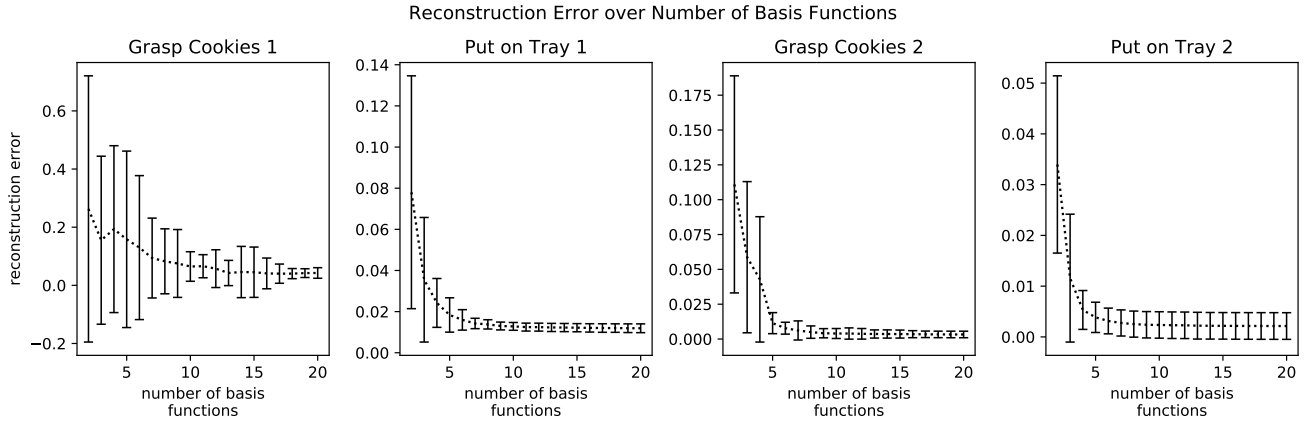


Figure 5.8.: Every plot shows the Reconstruction Error (after equation 5.1) for an different ProMP and an increasing number of basis functions. The dotted line is showing the mean of all given demonstrations while the error bars show the 2σ surrounding. The ProMPs were used to approach the cookies and lay them on the tray in Experiment 2 and were trained by two different users.

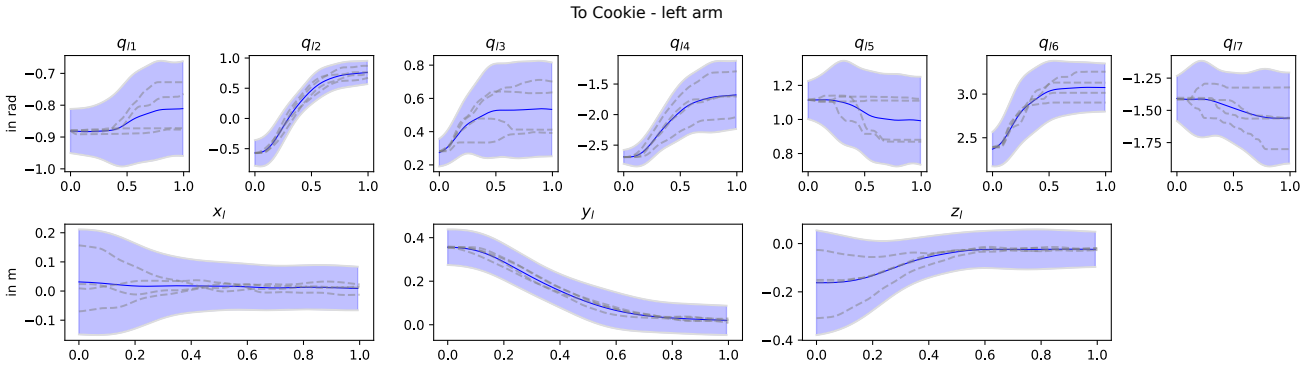


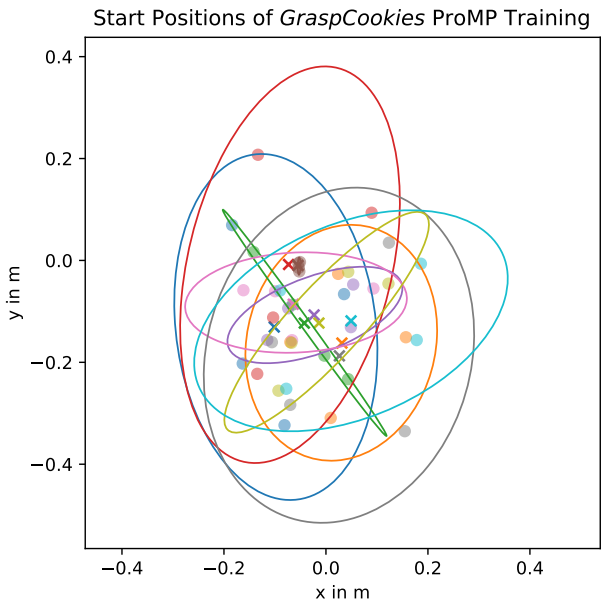
Figure 5.9.: Example for a well trained ProMP used to grasp the cookies in the first step of Experiment 2. It can be seen that the given demonstrations are similar, but the object position is different. The course of the mean is similar to the demonstrations, so the ProMP trajectory will follow a similar way. The task space trajectory of the robot end-effector is shown in the object frame, so different object positions lead to different start-positions, while the end position is the same. In joint space the start position is the same while the end-position is different.

5.4.1 Reconstruction Error

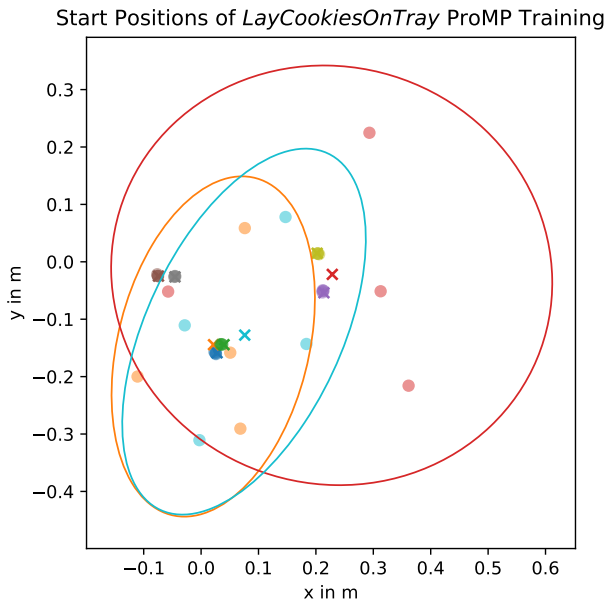
The reconstruction error of a ProMP can be used to evaluate how well the ProMP can describe a given demonstration. This is important to train ProMPs that can reproduce the desired behavior. The calculation of the reconstruction error e_R for a ProMP and one demonstration is described in equation 5.1, where d is the demonstration and d' is the reconstructed trajectory using the ProMP weights and conditioning on the start-point of d.

$$e_{R,i} = \frac{1}{T}\sum_{t=0}^{T}\left(\boldsymbol{d}_i - \boldsymbol{d}_i'\right)^2 \tag{5.1}$$

In this application, every ProMP is calculated from four demonstrations ($i = 1, ..., 4$), so the reconstruction error is calculated for every demonstration. The results of this calculation are shown in Figure 5.4.1 for different ProMPs and an increasing number of basis functions.
The chosen ProMPs were each two for grasping the cookies and to put them on the tray in Experiment 2. These ProMPs were trained well by the users and also can generalize well to new object positions. For every ProMP the mean and the 2σ-surrounding of the four demonstrations is plotted, which both are decreasing with a higher number of basis functions. At first, it can be said that the reconstruction error is very small for a higher number of basis functions. To visualize this one of the example ProMPs with ten basis functions is shown in Figure 5.9. It can be seen that the ProMP can describe the demonstrations very well and that the demonstration has a good amount of generalization.



(a) Cookie positions of all demonstrations for the *GraspCookies* ProMP used in the beginning of Experiment 2.



(b) Tray Positions for all demonstrations for the *LayCookiesOnTray* ProMP used in the end of Experiment 2.

Figure 5.10.: This figure evaluates the generalization capabilities of the trained ProMPs and is showing the different object positions the users used. The object positions of the four demonstrations are shown in different colors for each users and the ellipse indicates the 2σ surroundings. The cross symbolizes the mean of the four demonstrations. In Figure 5.10b the most users forgot to move the tray and used the same object position four times, so no 2σ-surroundings are visible.

Using Figure 5.8 and the reconstruction error the recommended number of basis functions can be evaluated: In general the number of basis functions at the point where the mean is short before constant should be used. In three of the example ProMPs, this is a value of six to seven basis functions, but the first example is showing a less predictable behavior and needs more basis functions. The mean of the error becomes constant at 15 basis functions, but a number of ten should be enough. In this user study always ten basis functions were used to execute the trained ProMPs. This seems to work well in most cases, but in the future, the usage of an algorithm to determine the best number of basis functions is recommended.

5.4.2 Generalization

When a ProMP is trained it is important to give demonstrations for different object positions because this is the main factor for the generalization capabilities of the ProMP. Because of this, it is examined where the users position the objects for their demonstrations. This is done for two examples in Figure 5.10. The first example is showing the position of the cookies for all demonstrations in the *Grasp Cookies* step of Experiment 2. In the second example, the tray position in the *Lay Cookies On Tray* step is shown. For every user, the mean and 2σ-surrounding is plotted as an ellipse in different colors. This can be used to assess the area in which the ProMP can be generalized.

For the first example, all users moved the object at variable positions and created different demonstrations. Some users only moved the object 15 cm away from the start position. That is a bit less for good generalization capabilities, but it still can be used. It must be said that the users were not instructed on how they must move the object and what the background of this procedure is, so most of them intuitively selected a good area to train the robot.

A general problem can be seen in the second example: Only three users moved the tray to the new position when giving the demonstrations, the rest gave four demonstrations with the same object position. In this experiment, this was not generating problems, because the tray was not moved before the trained task was tested, but in a real application, these ProMPs cannot be used. It seems like the users simply forget to move the object, even if they are told so in the teaching process. So in future, a system that can detect if the object was moved, should be used to remember the user.

Table 5.2.: Compare Movement Types

	GoTo	GoTo with Motion Planner	Trajectory Following	ProMP
Simple Training	+	+	o	-
Avoid Collisions	-	+	+	+
Generalization	+	+	-	+
Calculation Time	+	-	+	+

5.5 Movement Type Comparison

Finally Table 5.2 compares the used movement types GoTo and ProMP and besides the possibilities of GoTos with motion planning and following a single trajectory. A GoTo with a motion planner is using a complex motion planning system that knowns the environment of the robot and the robot arm limits and parameters to calculate a trajectory to the desired end-point. This trajectory has no collisions with the environment and respects the joint limits of the robotic arm. In theory, this is a very promising concept, but it has several disadvantages: It often needs a lot of computation time to find the trajectory, so the interaction with the robot will take a lot of time. Another problem occurs if the robot grasps an object: The motion planner needs the exact shape and position of all objects in the environment to calculate a trajectory with no collisions between the object and the environment. The shape detection is very difficult with the used hardware setup in this structured environment, so the disadvantages outweigh and no motion planners were used.
Another option is the training of simple trajectories the robot should follow. This movement type also is simply trainable and generates a reliable movement, but it not supports generalization at all. Because of this, it cannot be used in an unstructured environment like in this project.
The big advantage of GoTo commands is that the training and the generalization to new object positions is very simple. Collision avoidance is, however, more difficult and unintuitive because the user has to define manually the right via-positions. This problem is avoided by using ProMPs, but for them the training is more complicated and takes more time.

6 Conclusion & Future Work

The focus of this thesis was the development of intuitive processes to make Imitation Learning of whole tasks possible for everyone, even for users without a robot or computer science background. After developing these methods they were evaluated with a user study.

6.1 Conclusion

The basis of this framework is the newly designed high-level task structure. It is using *Task*, *Skills* and *Actions* to describe the complete behavior. It allows the creation of new robot tasks without using any programming language. Instead, it is only necessary to define and select the right skills. A skill is a small step of the robot behavior and the goal is easily understandable for humans. A skill can have several *Actions*, which are predefined commands with parameters that can be executed directly on the robot. The *Action* used in this project are *GoToCart*, *GoToJoint*, *RunProMP*, *MoveGripper* and *Grasp*.
The task structure allows the reuse of Skills in other Tasks, so the training of new tasks becomes easier with a growing number of skills.
To make the definition of new tasks and skills as easy as possible a teaching process was developed to guide the user through the creation of new Tasks and Skills and the training of the used Actions. This is done by giving direct instruction to what the user has to do and asking easily understandable questions to set the Action parameters. The whole process of switching robot control modes between teaching- joint-space- and task-space-mode is done automatically, so the user has not to care about this.
In addition to the general teaching process, special training processes were designed to make the ProMP training possible for inexperienced users. During this teaching process, the user only has to move the robot in the kinesthetic teaching mode to give the demonstrations, the complete data processing is done automatically.
It is possible to train ProMPs for one or both arms or use a special Coordinated ProMP teaching process. A Simple ProMP is a normal ProMP for one arm which can be trained by moving the arm four times on the desired trajectory and varying the end position. Between the demonstrations, the robot arm moves back to the start position automated. If both arms should be used the Coordinated ProMP can be used. It allows a synchronized training of both arms. At first, one arm is trained alone, when this movement is replayed and the user can train the second arm and synchronize the movement to the first arm. If the training of both arms independent is not possible, the Simple ProMP process can also be used in a similar version for both arms.
The whole teaching process for the task structure and the ProMP training was implemented with a graphical user interface, so it can be controlled by every user on his own. Besides, a low-level motion generator was developed which calculates trajectories from via points and sends them to the robot. It also can switch the different control modes automatically.
Another important outcome of this thesis is the result of the applied user study which evaluates the success and acceptance of the designed training framework with inexperienced users. The first important point is that all users understand the teaching process and were able to teach new tasks to the robot. Most users also think that the process is comprehensible. They all were able to define new tasks and skills on their own, but some users do not use the task structure in the recommended way and define a lot too many skills.
The users also understood how GoTo commands and ProMPs are working and what their difference is. In the application of GoTo commands the definition of via-point often was forgotten which lead to a collision between the robot and its environment. Yet the users think that training ProMPs is easy and that the robot behavior for GoTos is predictable. But it seems like the users predict the behavior wrong which often results into collisions.
All users were able to train new ProMPs, even if this is much more complicated. The users understand the higher training effort and have not complained about it. Instead, more than half of all users even prefer ProMPs over GoTos. The generalization capabilities of the ProMPs also were good enough to execute them with different object positions, but sometimes the users simply forgot to move the object to new positions. This can be avoided by a mechanism that remembers the user if they have not moved the object. All in all the success rate in ProMP training was higher than for GoTos, so maybe this kind of movement should be preferred in the future.
Another important finding is that all users can use the kinesthetic teaching mode without problems. Most users also agree that moving the robot to the desired position is easy. So kinesthetic teaching seems to be the right decision for the generation of demonstrations.

6.2 Future Work

The first point that should be considered is the usage of GoTo commands or trajectory training methods like ProMPs. Because GoTos often failed in the user study, using ProMPs seems to be a good approach, but using them for all robotic movements is too costly. A good compromise would be the usage of ProMPs for all movements related to an object and using GoTos for all movements to fixed positions. So the user only has to decide, if the robot should move to an object or a fixed position.
If ProMPs should be used in a real application, some further improvements must be done: In the training process, it is necessary to ensure that the user gives demonstrations with different object positions. This can be easily done by comparing the object positions before starting the demonstration recording. During the training process, it happened regularly that users gave demonstrations that are not similar enough to each other. A common example was that the last joint of the arm was rotated in different directions, but the end-position was the same. The ProMP calculated from these demonstrations will, unavoidably, results in undefined behavior, so this error must be avoided automatically. To do this an algorithm is needed that finds demonstrations that are not similar enough and deletes them from the training data. It should be possible to do this using cross-validation and evaluating if the demonstration is in the 2σ-surrounding of the other demonstrations.
A second important improvement is to check if the point, the ProMP should be conditioned on, is in the trained area, before the ProMP is executed. If ProMPs are conditioned on points outside the trained area, they often show an unpredictable behavior and this must not happen in a real-world application. A first idea to do this is a check if the point is between all demonstrations, or in the 2σ-surrounding.
Another challenge is the execution of both arm movements. In this project, they only were coupled by probabilistic methods and no really strict rules were used. Because this work focuses on the training process itself and not on the execution, this was not a problem, even if the execution often was far from optimal. So it will be necessary to use more advanced representations for both arm movements using force[62, 47] or relative position for the coupling[48, 49].
The reliability of the whole system could be improved a lot by using the task-space controller which can detect and avoid the joint limit automatically. This is often a problem during both the execution of task-space trajectories and the recording of demonstrations. This task can be done by different motion planners, but they often need too much computation time for real-time applications, so this is still a big challenge. An assisted teaching controller similar to [24] with automated joint limit avoidance would improve the generation of demonstrations with kinesthetic teaching a lot.

Bibliography

[1] B. Gates, "A robot in every home," *Scientific American*, vol. 296, no. 1, pp. 58–65, 2007.

[2] M. Ford, *Rise of the Robots: Technology and the Threat of a Jobless Future*. Basic Books, 2015.

[3] S. J. Russell, "Learning agents for uncertain environments," in *COLT*, vol. 98, pp. 101–103, 1998.

[4] S. Schaal, "Is imitation learning the route to humanoid robots?," *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[5] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in neural information processing systems*, pp. 2616–2624, 2013.

[6] 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.

[7] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The development of honda humanoid robot," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*, vol. 2, pp. 1321–1326, IEEE, 1998.

[8] M. A. Diftler, J. Mehling, M. E. Abdallah, N. A. Radford, L. B. Bridgwater, A. M. Sanders, R. S. Askew, D. M. Linn, J. D. Yamokoski, F. Permenter, *et al.*, "Robonaut 2-the first humanoid robot in space," in *2011 IEEE international conference on robotics and automation*, pp. 2178–2183, IEEE, 2011.

[9] "Franka emika." `https://www.franka.de/`, 2019. [Online; accessed 26. November 2019].

[10] 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.

[11] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, pp. 305–313, 1989.

[12] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[13] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in neural information processing systems*, pp. 1547–1554, 2003.

[14] M. N. Nicolescu and M. J. Mataric, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 241–248, ACM, 2003.

[15] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 131–157, 2015.

[16] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 2, pp. 286–298, 2007.

[17] A. Perzylo, N. Somani, S. Profanter, I. Kessler, M. Rickert, and A. Knoll, "Intuitive instruction of industrial robots: Semantic process descriptions for small lot production," in *2016 ieee/rsj international conference on intelligent robots and systems (iros)*, pp. 2293–2300, IEEE, 2016.

[18] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso, "Interactive robot task training through dialog and demonstration," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pp. 49–56, ACM, 2007.

[19] S. Ekvall and D. Kragic, "Robot learning from demonstration: a task-level planning approach," *International Journal of Advanced Robotic Systems*, vol. 5, no. 3, p. 33, 2008.

[20] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "Costar: Instructing collaborative robots with behavior trees and vision," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 564–571, IEEE, 2017.

[21] C. Paxton, F. Jonathan, A. Hundt, B. Mutlu, and G. D. Hager, "User experience of the costar system for instruction of collaborative robots," *arXiv preprint arXiv:1703.07890*, 2017.

[22] A. Weiss, J. Igelsbock, S. Calinon, A. Billard, and M. Tscheligi, "Teaching a humanoid: A user study on learning by demonstration with hoap-3," in *RO-MAN 2009-The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pp. 147–152, IEEE, 2009.

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

[24] S. Wrede, C. Emmerich, R. Grünberg, A. Nordmann, A. Swadzba, and J. Steil, "A user study on kinesthetic teaching of redundant robots in task and configuration space," *Journal of Human-Robot Interaction*, vol. 2, no. 1, pp. 56–81, 2013.

[25] M. Tykal, A. Montebelli, and V. Kyrki, "Incrementally assisted kinesthetic teaching for programming by demonstration," in *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, pp. 205–212, IEEE Press, 2016.

[26] H. Lee, J. Kim, and T. Kim, "A robot teaching framework for a redundant dual arm manipulator with teleoperation from exoskeleton motion data," in *14th IEEE-RAS International Conference on Humanoid Robots (Humanoids), 2014*, (Piscataway, NJ), pp. 1057–1062, IEEE, 2014.

[27] M. Hessinger, A. Buchta, R. Werthschützky, and M. Kupnik, "Imu-based motion capture system for real-time body joint angle measurement," in *Annual Meeting of the German Society of Biomedical Engineering and Joint Conference in Medical Physics*, Oktober 2017.

[28] M. Laghi, M. Maimeri, M. Marchand, C. Leparoux, M. Catalano, A. Ajoudani, and A. Bicchi, "Shared-autonomy control for intuitive bimanual tele-manipulation," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pp. 1–9, IEEE, 2018.

[29] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8, IEEE, 2018.

[30] R. Caccavale, M. Saveriano, G. A. Fontanelli, F. Ficuciello, D. Lee, and A. Finzi, "Imitation learning and attentional supervision of dual-arm structured tasks," in *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pp. 66–71, IEEE, 2017.

[31] N. Figueroa, A. L. P. Ureche, and A. Billard, "Learning complex sequential tasks from demonstration: A pizza dough rolling case study," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 611–612, Ieee, 2016.

[32] J. Silvério, L. Rozo, S. Calinon, and D. G. Caldwell, "Learning bimanual end-effector poses from demonstrations using task-parameterized dynamical systems," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 464–470, IEEE, 2015.

[33] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[34] J. Kober, K. Mülling, O. Krömer, C. H. Lampert, B. Schölkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *2010 IEEE International Conference on Robotics and Automation*, pp. 853–858, IEEE, 2010.

[35] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*. Springer London, 2000.

[36] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[37] P. Kormushev, S. Calinon, and D. G. Caldwell, "Robot motor skill coordination with em-based reinforcement learning," in *2010 IEEE/RSJ international conference on intelligent robots and systems*, pp. 3232–3237, IEEE, 2010.

[38] A. Paraschos, G. Neumann, and J. Peters, "A probabilistic approach to robot trajectory generation," in *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pp. 477–483, IEEE, 2013.

[39] G. Maeda, M. Ewerton, R. Lioutikov, H. B. Amor, J. Peters, and G. Neumann, "Learning interaction for collaborative tasks with probabilistic movement primitives," in *2014 IEEE-RAS International Conference on Humanoid Robots*, pp. 527–534, IEEE, 2014.

[40] D. Koert, S. Trick, M. Ewerton, M. Lutter, and J. Peters, "Online learning of an open-ended skill library for collaborative tasks," in *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pp. 1–9, IEEE, 2018.

[41] D. Koert, J. Pajarinen, A. Schotschneider, S. Trick, C. Rothkopf, and J. Peters, "Learning intention aware online adaptation of movement primitives," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3719–3726, 2019.

[42] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning to sequence movement primitives from demonstrations," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4414–4421, IEEE, 2014.

[43] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations," *Robotics and Autonomous Systems*, vol. 74, pp. 97–107, 2015.

[44] L. D. Rozo, S. Calinon, D. Caldwell, P. Jiménez, and C. Torras, "Learning collaborative impedance-based robot behaviors," in *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[45] Y. Huang, L. Rozo, J. Silvério, and D. G. Caldwell, "Kernelized movement primitives," *The International Journal of Robotics Research*, vol. 38, no. 7, pp. 833–852, 2019.

[46] R. Zollner, T. Asfour, and R. Dillmann, "Programming by demonstration: Dual-arm manipulation tasks for humanoid robots," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 1, pp. 479–484, IEEE, 2004.

[47] A. Gams, B. Nemec, A. J. Ijspeert, and A. Ude, "Coupling movement primitives: Interaction with the environment and bimanual tasks," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 816–830, 2014.

[48] J. Umlauft, D. Sieber, and S. Hirche, "Dynamic movement primitives for cooperative manipulation and synchronized motions," in *IEEE International Conference on Robotics and Automation (ICRA), 2014*, (Piscataway, NJ), pp. 766–771, IEEE, 2014.

[49] A. Batinica, B. Nemec, A. Ude, M. Rakovic, and A. Gams, "Compliant movement primitives in a bimanual setting," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, (Piscataway, NJ), pp. 365–371, IEEE, 2017.

[50] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive motion of animals and machines*, pp. 261–280, Springer, 2006.

[51] F. Rovida, B. Grossmann, and V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6793–6800, IEEE, 2017.

[52] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and Al: An Introduction*. CRC Press, 2018.

[53] D. Faconti, "Behaviortree.cpp." `https://www.behaviortree.dev/`, 2019. [Online; accessed 08. November 2019].

[54] S. Gomez-Gonzalez, G. Neumann, B. Schölkopf, and J. Peters, "Adaptation and robust learning of probabilistic movement primitives," *IEEE Transactions on Robotics*, 2018.

[55] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[56] H. Sadeghian, L. Villani, M. Keshmiri, and B. Siciliano, "Task-space control of robot manipulators with null-space compliance," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 493–506, 2013.

[57] A. Dietrich, C. Ott, and A. Albu-Schäffer, "An overview of null space projections for redundant, torque-controlled robots," *The International Journal of Robotics Research*, vol. 34, no. 11, pp. 1385–1400, 2015.

[58] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2016.

[59] "Ros." `https://www.ros.org/about-ros/`, 2019. [Online; accessed 08. November 2019].

[60] W. Meeussen, "Ros control." `http://wiki.ros.org/ros_control`, 2019. [Online; accessed 08. November 2019].

[61] S. G. Hart and L. E. Staveland, "Development of nasa-tlx (task load index): Results of empirical and theoretical research," in *Advances in psychology*, vol. 52, pp. 139–183, Elsevier, 1988.

[62] E. Pignat and S. Calinon, "Bayesian Gaussian mixture model for robotic policy imitation," *IEEE Robotics and Automation Letters (RA-L)*, vol. 4, no. 4, pp. 4452–4458, 2019.

A User Study

A.1 User Background

Bitte füllen Sie folgende Angaben aus:

Geschlecht:

Mann ☐ Frau ☐ divers ☐

Alter:

☐ 17 oder jünger
☐ 18-20
☐ 21-29
☐ 30-39
☐ 40-49
☐ 50-59
☐ 60 oder älter

Was ist Ihr höchster Bildungsabschluss?

☐ kein Schulabschluss
☐ Grund-/Hauptschulabschluss
☐ Realschule (mittlere Reife)
☐ Gymnasium/Fachoberschule (Abitur)
☐ anderer höchster Bildungsabschluss:
☐ abgeschlossene Ausbildung
☐ Bachelor (Fachhochschule oder Hochschule)
☐ Master oder Diplom (Fachhochschule/Hochschule)
☐ Promotion

In welchem Bereich arbeiten/studieren Sie:

Wie viel Erfahrung im Umgang mit Programmiersprachen haben Sie?

- [ ] Keine Erfahrung
- [ ] Grundkenntnisse
- [ ] Gelegentliche Anwendung (mindestens monatlich)
- [ ] Regelmäßige Anwendung (mindestens wöchentlich)
- [ ] Häufige Anwendung (nahezu täglich)

Wie viel Erfahrung im Umgang mit Robotern, die dem Roboter im Versuch ähnlich sind, haben Sie?

- [ ] Keine Erfahrung
- [ ] Ich habe bereits Roboter in Aktion gesehen
- [ ] Ich habe aktiv einzelne Aktionen mit Robotern durchgeführt
- [ ] Ich habe Grundkenntnisse über Aufbau und Funktionsweise von Robotern
- [ ] Ich bin in der Lage Roboter zu programmieren und kenne den Aufbau eines Roboters

A.2 Experiment Description

In dieser Studie besteht Ihre Aufgabe darin, dem Roboter neue Aufgaben beizubringen, die an einfache Haushaltsarbeiten angelehnt sind. Diese Aufgaben werden nach einer bestimmten Struktur aufgebaut, die im Folgenden kurz erklärt wird:

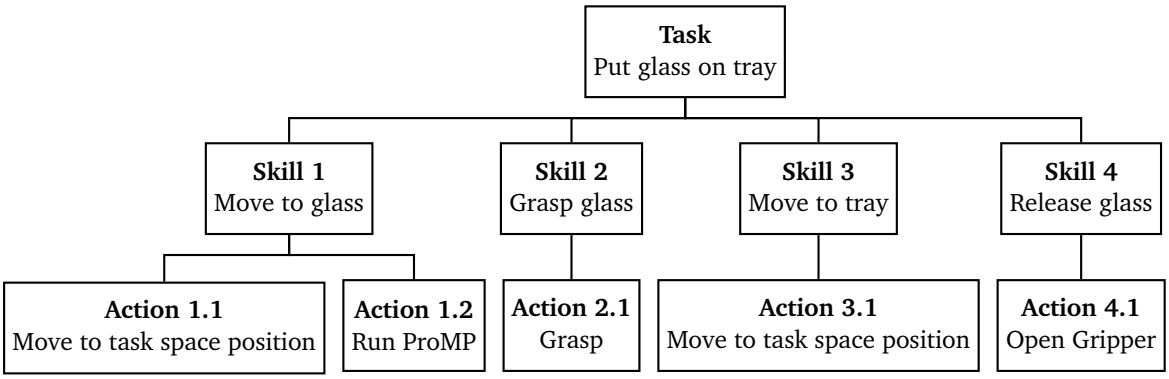


Figure A.1.: Aufbau einer Roboter Aufgabe

Abbildung A.1 zeigt die Beispielhafte Aufgabe: *Stelle ein Glas auf das Tablett*. Diese Aufgabe heißt in der verwendeten Struktur ab jetzt *Task*. Da diese Aufgabe aus mehreren kleinen Schritten besteht, kann auch ein Task aus mehreren Bausteinen bestehen. Diese heißen *Skills*. In der Beispielaufgabe sind mögliche Skills zum Beispiel *Greife das Glas* und *Bewege das Glas zum Tablett*.
Die Skills können von Ihnen erzeugt werden, indem sie eine oder mehrere Aktionen auswählen, die der Roboter in einem Skill ausführen soll. Eine Aktion bringen sie dem Roboter bei, indem sie diese „Trainieren“ und dem Roboter so zeigen, wie er die Aktion ausführen soll. Die dafür notwendigen Schritte werden Ihnen später vom Programm vorgegeben. Welche Aktionen Ihnen zur Verfügung stehen, wird im weiteren Versuchsablauf erläutert.
Ihre Aufgabe besteht darin verschiedene Tasks zu definieren, die unterschiedliche Aufgaben erfüllen sollen. Dafür können Sie beliebig viele Skills erzeugen, die sie dann in jedem Task wiederverwenden können.

A.2.1 Vorbereitung

Zur Vorbereitung der Experimente können Sie die Handhabung des Roboters üben. In den folgenden Experimenten werden Sie den Roboter manuell in die gewünschte Position bewegen, um ihm so das Anfahren einer neuen Position beizubringen. Um diesen Vorgang zu testen und sich an die Bewegungsmöglichkeiten des Roboters zu gewöhnen, kann der Versuchsleiter den Roboter in einen Trainingsmodus schalten in dem er sich sehr leicht bewegen lässt. Testen Sie dabei vor allem die Reichweite des Roboters und beobachten Sie welche Gelenke Sie bevorzugt bewegen.

A.2.2 Experiment 1

In diesem Experiment bringen Sie dem Roboter bei eine Kekspackung zu greifen und an eine neue Position zu legen. Der gewünschte Ablauf ist folgender:

- Der Roboter greift die Kekspackung mit dem linken Arm
- Der Roboter übergibt die Kekse an den rechten Arm
- Der Roboter stellt die Kekse auf das Tablett
- Am Ende sind beide Arme wieder in der Startposition

Für die Durchführung dieser Aufgabe können sie die Aktionen *Greifen*, *Bewege zu fester Position* und *Bewege zu Objekt* verwenden. Diese werden im folgenden kurz erklärt.
Die **Greifen** Aktion schließt den Greifer und greift so ein Objekt. Anschließend wird das Objekt festgehalten.
Der Roboter ist in der Lage, die Position von Objekten zu erkennen, deshalb kann er auch seine Bewegungen an Objekten ausrichten. Dazu können Sie die **Bewege zu Objekt** Action verwenden. Diese bewegt den Greifer des Roboters an eine Position, die sie vorgeben können und sich anschließend immer an der Objektposition orientiert. Dazu wählen Sie als erstes das Objekt (In diesem Versuch das Tablett oder die Kekspackung) aus, zu dem sich der Roboter bewegen soll. Anschließend können Sie den Roboter mit Hand an die gewünschte Position bewegen.
Wenn der Roboter nicht mit einem Objekt interagieren soll, können Sie die **Bewege zu fester Position** Aktion verwenden. Diese bewegt den Roboterarm nach dem Training immer an die genau gleiche Position und orientiert sich nicht an Objekten.
Für einen erfolgreichen Trainingsprozess beachten sie folgende Hinweise:

- Die **Bewege zu...** Aktionen bewegen den Roboter immer auf direktem Weg zur Zielposition, es ist oft sinnvoll einige Zwischenschritte zu definieren, damit der Roboter einem sinnvollem Weg folgt
  - Wenn der Roboter ein Objekt greifen soll, ist es sinnvoll Ihn erst in die Nähe des Objekts zu fahren und erst im zweiten Schritt direkt zum Objekt.
- Der Ablauf wird oft dadurch erleichtert, dass Sie den Roboter zwischen zwei Schritten wieder zurück in die Startposition fahren. Dazu können Sie den **goto_home** Skill verwenden.
- Zum Öffnen der Greifer können Sie den **open_gripper** Skill verwenden.

Der **Trainingsprozess** funktioniert wie folgt:

- Erzeugen Sie einen neuen Task mit dem *Neuer Task* Knopf
- Jetzt können Sie dem Task neue Skills hinzufügen. Dazu können Sie einen vorhandenen Skill auswählen und mit dem *Ausführen* Knopf direkt auf dem Roboter ausführen lassen. Mit dem *Hinzufügen* Knopf fügen Sie den Skill zum Task hinzu. Alle Skills in einem Task werden Ihnen im linken unteren Feld angezeigt.
- Immer wenn der Roboter etwas neues lernen soll, müssen Sie einen neuen Skill erzeugen. Dazu verwenden Sie den *Neuer Skill* Knopf. Anschließend können Sie eine Aktion auswählen und diese mit dem *Trainieren* Knopf trainieren. Nach dem Training können sie die Aktion ausführen und dem Skill hinzufügen.
- Wenn Sie dem Skill alle gewünschten Aktionen hinzugefügt haben, können Sie ihn speichern und er wird automatisch zum Task hinzugefügt.

A.2.3 Experiment 2

In diesem Experiment sollen Sie dem Roboter noch einmal die gleiche Aufgabe wie in Experiment 1 beibringen. Dazu sollen Sie jetzt aber statt der *Bewege zu ...* Aktionen die *Bewegung ausführen* Aktion verwenden.
Mit der **Bewegung ausführen** Aktion können sie den Roboter genau einem Weg folgen lassen, den sie vorher vorgeben. Dies erleichtert das Umfahren von Hindernissen oder das Anfahren von Objekten aus einer bestimmten Richtung.
Um die *Folge Trajektorie* Aktion zu trainieren müssen sie dem Roboter die Bewegung vier mal demonstrieren. Auch diese Bewegungen orientieren sich an einem Objekt, deshalb müssen Sie dieses als erstes auswählen. Zwischen den verschiedenen Demonstrationen werden sie aufgefordert das Objekt an eine neue Position zu legen, so kann der Roboter lernen, eine ähnliche Bewegung für verschiedene Postionen des Objekts auszuführen.
Das Training einer neuen Bewegung funktioniert wie folgt:

- Wenn Sie einen neuen Skill erzeugen, können Sie die *Bewegung ausführen* Aktion verwenden. Bevor Sie diese trainieren, müssen sie mit dem *Neue einfache Bewegung* Knopf eine neue Bewegung aufnehmen.
- Als erstes können Sie den Roboterarm in eine Startposition bewegen oder dafür einfach die aktuelle Position verwenden
- Anschließend werden sie aufgefordert den Roboterarm auf dem gewünschtem Weg zu bewegen
- Danach bewegt sich der Arm automatisch wieder in die Startposition und sie können die eben erzeugte Demonstration verwenden oder verwerfen, falls Sie eine bessere Demonstration erzeugen möchten
- Dieser Prozess wird vier mal wiederholt
- Anschließend müssen Sie die *Bewegung ausführen* Aktion noch trainieren und können Sie Ausführen und dem Skill hinzufügen.

A.2.4 Experiment 3

Das Dritte Experiment ist etwas aufwändiger, dabei soll der Roboter die Kekse auf das Tablett legen und dieses anschließend anheben. Der gewünschte Ablauf ist der Folgende:

- Der Roboter greift das Tablett und stellt es vor sich
- Der Roboter legt die Kekse auf das Tablett
- Der Roboter greift das Tablett mit beiden Armen gleichzeitig und hebt es anschließend an

Zur Durchführung dieser Aufgabe können Sie alle in Experiment 1 und 2 verwendeten Skills und Actions verwenden.
Zusätzlich können Sie die **Bewegung ausführen** Aktion jetzt auch mit zwei Armen verwenden. Dies ist notwendig, wenn der Roboter beide Arme gleichzeitig bewegen soll. Im **koordinierten Modus** trainieren Sie erst einen Arm einzeln, anschießend wird die Bewegung auf diesem Arm ausgeführt und sie trainieren den zweiten Arm. Der Roboter ist danach in der Lage beide Roboterarme gleichzeitig zu bewegen. Diesen Modus können Sie zum Beispiel verwenden um das Tablett zu greifen. Der Training für diesen Bewegungstyp funktioniert genauso wie in Experiment 2, Sie müssen nur *Neue koordinierte Bewegung* statt *Neue einfache Bewegung* auswählen.
Wenn es nicht möglich ist beide Arme nacheinander zu trainieren, können Sie die neue Bewegung auch trainieren, indem Sie beide Roboterarme gleichzeitig bewegen. Dazu wählen Sie neue *Neue einfache Bewegung* und anschließend *Beide Arme* aus. Diese Aktion bietet sich immer dann an, wenn beide Arme verbunden sind, zum Beispiel wenn sie das gleiche Objekt greifen.
Während dem Training bewegen Sie beide Arme zusammen auf dem gewünschtem Weg und müssen diese anschließend auch wieder manuell in die gewünschte Startposition bewegen. Achten sie dabei besonders darauf, dass der Roboter das Objekt weiter festhält und nicht verliert.

A.3 Questionnaire 1

Bitte geben Sie im Folgenden an in wieweit die jeweiligen Aussagen auf Sie zutreffen.
Hierbei können Sie Abstufungen zwischen „trifft nicht zu“ und „trifft zu“ auswählen. Sollte eine Aussage nicht zutreffen, tragen Sie bei Anmerkungen den Grund dafür ein.
Die Aussagen beziehen sich hierbei nur auf das direkt vorangegangene Experiment.

Der Ablauf des Trainingsvorgangs ist nachvollziehbar.

☐ trifft nicht zu ☐ trifft eher nicht zu ☐ teils-teils ☐ trifft eher zu ☐ trifft zu

Ich habe mich bei der Bedienung des Roboters unwohl gefühlt.

☐ trifft nicht zu ☐ trifft eher nicht zu ☐ teils-teils ☐ trifft eher zu ☐ trifft zu

Das Verhalten des Roboters ist vorhersehbar.

☐ trifft nicht zu ☐ trifft eher nicht zu ☐ teils-teils ☐ trifft eher zu ☐ trifft zu

Die Bewegung des Roboters an die gewünschte Position fällt mir leicht.

☐ trifft nicht zu ☐ trifft eher nicht zu ☐ teils-teils ☐ trifft eher zu ☐ trifft zu

Anmerkungen:

A.4 Questionnaire 2

Bitte geben Sie im Folgenden an, inwieweit die jeweiligen Aussagen auf Sie zutreffen.
Die Aussagen beziehen sich auf die zwei verschiedenen Roboterbewegungstypen(Bewegungen zu einem Punkt und das Folgen einer festgelegten Trajektorie), die Sie in den vorangegangenen drei Experimentensets kennengelernt haben.

Bitte geben Sie an, für welchen Bewegungstyp Ihnen das Training am **einfachsten** erscheint:

- Ich finde das Training des Bewegungstypen in Teil 1 am einfachsten
- Ich finde das Training des Bewegungstypen in Teil 2 am einfachsten
- nichts davon, sondern:

Bitte geben Sie an, für welchen Roboter-Bewegungstyp Sie das Training im Vergleich am **intuitivsten** empfunden haben:

- Ich finde das Training des Bewegungstypen in Teil 1 am intuitivsten
- Ich finde das Training des Bewegungstypen in Teil 2 am intuitivsten
- nichts davon, sondern:

Bitte geben Sie an, welcher Bewegungstyp Ihnen in der Anwendung am **nützlichsten** erscheint:

- Ich finde das Training des Bewegungstypen in Teil 1 am nützlichsten
- Ich finde das Training des Bewegungstypen in Teil 2 am nützlichsten
- nichts davon, sondern:

Anmerkungen: