Technische Universität München
TUM School of Computation, Information and Technology

Intuitive Robot Programming with Force-based Skills and Task Structures

Thomas Eiband

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:
Prof. Dr. Reinhard Heckel

Prüfende der Dissertation:
1. Prof. Dr. Dongheui Lee
2. Prof. Dr. Jochen Steil
3. Prof. Dr.-Ing. Sami Haddadin

Die Dissertation wurde am 27.09.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 09.04.2024 angenommen.

Abstract

Robot programming is often still reserved for experts because it requires knowledge of coding, kinematics, and dynamics. This prevents novice users such as production workers from easily using robots and programming them for new tasks. However, fast reconfigurability is required to increase flexibility in production planning, allowing automation for small lot sizes and coping with market uncertainties. Therefore, intuitive programming solutions are needed to revolutionize how we program robots today. Programming interfaces for collaborative robots have already moved towards the concept of robot skills, which are reusable building blocks with a clear meaning to the human operator. However, such building blocks are often constrained to manipulation actions such as pick and place and do not tackle contact-rich problems requiring interaction forces. This thesis introduces the concept of contact skills, which implement robot capabilities to interact with the human and the environment. Intuitive programming interfaces then use these contact skills, allowing more sophisticated robot behaviors by incorporating the force-sensing capabilities of the robot. The main contribution of this thesis is to simplify robot programming for end-users and enable robots to learn from human demonstrations as an intuitive way of transporting knowledge between humans and robots. Kinesthetic teaching is the primary input modality to intuitively program single skills, sequences of skills, and multiple solutions for task variants. Further, interactive programming schemes allow the teaching of robotic decision-making and recovery behaviors by generating robot programs as task graphs. The presented methods segment and classify actions from human demonstrations, termed skill recognition. This approach allows incrementally building and refining task representations that display the robot's knowledge in a human-understandable form. The identified skills can be automatically parameterized and adaptively sequenced to be executed on the robot. The proposed methods in this thesis are validated in real robotics experiments and various user studies. The results show that the proposed contact skills are interpretable by both humans and robots and can be recognized online during kinesthetic teaching while the user receives immediate feedback. Finally, a unified framework is proposed that combines the automatic skill recognition and parameterization approach with existing manual programming methods. An ontology serves as central element to bundle the knowledge.

Zusammenfassung

Die Programmierung von Robotern ist häufig noch Experten vorbehalten, da sie Kenntnisse über Codierung, Kinematik und Dynamik erfordert. Dies hindert unerfahrene Benutzer daran, Roboter für eine neue Aufgabe zu programmieren um die Flexibilität in der Produktionsplanung zu erhöhen, die Automatisierung für kleine Losgrößen zu ermöglichen und mit Marktunsicherheiten umzugehen. Daher sind intuitive Programmierlösungen eine Möglichkeit, die Art und Weise der Roboterprogrammierung zu revolutionieren. Programmierschnittstellen für kollaborierende Roboter benutzen bereits sogenannte Roboterfähigkeiten, die wiederverwendbare Bausteine darstellen und für den menschlichen Bediener klar verständlich sind. Diese Bausteine beschränken sich jedoch oft auf die Manipulation von Objekten wie das Aufnehmen und Ablegen von Gegenständen und gehen nicht auf kontaktreiche Probleme ein, die Interaktionskräfte erfordern. In dieser Arbeit wird das Konzept der Kontaktfähigkeiten eingeführt, die Roboterverhalten zur Interaktion mit dem Menschen und der Umgebung implementieren. Diese Kontaktfähigkeiten werden dann in intuitiven Programmierschnittstellen verwendet, die sowohl den Bewegungs- als auch den Kraftbereich einbeziehen und dadurch anspruchsvollere Verhaltensweisen ermöglichen. Es werden neuartige Algorithmen für das Erlernen kraftbasierter Roboterfähigkeiten vorgeschlagen, die für Systeme mit eingeschränkter visueller Wahrnehmung von entscheidender Bedeutung sind. Der Hauptbeitrag dieser Arbeit ist die Vereinfachung der Roboterprogrammierung für den Endbenutzer mit Hilfe von menschlichen Demonstrationen um auf intuitive Weise das Wissen vom Menschen auf den Roboter zu übertragen. Intuitives Programmieren wird vor allem durch das Vormachen von einzelnen Fähigkeiten oder Sequenzen von Fähigkeiten erreicht. Darüber hinaus ermöglichen interaktive Programmierverfahren das Erlernen von Entscheidungs- und Fehlerbehebungsverhalten des Roboters durch die Generierung von verzweigten Ablaufplänen. Die vorgestellten Methoden segmentieren und klassifizieren Aktionen aus menschlichen Demonstrationen, was als Fähigkeitserkennung bezeichnet wird. Dies ermöglicht den schrittweisen Aufbau und die Verfeinerung von Ablaufplänen, die das Wissen des Roboters in einer für den Menschen verständlichen Form darstellen. Darüber hinaus können die identifizierten Fähigkeiten automatisch parametrisiert und bei der Ausführung adaptiv sequenziert werden. Die vorgeschlagenen Methoden werden in realen Roboterexperimenten und verschiedenen Benutzerstudien validiert. Die Ergebnisse zeigen, dass die vorgeschlagenen Kontaktfähigkeiten sowohl von Menschen als auch von Robotern interpretiert werden können. Sie können während der Demonstrationsphase erkannt werden, wobei der Benutzer sofortiges Feedback erhält. Schließlich wird ein einheitlicher Ansatz vorgeschlagen, der die automatische Erkennung und Parametrisierung von Fähigkeiten mit bestehenden manuellen Programmiermethoden kombiniert.

Dabei wird eine Ontologie als zentrales Element zur Bündelung des Wissens verwendet.

Contents

List of Figures

List of Tables

Acronyms

BP-HMM	Beta Process Autoregressive HMM
CIP	Collaborative Incremental Programming
CPU	Central Processing Unit
DMP	Dynamic Movement Primitive
DOF	Degrees of Freedom
DTW	Dynamic Time Warping
FoF	Factory of the Future
FSM	Finite State Machine
FTS	force-torque sensor
GMM	Gaussian Mixture Model
GMR	Gaussian Mixture Regression
GUI	Graphical User Interface
HFI	Human Factory Interface
HMM	Hidden Markov Model
LfD	Learning from Demonstration
LLM	Large Language Model
MDP	Markov Decision Process
MLP	Multi-layer Perceptron
NN	Neural Network
OSP	offline skill-based programming
PbD	Programming by Demonstration
PDDL	Planning Domain Definition Language
POMDP	Partially Observable Markov Decision Process
PRM	Probabilistic Roadmap
RBF	Radial Basis Function

RRT	Rapidly Exploring Random Tree
SBP	Sequential Batch Programming
SP	solution pool
SVM	Support Vector Machine
TCP	tool center point
TRL	Technology Readiness Level
UIP	User-triggered Incremental Programming

1 Introduction

The first known tool usage by the genus Homo, an ancient of the Homo Sapiens, is dated back 2.6 million years ago when large quantities of stone tools were found in Tanzania [1]. Through millennia of human tool usage, there must have been a social form of knowledge sharing, which improved the tools and how we use them. Social learning theory describes one of these methods as observational learning, where simply one subject learns by observing the other [2]. Robotics researchers have long tried to employ this principle of human knowledge transfer in the robot learning domain. Here, a well-known concept is Learning from Demonstration (LfD), shown in early works that use teleoperation [3, 4], visual observation of the human [5], or kinesthetic teaching [6]. Similar to LfD, PbD is a well-known concept that describes how an end-user programs a robot by demonstrating the required strategy or behavior. The research community currently aims to use the concept of PbD to simplify robot programming for inexperienced users.

Another important concept of robot programming uses so-called robot skills, which implement a robot capability that solves a specific problem in the physical world. This concept has two advantages. First, it enables the reuse of behaviors among different tasks by parameterizing these skills. Second, it allows the representation of robot knowledge in a legible way to the end-user.

Given these two enablers, PbD and robot skills, they can be combined to develop new learning algorithms. Skill learning from demonstration has been shown in various applications using state variables such as position, orientation, velocities, and kinematics. These are located under the term motion domain. However, since a human naturally exploits forces and tactile feedback, contact-based interactions deserve more attention in today's research landscape. The state variables of force and torque are located under the term force domain.

This thesis proposes new algorithms and frameworks to recognize and learn robotic skills from human demonstrations, putting a major focus on the force domain beside the motion domain. Such force-based skills are defined as contact skills in this thesis. They are required by various physical actions that both humans and robots can perform individually. Exemplarily, a human uses the sense of touch, also known as tactile feedback, for manipulation tasks as a natural enhancement besides the visual and proprioceptive input. The tactile feedback becomes even more critical if the visual perception is impaired, e.g., due to poor lighting conditions or occlusion. Similarly, a robot can be equipped with force sensing, visual perception, and proprioception and then be employed in manipulation tasks. Analogously to the human scenario, the vision of the robot could be impaired, which raises the importance of force sensing.

The previous example illustrated the importance of incorporating the force domain into the learning problem of robot skills. Furthermore, this thesis proposes that the knowledge transfer from a human to a robot skill is achieved in two steps:

1. Recognition of the robot skill from a human demonstration.

2. Parameterization of the robot skill by automatic parameter extraction from the human demonstration.

The presented approaches in this thesis allow an inexperienced user to program complete robot tasks from scratch, summarized as task definition. In the following, possible task definition methods are introduced.

1.1 Overview of Task Definition Methods

Definition 1 *A task is governed by fixed goals that are reached with a parameterized behavior of the robot.*

While a task can be adaptive, e.g., picking object A from varying locations, its goal and parameters are always fixed, e.g., placing the picked object A on top of object B.

This thesis focuses on a scenario where a single user programs a robot by demonstration to solve a given task. This process is termed task definition and consists of three phases:

1. analysis of inputs, i.e., required resources, robot capabilities, and task goals;
2. human selection of a task definition method considering the user's level of experience; and
3. usage of a task definition method to create a task representation.

Fig.1.1 gives an overview of established task definition workflows that connect different inputs, task definition methods, and possible task representations. The inputs in Fig.1.1 (lefts) are goals, resources, and demonstrations. A goal formulation is mandatory in task planning and can be achieved with a symbolic description of the final state of an environment. Besides that, the resources need to be described that a task planner can use to reach the specified goal, for instance, the parts that shall be manipulated. One or multiple user demonstrations serve as input for PbD or collaborative programming.

The task definition methods in Fig.1.1 (middle) are manual programming, task planning, programming by demonstration (PbD), and collaborative programming. From top to bottom, domain experts can implement a new task through manual programming. However, this requires expert knowledge in two fields: First, in the used programming language, the user must be able to write programs using g-code, robot-specific dialects, or high-level languages such as Python, C++, or C#. Second, in the software ecosystem, the user needs to know the application programming interface or the capabilities of a software library and decide how specific software functions are used. Alternatively, also skill sequences or task graphs can be programmed manually supported by graphical

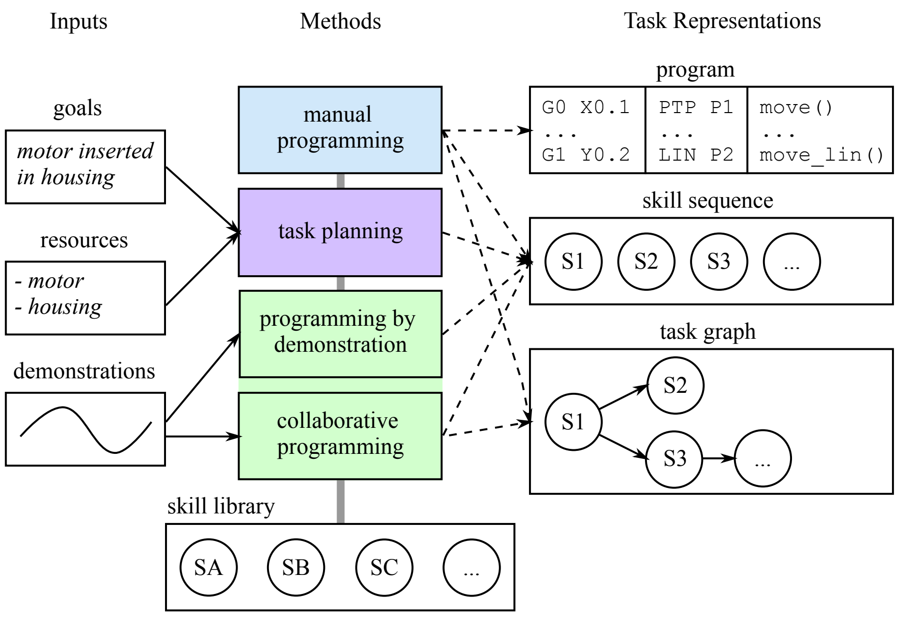


Figure 1.1: The task definition process connects different inputs with task definition methods that create possible task representations.

programming environments. Task planning algorithms enable the automatic definition of a task plan, usually represented as a sequence of actions, referring to the skill sequence in Fig. 1.1. A limitation from an end-user's perspective is that the task goals and resources need to be explicitly defined, which is a difficult task that might require expert knowledge. Furthermore, it is known that planned tasks do not perform well in practical applications due to the symbol grounding problem. Finally, end-users have the option to define a task by PbD (Chapters 4, 5 and 6) or collaborative programming (Chapter 7). The latter allows the incremental extension of a task graph, enabling decision-making or recovery behaviors (Chapter 7).

The task representations in Fig.1.1 (right) are program, skill sequence, and task graph. A manually coded robot program can encode dexterous robot behaviors such as decision-making, reactive behaviors, or synchronization with other agents. However, it is not intuitive to read and to adapt for end-users. A skill sequence instead represents explicitly the intended actions of the robot. Each skill encapsulates several hardware functions and, therefore, hides the internal complexity from the end-user. While a skill sequence is limited to a sequential execution flow, a task graph can encode more dexterous robot behaviors, for instance, decision-making or error recovery. Although its representation is more complex than a skill sequence, it is well-interpretable by end-users since the individual building blocks remain legible. Therefore, a task graph is assumed to be a good trade-off between legibility and allowed complexity of the robot behavior it can implement.

Indeed, the underlying skills used in PbD or collaborative programming use some implementation in one of the previously mentioned programming languages. However, their code is hidden from the end-user, which reduces complexity and gains explainability. PbD can also be combined with other paradigms such as human feedback and transfer learning [7], while this thesis focuses only on PbD and human feedback in the form of additional demonstrations, used to extend the robot's knowledge.

1.2 Motivation of Robot Skills

This work assumes that a system is equipped with previously defined robot skills. Two different perspectives motivate the usage of the concept of skills. From the human perspective, researchers suggest that humans structure their learned movement capabilities into reusable blocks called motor skills [8, 9]. Due to the separation of learned motions into these building blocks, humans can reuse and adapt the learned patterns for similar tasks without retraining them. More recently, it has been shown that such skills are developed over the whole lifespan of a human [10], which is an appreciated feature if transported to the robotics domain.

From the technical perspective, this work states four main motivations for using skills. First, the system designer is able to implement a skill as a function, which has parameterizable arguments for a specific task. Parameterization allows adaptation and reusability of the same skill in different environments. Second, an end-user can reuse existing building blocks when a new task needs to be defined. With a known set of skills, the user can learn the system's capabilities by looking only at an interpretable representation of the robot actions. The skills offered by a robotic system can be loosely related to human skills or actions. This relation is beneficial when a user tries to understand or modify an existing task representation consisting of a sequence of skills with legible names, for instance, *pick, move, place, press.* Third, a robot execution system that performs skills can run these as delimited execution blocks. This structuring helps to start, stop, pause, or resume the operation at known states, e.g., between two skills. Fourth, to manage the complexity within skills, the underlying implementation, specific robot controller parameters, or only machine-readable data are hidden from the end-user, who has potentially no experience in robotics. This encapsulation of the functionality simplifies the usage of skills for end-users since the complexity of the program code is hidden by the system designer, who needs to ensure that only human-readable and relevant parameters are visible from the outside.

This thesis puts a high emphasis on force-based robot skills. It is assumed that their definition and parameterization are more complex to achieve for end-users than pure motion or manipulation skills such as *move*, *pick*, and *place*. Compared to manual programming, where code is often repeatedly implemented for every new task, skills use existing building blocks such as control algorithms and strategies, which can be parameterized for the context in which they are used. From the end-user's perspective, a skill is a self-contained entity with an explainable behavior described by its name. From the outside, a skill exposes only its parameters as a human-readable interface.

An essential part of this work is defining robot skills that both humans and robots can interpret. From the human perspective, this is mainly achieved by legible skill labels, which provide an understanding of what the actual skill intends to do. From the robot perspective, a skill shall be interpretable so that its characteristics can be recognized from a human demonstration. Interpretability does not necessarily require human interpretable data that is describing the characteristics of a skill. Still, it is required that an algorithm, such as a classifier, can predict the correct skill with sufficient accuracy.

1.3 Definition of a Robot Skill

Definition 2 *A robot skill implements a robot capability and represents it as a generic building block for robot programs that can be parameterized for different contexts.*

A skill can have multiple inputs, termed parameters, and has at least one or multiple outputs. A parameter has a name and a value, allowing to set default parameter values. An output has a name and a value, where at least one output specifies the outcome of a skill. A skill writes to the output once it terminates. Additionally, outputs can be connected to parameters of other skills.

A skill can be seen as a building block of a robot program that implements a policy to reach a specific goal. Task-specific parameter values passed to the skill parameterize this policy. More flexibly, a skill could even employ multiple strategies or policies to reach the same goal, where a selection could be made based on the passed parameters. In literature, skills are often conceptualized with pre- and postconditions, also called effects. This conceptualization is reasonable and necessary in symbolic reasoning or task planning problems. However, this thesis does not require the modeling of skill-specific pre- and postconditions per se in the context of PbD. This additional specification is only needed if PbD is combined with symbolic reasoning. Otherwise, trivial conditions that describe if a skill is running or terminated are also sufficient. Figure 1.2 shows a schema of how skills can be implemented in a hardware-agnostic manner. The following sections refer also to this concept.

1.3.1 Skill Granularity

Skills can be defined in different granularities, describing the intended scope of their actions that form a reusable building block for different robot programs. For example, a pick skill can be in charge of approaching and grasping objects, whereas a pick and place skill can handle picking and placing objects in one go. If the granularity is fine, skills can be reused and assembled more flexibly to define tasks. However, if the granularity is too fine, handling them for the end-user might be cumbersome, and the explainability might decrease. If the granularity is too coarse, the reusability and generalization capability of a skill might be decreased, allowing just to handle very specific situations. Furthermore, the explainability decreases again if it is no longer possible for the end-user to understand what the skill does because of its internal complexity. The question about the granularity is not yet consistently answered considering existing literature about robotic actions [11]

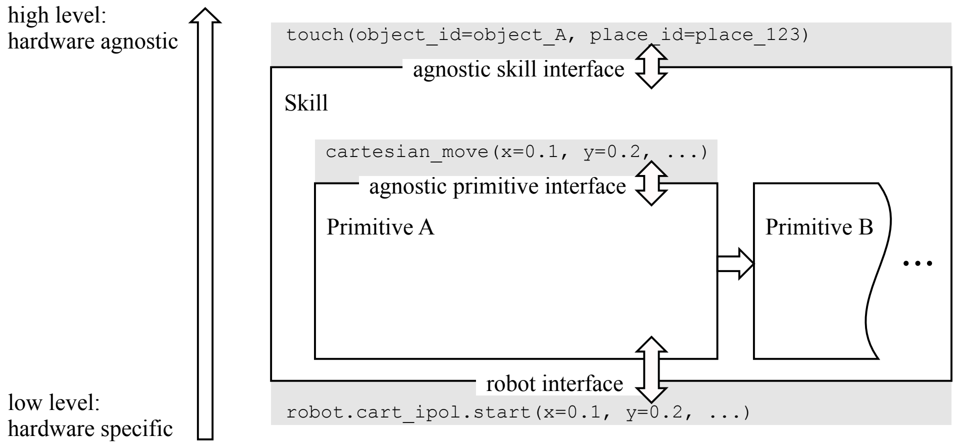

high level:
hardware agnostic
touch(object_id=object_A, place_id=place_123)
agnostic skill interface
Skill
cartesian_move(x=0.1, y=0.2, ...)
agnostic primitive interface
Primitive A
Primitive B
...
robot interface
robot.cart_ipol.start(x=0.1, y=0.2, ...)
low level:
hardware specific

Figure 1.2: A skill can be composed of multiple primitives. A skill uses a hardware-agnostic interface towards a higher-level control instance (top). Its primitives use a hardware-agnostic interface within the skill and a hardware-specific interface towards the low level (bottom), connecting them to the robotic system.

and no consensus exists yet in this area. There also exist hierarchical concepts of skills that can be composed of other skills [12]. This work follows the structure surveyed in [13], where a skill comprises a single layer, consisting of one to many primitives that command the robotic hardware or sense the environment.

1.3.2 Skill Composition by Primitives

Primitives are defined as basic robot capabilities, which connect hardware-agnostic functionality with hardware-specific interfaces (Fig. 1.2). The term agnostic describes that the skill is independent from hardware-specific assumptions. Primitives are also termed atomic functions [14], hardware functions, or elementary actions as explored in [13]. An exemplary primitive is a point-to-point interpolation in Cartesian space, supported by most modern robot control architectures. Although the expected outcome can be considered hardware agnostic, i.e., the robot shall move its end-effector linearly from point A to point B, the communication with the robotic system is often hardware-specific. Nowadays, system integrators often have to deal with robot-specific and proprietary interfaces for different robot types and implement new primitives and skills in a hardware-specific way. However, some progress can be seen in the standardization of hardware interfaces, considering real-time control such as OPC UA over Time Sensitive Networking (TSN) [15]. For high flexibility and scalability, primitives could implement a hardware-agnostic interface visible to the skills.

1.4 Teaching Data Sources

Programming by Demonstration, Learning from Demonstration, or Imitation Learning all refer to extracting knowledge from the user by observing a task demonstration. The common goal is to learn an execution behavior that solves a task. Compared to autonomous or reinforcement learning approaches, learning in the presence of a human teacher can speed up the task definition process because the user can account for exceptional task constraints that a system designer did not foresee. Since the research domain of glspbd considers a variety of data sources, the most relevant ones in the context of this thesis are introduced and summarized at the end of this section.

1.4.1 Kinesthetic Teaching

Kinesthetic teaching is a widespread methodology in the LfD domain. Usually, a human teacher physically interacts with a robot by exchange of forces. Typically, the human holds the robot at a single or multiple locations at the end-effector. Furthermore, the human can interact with additional parts of the robot, for instance, close to the elbow joint, to configure the null space of the robot [16–18]. During the teaching process, robots are usually controlled in a compliant or free-floating manner such that the controller compensates for their weight and the payload. This control mode is known as gravity compensation or zero-gravity control. When considering a robot's capability, a robot is described to be backdrivable. This capability can be achieved mainly by two different means [19]. Firstly, admittance control uses an external force measurement to regulate the motion when facing external forces caused by the user. Usually, a velocity reference is generated by measuring the external force on the robot [20]. Secondly, impedance control measures the deviation from an equilibrium point to regulate a force [21], which offers a more sensitive control mode with smaller interaction forces between human and robot. The controller is usually parameterized such that it behaves like a spring mass damper system with zero stiffness. This system enables a free-floating robot arm that holds its weight and, if desired, the weight of the payload.

A system used for PbD usually records the Cartesian pose and the Cartesian forces and torques at the end-effector. Force and torque are collectively termed as wrench in the following. Furthermore, a system can record the joint positions and, if applicable, end-effector states. These states can be the gripper finger opening width, grasp forces between gripper fingers, and grasp status, which describes the presence of an object in the gripper.

For torque-controlled robots equipped with joint torque sensors, it is possible to measure the individual joint torques and estimate the Cartesian wrench at the TCP or any known contact point at the tool or robot structure.

A challenge with kinesthetic teaching is recording forces at the end-effector, which requires an additional FTS during the teaching phase. For clarification, there are two different situations with each two cases that can be distinguished. In the first situation, it is assumed that the end-effector is in free motion as shown in Fig. 1.3 on the left. In case 1, a human interacts with the robot structure before an additional FTS, considering the kinematic chain in green. The interaction force $\boldsymbol{f}_{\mathrm{T}}$ equals the force $\boldsymbol{f}_{\mathrm{dyn}}$ caused by the robot inertia. The measured force at the FTS $\boldsymbol{f}_{\mathrm{FT}}$ is subject to the tool inertia, which can be canceled out if the tool inertia, gravity vector, and current acceleration are known. In a state of constant velocity and without touching the end-effector, $\boldsymbol{f}_{\mathrm{FT}}$ approaches zero.

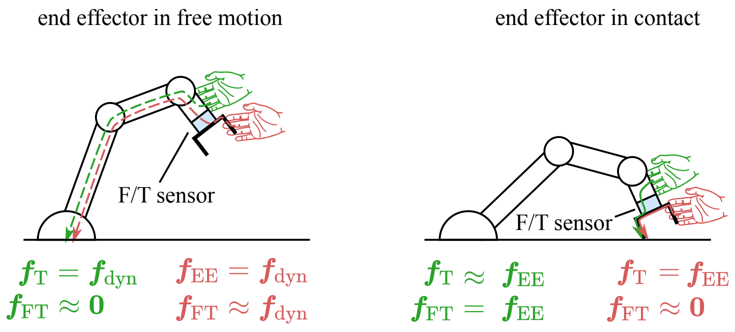
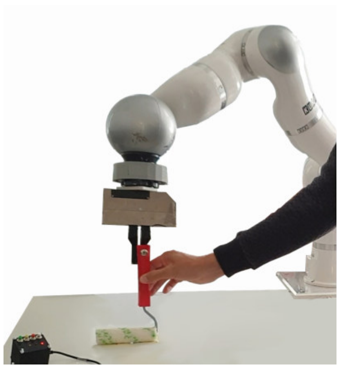

end effector in free motion
end effector in contact
F/T sensor
F/T sensor
$\boldsymbol{f}_{\mathrm{T}} = \boldsymbol{f}_{\mathrm{dyn}}$
$\boldsymbol{f}_{\mathrm{FT}} \approx \boldsymbol{0}$
$\boldsymbol{f}_{\mathrm{EE}} = \boldsymbol{f}_{\mathrm{dyn}}$
$\boldsymbol{f}_{\mathrm{FT}} \approx \boldsymbol{f}_{\mathrm{dyn}}$
$\boldsymbol{f}_{\mathrm{T}} \approx \boldsymbol{f}_{\mathrm{EE}}$
$\boldsymbol{f}_{\mathrm{FT}} = \boldsymbol{f}_{\mathrm{EE}}$
$\boldsymbol{f}_{\mathrm{T}} = \boldsymbol{f}_{\mathrm{EE}}$
$\boldsymbol{f}_{\mathrm{FT}} \approx \boldsymbol{0}$

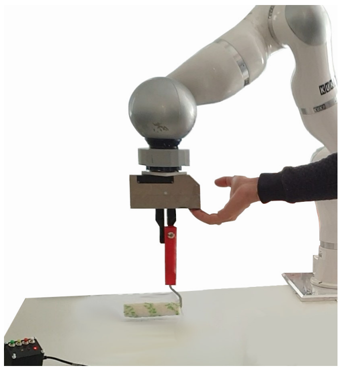
Figure 1.3: Force transmission flows depicted as arrows for different contact situations during a gravity compensation control mode. Left: In the free motion case, no static forces act on the end-effector when interacting with the robot at the point of the green hand. Interacting at the point of the red hand causes the FTS to measure the forces caused by the robot dynamics. Right: In the contact case with a closed-loop kinematic chain, forces at the end-effector can be measured by the FTS when interacting with the robot at the point of the green hand. Instead, no human-induced contact forces can be measured at the end-effector when interacting with the robot at the point of the red hand.

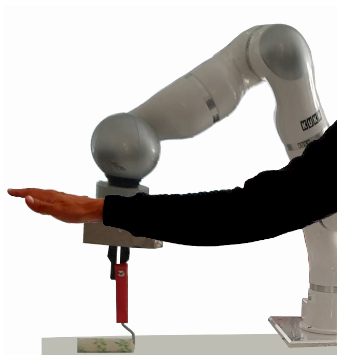
In case 2, a human interacts with the robot structure after an additional FTS at the end-effector, considering the kinematic chain in red. Now, the force at the end-effector $\boldsymbol{f}_{\mathrm{EE}}$ equals the robot's dynamics force $\boldsymbol{f}_{\mathrm{dyn}}$. Additionally, the force measured by the FTS approximates the robot's dynamics force $\boldsymbol{f}_{\mathrm{dyn}}$.

In the second situation, it is assumed that the end-effector is in contact with the environment as shown in Fig. 1.3 on the right. Once the end-effector makes contact with the environment, traction can be established between the human hand and the environment via the end-effector. In case 1, a human interacts with the robot structure before an additional FTS, considering the kinematic chain in green. The interaction force $\boldsymbol{f}_{\mathrm{T}}$ approximates the end-effector force $\boldsymbol{f}_{\mathrm{EE}}$. The FTS measures now the end-effector force, i.e. $\boldsymbol{f}_{\mathrm{FT}} = \boldsymbol{f}_{\mathrm{EE}}$. In case 2, a human interacts with the robot structure after an additional FTS at the end-effector, considering the kinematic chain in red. The interaction force $\boldsymbol{f}_{\mathrm{T}}$ equals the end-effector force $\boldsymbol{f}_{\mathrm{EE}}$] between end-effector and environment. However, the FTS force $\boldsymbol{f}_{\mathrm{FT}}$ approaches zero because the kinematic structure between the sensor and robot base is free-floating due to the gravity compensation control.

A further challenge is to ensure a safe interaction between humans and robots. Most works consider separated teaching and execution phases where the user interacts with the robot in the first phase, and the robot executes an autonomous behavior in the second phase. Ideally, a user should be allowed to physically interact with or correct the robot at any time, which requires a detection mechanism when and how this happens [22]. Overall, important interaction types to be considered are intended, and unintended interactions. Interaction contact points can be found at the manipulated object, tool, and robot body (Fig. 1.4). At the same time, the robot tool can be in contact with the environment or not.


(a) Interacting with tool


(b) Interacting with gripper


(c) Unintended interaction / collision

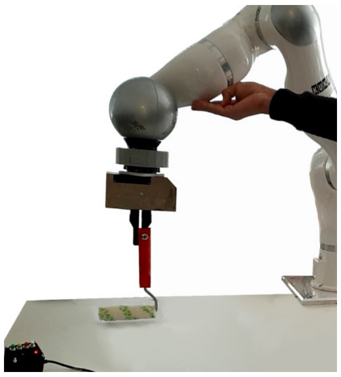
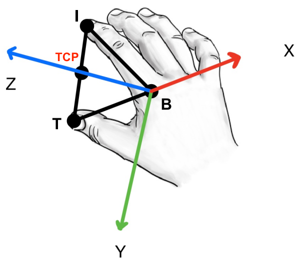
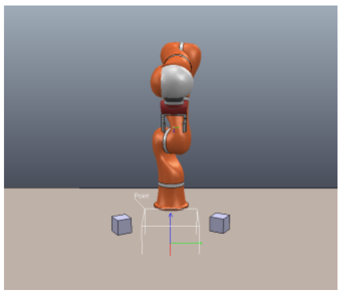

(d) Interacting with robot body

Figure 1.4: Four different interaction types between human and robot in kinesthetic teaching. In all cases, the tool can be in contact with the environment or not.

(a) Task demosntration by hand pose.



(b) Kinematic mapping between hand and gripper.



(c) Reproduction of the task on a simulated robot.

Figure 1.5: A task definition system that uses hand pose demonstrations (left). The hand coordinate system is constructed from the three points I, B, T forming the kinematic structure of a two finger gripper (middle). This kinematic mapping allows to reproduce the task on a robot in simulation (right).

1.4.2 Hand Pose

Intuitively, humans learn from each other when they use their hands in everyday manipulation tasks. PbD aims to transfer this learning concept to robots. Therefore, techniques to observe the human hand have emerged, known as hand pose estimation [23], and hand pose tracking [24]. These techniques allows to estimate the position of individual finger joints and links, which enables the teacher to transfer specific grasp configurations to a robotic system. Usually, a deep neural network generalizes the estimation among different biometric hand shapes, lighting conditions, backgrounds, and even occlusions caused by objects in the observed hand [25].

Relevant data from the hand pose estimation consists of each finger's joint and link position, obtained periodically by tracking the hand with a depth camera and feeding the image frames into a neural network. The estimated finger poses can then be used in the respective application to learn from a human strategy and to parameterize existing robot skills [26]. Additional data from a human hand and arm are biosignals obtained by force- and electromyography [27] or by optical myography [28], which allows a system to learn from hand or finger gestures or muscle activation patterns. Figure 1.5 shows an approach that maps a simplified kinematic structure onto a human hand and how it can be used to command a two-finger robot gripper [26].

1.4.3 Teaching Device

Teaching devices are systems that a human demonstrator can hold to transfer motions and forces to a learning algorithm or directly to a robotic system. They can be manually [29] or actively actuated to operate a gripper as an end-effector. Figure 1.6 shows a teaching tool developed at the German Aerospace Center that can actively operate a gripper. It connects an optically tracked grasping handle with an FTS and a flange to attach different tools, such as a two-finger gripper. The whole device is battery-powered.

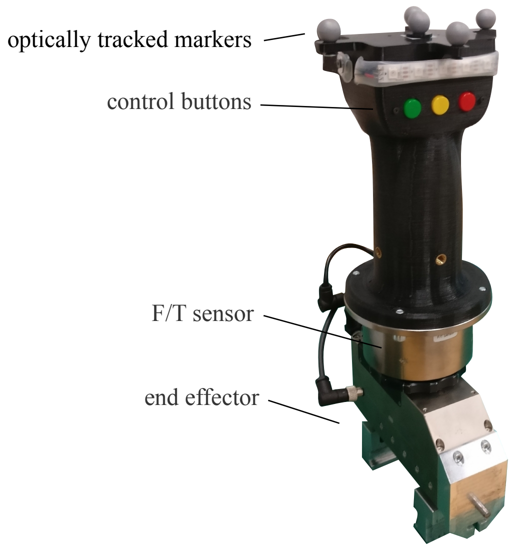


Figure 1.6: A wireless, manually guided teaching device that is used to record the Cartesian motion and wrench of a human demonstration.

Table 1.1: Data sources, their sensing modalities, and properties

	Modalities			Properties	
	joint pose	Cart. pose	wrench	corre-spon-dence	inertia free
Kinesthetic teaching	✓	✓	-	✓	-
Kinesthetic teaching w/ FTS	✓	✓	✓	✓	-
Hand-pose estimation	✓	✓	-	-	✓
Teaching device	-	✓	✓	-	-

It wirelessly transmits the measured wrench, readings from an inertial measurement unit, and the button states that can be used, for instance, to trigger the start and stop of a demonstration recording. An optical tracking system observes the markers on top of the device.

1.4.4 Summary

Tab. 1.1 shows the introduced teaching methods with their data sources. The "correspondence" column refers to the property if the demonstration system is the same embodiment as the one used for task execution. This property is true for kinesthetic teaching, which inherently solves the correspondence problem between demonstration and execution. Instead, other data sources might collect data that leads to behaviors not executable by the robot due to limitations in its manipulability. The "inertia-free" column describes whether the user can serve a data source without being constrained by additional inertia other than the manipulated objects. For instance, kinesthetic teaching or a teaching device require the user to deal with the inertia of the robot or teaching tool, respectively. Since this inertia has to be accelerated and decelerated during the demonstration phase, it might negatively influence the teaching performance regarding speed and accuracy. Hand-pose estimation overcomes this limitation since the human teacher naturally uses their hand as a tool. However, measuring forces on the human hand requires attaching additional sensors to the palm and fingertips. In summary, this thesis focuses on kinesthetic teaching with a FTS due to its advantages over the other data sources.

1.5 Technology Readiness Level

The framework proposed in this work targets Technology Readiness Level (TRL) 5 as an indicator of the maturity of a technology. Level 5 describes a "Component and/or breadboard validation in relevant environment." according to the NASA definition [30] or "Technology validated in a relevant environment" according to the European Union Horizon 2020 Programme [31].

This statement emphasizes the applicability of the proposed methods. Compared to theoretically focused research, which often lacks a realistic validation in a relevant environment, this work aims to reach a high practical value to the field of robotics in industry and manufacturing. Specifically, all algorithmic developments of this work were tested on real robotic hardware, and all of them were evaluated with humans in the loop, where several user studies helped to stress the proposed methods against novice users.

1.6 Structure of this Thesis

The main topics of this thesis cover skill identification, recognition, and parameterization. While the skill identification is conducted once and leads to a set of legible skill types, skill recognition uses this set during the task definition process. Once a skill is identified (Chapter 4), different approaches are proposed to recognize it from a user demonstration (Chapters 4, 5, and 6). Besides programming of single skills, sequential tasks and task structures that involve decision-making or recovery behaviors can be programmed (Chapter 7). A unifying framework shows how a user can program a new task either via demonstration or via offline skill-based programming (OSP), and how the skills are parameterized and executed (Chapter 8).

Chapter 2 : Related Work introduces the state of the art and points out the advancements of this thesis concerning skill identification, recognition, and parameterization. It compares existing frameworks for PbD and examines how end-users interact with them.

Chapter 3 : Prerequisites and Background states the prerequisites and the background that the following chapters commonly use. It describes the mathematical symbols and expressions and defines common terms to create a consistent understanding across the following chapters.

Chapter 4 : Skill Identification and Recognition presents the approach of skill identification through a user study with a manual labeling task. In a manual classification task, a human test group verifies the interpretability of the extracted labels. Subsequently, the chapter proposes an approach to recognize the identified skills with an automatic segmentation and classification system using only proprioceptive data.

Chapter 5 : Skill Sequence Recognition proposes an approach to extract sequences of skills to program complete tasks. It exploits the parallelization of two skill recognition pipelines, namely a symbolic and a data-driven recognition pipeline, to recognize the demonstrated skills on the fly. At the same time, it builds a visual task representation, which provides feedback to the user about what the robot has already understood from the demonstration. It also presents a user study that evaluates both the recognition capabilities and the explainability of the task representation.

Chapter 6 : Contact-based Exploration presents an approach that can learn from multiple demonstrations to explore uncertain object locations in the workspace. This concept combines the previously identified touch skill with gripper-based object manipulation skills. The exploration strategy is fully extracted from the demonstrations. The recognized and automatically parameterized touch skills can parameterize subsequent skills at execution time, leading to an adaptive execution behavior.

Chapter 7 : Task Decision Programming proposes two approaches that each allow end-users to program task decisions or recovery behaviors by demonstration. While one approach allows end-users to program them in the teaching phase, the other allows adding them incrementally during the execution phase. A user study compares the introduced approaches to a baseline, providing various insights about how users behave when programming task decisions or recovery behaviors.

Chapter 8 : Unified Framework for Skill-based and Demonstration-based Programming proposes a unified programming framework that combines PbD with OSP as task definition methods. A knowledge-driven system based on ontologies integrates the stages of skill recognition, parameterization, and execution. The chapter presents the advantages that arise from this combination: improved skill parameterization with ontological knowledge, ease of programming by empowering users to select a method based on their capabilities, and reduced demonstration effort by exploiting existing knowledge in the ontologies.

Chapter 9 : Conclusion summarizes the findings from each of the previously mentioned chapters and states future research directions.

2 Related Work

The state of the art is structured with respect to the main chapters of this work. *Skill Identification and Recognition* (Sec. 2.1) refers to Chapter 4, *Recognition of Skill Sequences* (Sec. 2.2) refers to Chapter 5, *Force-based Exploration and Manipulation* (Sec. 2.3) refers to Chapter 6, and *Learning Task Decision Making* (Sec. 2.4) refers to Chapter 7. Finally, *Intuitive Robot Programming Frameworks* (Sec. 2.5) refers to Chapter 8.

2.1 Skill Identification and Recognition

Various tasks can be listed that require force-based robot behaviors, which could be implemented in the form of skills. These are for instance peg-in-hole [32, 33], wood planing [34], engraving [35], pouring from a bottle [36], door opening [37, 38], screw driving [39], object grasping [40], touching and contact-sensing [41–43], pushing [37], rotating a hand-wheel [44], and ironing [38]. Additionally, a number of methods for collaborative use-cases were proposed, such as for assembly [45] or human-robot joint transportation tasks [46, 47], whose implementations can be also seen as robotic skills.

Skill identification will be later introduced in Chapter 3, Definition 4. It refers to the process of finding which skills are purposeful to a human for programming a system and to a robot in achieving the goals of a task. This process is considered to be completed before the actual programming phase and is also known as skill definition [48] or skill formalization [49].

Skill recognition will be later introduced in Chapter 3, Definition 5. It is the process of inferring about which skill is performed in a human demonstration. Usually, recognition of a skill is based on time series data, hence requiring to solve a twofold problem consisting of simultaneous segmentation and classification. This thesis defines skill recognition to solve both problems of time series segmentation and classification to infer the semantic meaning of each segment. Generally, the research community is unsure about if segmentation is defined to involve only the process of splitting a time series into multiple segments or if it also incorporates action recognition, meaning that each segment has a descriptive character.

Force and motion based skill recognition has been early proposed in [3]. They show how force-based state transitions are classified given a task model in the form of a finite state machine. The surgical skill evaluation in [50] employs Hidden Markov Models to detect manual skills in a fixed environment. In both of these works, the classification is task-specific and would have to be re-learned if the environment changes. A better task invariance is shown in [51] by introducing force primitives and claiming that only 20 different relative motions between two mechanical parts are possible. Instead of labeling actions or skills, the proposed primitives are labeled, for example *edge parallel to surface* or *surface against surface*. It is left open how the process of task decomposition into primitives can be automated or even learned from demonstration. In [41], only contact events are classified from time series data in a task specific setup without considering additional contact-based behaviors.

There exist also explicit approaches that focus on the problem of task invariance [52–56], presenting descriptors that can be used to encode a motion in a task invariant manner avoiding any relation to task specific motion frames. However, none of these works consider task invariance of forces and torques, which is required to describe a skill on the robotic capability level and making it task invariant.

Unsupervised learning to solve the segmentation problem could be an option to avoid a strong overfitting on specific tasks. Relevant works are about surgical gesture recognition [57], bi-manual robot task learning [58], surgical trajectory segmentation [59], and painting task imitation [60]). However, unsupervised segmentation methods have the problem that they do not provide human interpretable labels for the identified segments and therefore cannot semantically describe the segmentation result. This leaves the user alone with a task representation that is difficult to interpret.

Often, probabilistic models are considered to compare the current action with, e.g., in the form of Hidden Markov Model (HMM) or GMM [61, 62], Beta-Process Auto-Regressive HMM (BP-AR-HMM) [58, 63], or probabilistic flow tubes [64].

2.2 Recognition of Skill Sequences

Approaches for recognition of skill sequences from a human demonstration can be divided into two major categories. One category utilizes symbolic action descriptions and pre- and post-conditions, referred to as a symbolic approach. The other category involves data-driven methods that use a pre-trained model to recognize skills, referred to as a data-driven approach. Each approach has its own strengths, which can be best illustrated through an example about kinesthetic teaching. Suppose an end-user opens the robot's gripper during a task demonstration. This event can be easily captured and described on a symbolic level by evaluating the truth value of a symbol that describes the open state of the gripper. Such events are useful in establishing preconditions for robotic skills, such as *open-gripper* or *place* skills. Applying a symbolic approach is suitable for this scenario as the underlying conditions can be easily designed, implemented, and confidently evaluated with minimal cost. On the other hand, training a classifier to infer about opened and closed gripper states would be computationally inefficient and would require a dataset of opened and closed gripper states. Another example is about a user who moves the robot tool along a surface while exerting force onto the plane. Here, a sequence of measurements is generated that is challenging to be described by manually designed conditions. Consequently, a trained classifier is the favorable choice to recognize an appropriate skill in this phase of the demonstration, such as a surface processing skill as required in robotic grinding.

2.2.1 Symbolic Approaches

Symbolic approaches assign names to specific robot states to define the meaning of the represented knowledge. Such names are referred to as symbols, which can be used to form predicates that allow to evaluate the pre- or post-conditions of a skill. According to [65], a well-known syntax for defining these conditions is the Planning Domain Definition Language (PDDL) [66]. By evaluating the pre- and post-conditions of all predefined skills simultaneously, a skill can be inferred if all of its conditions are fulfilled. This is referred to as the symbolic approach to recognize a sequence of skills from a demonstration. An overview recognition algorithms that focuses on human motion recognition is presented in [67], highlighting their semantic aspects.

The action recognition system presented in [68, 69] uses a module that produces a symbolic description of the scene based on visual perception. The work in [70] provides a definition of a robot skill within an industrial setting, which outlines a skill conceptual model that includes pre- and post-condition checks. In this context, the sequence of skills that found a task is programmed manually and afterwards parameterized.

The approach in [71] suggests that each skill has a unique set of post-conditions, which describe the effects the skill has on the state of the robot and its environment. Additionally, pre-conditions can be employed, which describe the conditions that need to be fulfilled before or during skill execution. A skill description used for skill recognition that employs pre- and post-conditions can be achieved with Planning Domain Definition Language (PDDL) [72]. To check if such conditions are fulfilled, a world model can be used that provides the poses of all relevant objects in the workspace [72]. If such world model is not updated frequently by the robot's perception capabilities, it can lead to inconsistent states. Therefore, [73] used image recognition to evaluate which conditions are fulfilled. It introduces so-called Object Action Complexes, where the predefined conditions in the action recognition describe contact changes between objects in the sense of overlapping image segments. An estimation of physical contacts is not considered in their work.

In summary, pure symbolic segmentation methods require the manual design of pre- and post-conditions for each skill, making it challenging to find distinguishing symbolic expressions and requiring manual design effort. This is particularly problematic for skills that act on the same object but differ only in the forces they apply. Several attempts have been made to extract pre- and post-conditions automatically [74–76], but none have been applied to a variety of force-based interactions such as consecutive in-contact movement primitives.

2.2.2 Data-driven Approaches

Data-driven approaches learn to discriminate skills based on a pre-trained model or by finding grouping patterns in the data. For instance, demonstrated motions can be encoded in a collection of probabilistic models. During inference time, newly observed motions can be classified by comparing them with the existing models [77]. A data-driven framework for task structure extraction is presented in [78], which relies on an activity recognition method based on visual perception [79]. In [80], a segmentation technique is proposed that utilizes variance analysis among demonstrations relative to data variability within a time window. Simultaneously, various object frames are tracked to map segments to them. The approach requires a world model with associated coordinate frames and can only be used with multiple demonstrations of the same task.

Visual perception of the human hand is used in [26], applying so-called template matching with HMMs to recognize skills that are performed by hand and finger movements. This allows the user to naturally pick and place objects with the hand, without dealing with the inertia of the robot as in kinesthetic teaching. Visual hand-pose estimation alone does not allow to infer about interaction forces, as they occur in contact skills. Although attempts have been made to estimate forces and dynamics indirectly from visual perception [81], such an approach can only provide a rough estimate about real contact forces. A sliding window approach is used in [82], where a SVM classifies the image sequence in the current stream window to recognize skills. The previously mentioned approaches limit the detection of skills to visually observable behaviors and ignore measurable interaction forces or torques. Instead of relying on vision data, tactile and proprioceptive signals were considered in [83], presenting a segmentation algorithm that is based on Bayesian online change-point detection, but without the capability of inferring a human readable task representation due to the unsupervised nature of the approach.

Unsupervised approaches typically attempt to identify similarities between demonstrations or fit generative models to segment the data. Similar skill transition states in repeated task demonstrations are identified in [59] using hierarchical clustering based on GMMs. A first GMM is fitted to each demonstration to obtain candidate transition states, which were then clustered in secondary GMMs in the spatial, sensory, and temporal domains. Bayesian non-parametric extensions of the HMM are used in several studies [63, 84, 85]. A beta process (BP) prior helps to infer the number of active modes, which are referring to skills, per demonstration and enables sharing of identified modes across different demonstrations. As shown in [63, 84], a Beta Process Autoregressive HMM (BP-HMM) relaxes the conditional independence of observations by describing their time dependencies as a so-called vector auto-regressive process. In [85], a BP-HMM is combined with a clustering approach to determine the appropriate level of granularity for identified motion primitives based on clustering performance. Only single demonstrations were exploited in [86], which accelerated the teaching phase. The approach infers the observed skill sequence by using the cost of compliance with predefined feature constraints as grouping mechanism in the data, as it is also known from inverse reinforcement learning.

A known limitation of data-driven approaches is the requirement of some amount of training data. Supervised approaches even require capturing of labeled training data, which can be an expensive process that is blocking hardware, computational resources, and manpower. During inference time, data-driven approaches also comes with a delay in prediction time due to computational cost. Further, the prediction accuracy might be limited depending on the generalization capability of the employed model and extent of the training set. Such disadvantages should only be accepted by a system designer if the underlying problem cannot be solved by straightforward and manually implemented rules, as they are used in the symbolic approach.

2.2.3 Combined Approaches

Combining symbolic evaluation and data-driven algorithms aims to utilize the strength of both approaches. In [87], it is shown that symbols can automatically emerge through task observations by learning to discriminate a variety of system states with a probabilistic model. These symbols are extracted to be used in task planning, which operates in a discrete and abstract state space through PDDL, where they are used as pre- and post-conditions. However, the approach requires a significant number of manually collected samples to cover all potential scenarios in the surroundings of the robot. With that samples, multiple symbol grounding classifiers are trained. This implies that also for simple state symbols such as `gripperOpen`, a model is learned before symbolic evaluation, which contradicts the motivation to learn only the necessary parts of a task and provide prior knowledge where applicable. These symbols are then able to segment the data in an event-based manner whenever they change their truth value.

With the aim to predict skills online during the demonstration, fast evaluating of a symbol can be based on a single measurement. The approach in [87] combines data-driven and symbolic approaches. Basically, the evaluation of any symbolic state could be learned if there is sufficient data present, which would take away the burden of implementing evaluation strategies from the system designer. Notwithstanding, only complex features should be learned that cannot be easily described by the system designer. For example, it would be hard for a programmer to take care of multi-dimensional and mutually interacting dimensions, which is for instance known as cross-talk [88] or action interference [27, 89]. Another well-known problem in classification describes if a dataset is linearly separable or not, which is rather hard to tackle by manually described rules. Data-driven classification algorithms can easily address this with nonlinear decision boundaries or kernel methods. Lastly, information could also be hidden in temporal sequences that describe the process dynamics instead of being observable in a single states. For instance, an approach can observe only the force thresholds in robotic assembly [90] or consider the temporal information of force transients instead [43].

This thesis proposes to combine the symbolic with the data-driven approach to utilize the strength of both techniques in an online skill recognition framework (Chapter 5). With that, manipulation skills such as *pick* and *place* can be defined mostly through symbols that can be intuitively described and implemented by simple rules. Instead, contact skills, which are characterized by their time series and process dynamics, are recognized through data-driven methods.

2.3 Force-based Exploration and Manipulation

Force-based exploration is defined as the strategy, where the robot uses its body or an attached tool to explore unknown regions in the workspace in order to update the world state with the gained knowledge. Section 2.1 introduced a number of works that consider contact skills. It is noticeable that none of the mentioned approaches addresses the sensing of object locations in the environment.

Once an object location or a specific state of the environment has been sensed, the robot shall be able to adapt its task to the new situation. Therefore, a number of force-based learning frameworks are analyzed, which learn from forces to allow the adaptation of the robot behavior during execution time. The framework proposed in [91] allows the robot to continuously react to force variables which have been identified by Mutual Information analysis from the demonstrated behavior. The purpose of this algorithm is to find a mapping between state space variables and control input variables to allow real-time control of a force-based task. The experimental task was to guide a ball trough a maze by measuring the forces that the ball exerts and controlling the pitch and roll angles of the surface where the ball is rolling on. In the learning process, there is no motion segmentation employed that would consider involvement of several objects, making it unsuitable for manipulation tasks.

The method in [40] is applied on object manipulation. The robot learns grasping skills from motion and force data while it is tele-operated with a data glove. The learned forces are used to generalize the learned skill on objects of different mass and size. Although different object types are considered, the approach has no ability to explore the object location itself. Continuous online movement adaption is presented in [92] and also applied to reactive object manipulation. The approach lets the robot adapt its motion based on the perceived forces in accordance with a model learned from demonstration. This allows the robot to react to uncertainties and disturbances during execution. However, exploration of uncertain object locations is not considered.

Learning force-based tasks can be also combined with motion segmentation to support different reference frames that are associated with objects [80]. Another learning scheme pairs vision with tactile sensing [93] and focuses on material type and object detection. However, the system is not initialized or parameterized by means of PbD and the inferred class is not used to adapt the behavior of the robot. In the context of segmenting and sequencing, [94] proposes the prediction of manipulation phases and [12] shows learning of hierarchical skills by reinforcement learning, where both approaches require object tracking if the object is not initially in the hand. Complex tasks can be also learned by a combination of kinematic and video data from human demonstrations [84]. The approach does not incorporate force-based sensing and assumes that all objects can be visually observed in the scene. In [95], compliant manipulation is achieved by interaction-based phase transitions in a Hidden Markov Model, which is able to transition between non-constrained and constrained motions. The framework does not support non-linear unconstrained motions and exact goal points in free space were not addressed.

Autonomous exploration can be achieved even without a task demonstration [96]. Here, the whole robot workspace is haptically explored in combination with tactile-based object discrimination. Although this strategy does not require prior demonstrations, its goal is to explore the whole workspace to recognize eventual objects which is usually not the objective of a repetitive robotic task such as in production environments. In [97], the search policy is learned from human behavior to act in the whole workspace. Although the strategy is learned from human demonstrations, it cannot be tailored to a specific application by the end-user.

Instead of exploring a whole workspace, in-hand object localization reduces the spatial search volume but requires to have the object already fitted in the robotic hand. It can be achieved by a tactile sensor array [98], which can be combined with tactile based manipulation. Here, a DMP approach is used during execution similarly to [99], which reproduces both desired motion and tactile trajectory. This tactile trajectory contains the tactile image time series obtained from the sensor array. More elaborate sensing techniques make use of embedded sensors in gripper fingers [100], or tactile sensor arrays for object recognition [101]. The latter authors use sampling-based motion planning [102], which requires a model of the environment or a huge number of real robot executions.

Learning from human hand motion observation when grasping an object is presented in [103]. The goal is to correct the observed hand postures with the sensed contact information while grasping. Considering the field of tactile sensing, a review [104] shows that a variety of sensors is technically available but the major challenge lies in development of novel manipulation algorithms to actually make use of them.

Generally speaking, the analyzed approaches do either not explore the object location itself [40,91,92,95], unnecessarily search the whole workspace without the focus on object manipulation [96, 97], require visual perception for object tracking [12, 80, 84, 93, 94], depend on additional hardware such as sensitive tactile sensors and specific end-effectors [100, 101, 104], or allow only in-hand localization [98, 103].

2.4 Learning Task Decision Making

2.4.1 Learning Conditional Tasks from Demonstration

Conditional tasks require the robot to adapt to an environmental state and to make a decision about the further execution. Such strategies can be designed and implemented by hand in the form of a finite state machine (FSM) [105]. Alternatively, such plans can be also learned from multiple demonstrations [106]. Here, a knowledge base of manipulation tasks is built from demonstrations by observation of a virtual workspace and by segmentation into predefined skills with pre- and post-conditions. In [58], these plans are interactively built in form of a Finite State Machine (FSM), using visually tracked object poses. The state transitions are inferred by a classifier in case that there is more than one transition option. A classifier is trained on final robot pose and object pose for each state of the FSM. Recovery behaviors are considered for grasp and object pose failures, but unknown anomalies are not detected by the system itself and the user has to intervene if a new behavior shall be added.

Instead of learning the whole decision making strategy from demonstration, it could potentially be solved analytically such as with Markov Decision Process (MDP)s, if enough knowledge about the robotic workspace and possible world states would have been modeled. There exist also solvers that can handle continuous state spaces and do not require predefined transition probabilities, such as a Partially Observable Markov

Decision Process (POMDP)s [107]. They require a black-box model to sample from and the reward or possible goal states need to be defined manually. In [108], a switching scheme between deterministic planning and decision-theoretic planning in the form of POMDP is proposed, which requires a problem description in the Planning Domain Definition Language.

Common planning approaches, e.g. based on a Probabilistic Roadmap (PRM) [109] or on a Rapidly Exploring Random Tree (RRT) [110], always require a model in order to reason in the world, which has to be built beforehand. If this could be avoided, the end-user would have more freedom to program the robot in new, unmodeled scenarios. In [111], multiple demonstrations are encoded in sequences of predefined symbols to find the longest common sub-sequence. This approach can be used to construct generalized task graphs from multiple demonstration sequences of actions. As a prerequisite, this would require that the observed demonstration is first segmented into predefined symbolic actions that can be later used by the mentioned approach. Extraction of symbolic actions for planning is actually shown in [112] and [113]. In both works, pre-conditions and effects of symbolic actions are learned from demonstration but the corresponding symbols need to be manually defined. In [114], complex tasks are learned by kinesthetic teaching involving multiple visually tracked objects, where the task structure itself has to be defined manually.

2.4.2 Fault Detection and Recovery

Common terms for recognition of abnormal states are fault detection, error detection, or anomaly detection. Faults in the scope of robotics can be divided into internal robotic system faults and task execution faults [115]. This thesis only addresses the latter, where anomalies in expected robot pose, external wrench or grasp status are considered.

As part of a task execution faults, geometric assembly errors are theoretically described in [116], and possible recovery strategies are provided. The authors explicitly state that an event is termed an error if no solution exists to handle it. Further, they term an action a recovery strategy, if it is a possible solution to the given problem. Hence, it can be concluded that a task execution fault is considered to be a simple event in a guaranteed plan, if the system has the ability to detect it and change its plan or execution behavior accordingly.

Several approaches in the literature exploit time series of previously observed executions for error detection. Force signatures extracted from force or torque time series are exploited in [117] to train a SVM with a batch of successful and unsuccessful assemblies from hand-labeled trials. In [118], HMMs are trained as a process monitor in robotic assembly. The approach is only evaluated in a 2D experimental setup, requiring 20 samples to be observed for each contact event. In [119], a HMM is trained on the wrench and its derivative to monitor force-based interactions with the environment to enable reasoning about if the execution is successful or not. As a drawback, the specific modality, which allows for detecting the deviation, needs to be selected beforehand and an expert user need to observe a number of executions to manually label them. The execution strategy has to be manually programmed as a FSM. Also based on HMMs, a multi-modal abnormality detection is presented in [120], which monitors forces, vision and sound during execution.

The authors of [121] use a pre-processing mechanism that uses statistical dependency-detection methods to determine sub-groups of monitoring variables. These sub-groups are used in the online phase to compute the Mahalanobis distance for anomaly detection. A data-driven anomaly detection approach based on dimensionality reduction of sensor data, pattern recognition and a threshold on the Mahalanobis distance is presented in [122] and further evaluated in [123]. All these works have in common that they are able to detect abnormal states or faults but are not designed to recover from faults automatically.

The approach of task stratification is used to order possible faults into classes such as execution, planning, modeling, or sensing error [124]. This method also describes a forward and backward correction process to successfully accomplish an error-prone task. While providing labels for error classes, there is no methodology presented on how to identify an error given a state or time series of observations. A so-called task outcome prediction compares sensor signals with successful trials coming from a reinforcement learning system [125]. A z-test infers if the observed signals stem from a population of successful trials. This approach relies on manual labeling of successful and unsuccessful trials. Learning from failed demonstration trials is presented in [126] to avoid repeating the human's mistakes with the goal to converge faster to an optimal policy. The underlying assumption is that every shown trial is a failing one and the robot tries to find a better strategy in between these failures.

Error detection and recovery can be hand-designed in the form of an FSM. Such hand-designed detection schemes can then be tailored for specific autonomous behaviors. An example is the haptic-based detection of a hand driller's operation state by analyzing the frequency spectrum of a force sensor [105]. However, such detection schemes require manual design effort and they need to be activated in the right state during execution. Therefore, the system can only handle errors that the user has foreseen and implemented already.

An FSM can also be constructed from a number of human demonstrations [58]. Possible recovery behaviors were only considered, if the human triggered a button during execution. In contrast, the presented system in this thesis decides autonomously with the help of anomaly detection when a new demonstration is required. Furthermore, the pose of all task relevant objects is visually tracked by the system and the force domain is not considered in the task definition process.

2.4.3 Clustering of Time Series

The goal of time series clustering is to group similar demonstrations that could be later encoded in a model for motion primitives. The demonstration data in PbD can be in the form of multiple, multidimensional time series with unequal length due to demonstration variations. The data may consist of continuous variables from multiple modalities such as position, orientation, wrench and end-effector grasp information. A survey of time series clustering [127] groups popular methods into raw-data-, feature- and model-based approaches. Methods which act in the raw-data domain with multidimensional input and variable length often employ Euclidean distance as the general distance metric. The approach in [128] uses Dynamic Time Warping (DTW) to cluster the raw data time series. The authors not only warp multiple time series but also provide a technique to find the average time series representing the cluster center, which they term as the prototype. All feature-based approaches analyzed in [128] consider only single variables and are therefore not further compared. However, feature computation might be applicable on individual dimensions of multi-dimensional time series and the time series might be processed again as described in the raw-data approaches. Model-based approaches use for instance GMMs [129] or continuous HMMs [130], both employing the log-likelihood as distance metric. In the setting of multivariate data, dimensional dependence within the time series has been examined in [131]. Here, a Euclidean distance metric is either applied dimension-wise in so-called independent DTW or applied in a multidimensional manner in so-called dependent DTW.

One problem is to identify the number of clusters automatically, e.g., with the help of a distance parameter [132]. However, this parameter has to be specified by the system designer. Consequently, substituting the hyperparameter for the number of clusters with a distance parameter might lead to an unpredictable number of clusters depending on the variance of the input data.

2.4.4 Knowledge Representations for Decision Making

To allow a robot to make decisions during task execution, it might rely on a more complex knowledge representation as such a sequential arrangement of actions. Therefore, the research community came up with structured representations of tasks, for example in the form of task graphs [58, 83, 114, 133–135].

Some of these task graphs are implemented as decision trees while others use the concept of behavior trees. While decision trees are stateless, behavior trees are stateful, i.e. they can mark a behavior in a branch as currently running or failed, hence being able to skip the current evaluation or to select another behavior the next time. Branches in the tree are usually arranged from left to right according to their priority and the behavior in the branch that first has a valid condition is executed next. Another possible implementation for a task decision framework is a FSM, which does not require a tree-like arrangement of states, hence being able to represent loop closures of action sequences. Since a behavior tree allows to mark a branch as running or completed, it could also achieve the definition of loop closures by setting the running flag of a branch once started and resetting it once completed.

A decision tree implementation can also achieve the functionality of a behavior tree by adding pre- and post-conditions to the relevant branches [114]. This approach is exploited in the presence of humans, who might cause uncertainties or restructure the workspace on the fly. The approach employs visual perception and activates only branches of the tree that are feasible for the robot at the given environmental state. The collaborative robot programming framework in [136] uses augmented reality projections and a touch-enabled hardware table to intuitively parameterize an existing robot program. The program itself allows pre-defined branching or cyclic operations and the task graph appears to be a FSM.

It is also common to use a pure sequential task execution, where the robot executes a sequence of actions or skills [42, 72, 80]. However, a fixed sequence of actions is not able to solve conditional tasks, since it does not include re-planning or decision making on the task level. Although re-planning of a robotic task during execution is possible, a task planner requires a goal definition and world representation to work.

The framework presented in [134] proposes a robot state automaton which models a state for each discrete time step in the execution. The framework allows to observe environmental conditions and to branch into different states during execution. This thesis adapts the approach and employs it as a baseline method for comparison. It is adapted in a way to only create graph-nodes where a decision state is required in order to obtain a task graph. Hence, ordinary robot states within a trajectory are not represented as graph nodes. This allows visual representation of the task graph with only the relevant information for the user, which are the decision states. Similarly to the approach presented in this thesis, a task graph is incrementally constructed in a combined teaching- and execution phase. The main difference is that with [134], the user has to observe anomalies during execution of the task and needs to decide if and when a new demonstration is needed.

2.5 Intuitive Robot Programming Frameworks

This section examines intuitive programming frameworks with respect to four aspects: 1) Usability by end-users; 2) reuse of existing knowledge to reduce human programming effort and increase autonomy; and 3) automatic skill parameterization from demonstration data.

The first aspect concerns the usability for end-users. Well designed graphical programming interfaces allow a good user experience and should represent the robot program, or more generally, the knowledge that is available within the system in an understandable manner. Therefore, skills as graphically represented elements are considered in [137]. Commercially available solutions can be found in [138, 139]. Here, skills can be sequenced and parameterized on a Graphical User Interface (GUI), which is termed as OSP in the following. Lower-level representations for programming that construct finite state machines rather target domain experts and more experienced users to develop the robot skills itself [140, 141]. Many other approaches allow to define a task by the end-user without implementing task specific knowledge [26, 58, 72, 142, 143]. However, the aspect of usability and intuitiveness is often neglected. For example, in [58], tasks can be fully programmed by demonstration and also executed, but the task representation gives only a vague idea about what the robot will perform in the form of left and right arm movements. In [142], skill sequences are learned and executed but without providing human readable skill labels. A task representation in the form of a legible skill sequence is considered in [26] and [72]. In [72], a user study assesses the intuitiveness of the PbD method in comparison to manual programming on a GUI. In [144], user effort and discomfort are assessed in a visual demonstration framework. Another user study assesses the challenges that arise from redundant manipulators and how kinesthetic teaching can be used in task space and null-space [17]. Another limitation of some programming frameworks is that they focus on the programming of individual skills without considering scenarios that consist of skill sequences [18, 44, 145].

The second aspect concerns the reuse of existing knowledge. This is often tackled by common knowledge bases and semantic reasoning. Starting from unlabeled demonstrations, symbols can be assigned to detected skills to be used in task planning for semantic reasoning [87], which inverts the well-known problem of symbol grounding. A system can also query only missing knowledge by means of symbol grounding and reuse existing knowledge in the form of previously learned motion primitives as shown in [114]. Hardware abstractions for skills are presented in [48, 146] and allow a user to reuse a skill independent of the robot hardware. Later on, skills could be transferred to other systems without re-implementing them.

The third aspect concerns automatic skill parameterization from human demonstrations. The parameters itself can be extracted from the demonstration data such that no further human input is required. In comparison, the method of OSP uses a GUI that requires the manual parameterization of skills [137, 147]. Examples of automatic parameter extraction can be found for hybrid motion and force control in force-based skills [148], for task-parameterization techniques by using a demonstrated target frame as task parameter [149] or for visually extracted task parameters [150, 151]. Further, task parameters can be automatically extracted from trajectory data [152]. By using semantic skill models, parameter values can be also extracted by symbolic evaluation using a world model [72], for instance an object label can be used as parameter of a pick skill. Furthermore, parameter values can also be extracted by predefined functions that are mapped to the skill parameters, for example by computing the relative grasp transformation between robot gripper and object during demonstration [26].

2.6 Unsolved Issues

Most skill learning approaches assume that the underlying methods are pre-selected and parameterized by a domain expert, usually being the system designer or experimenter. This is not meant to be a drawback of the presented methods itself. Rather, it shows that appropriate methods and controllers exist that can be tailored for specific applications. The problem is that the method selection in a specific task must be made by a domain expert, which would prevent novice users to intuitively use them to setup new applications. Skill recognition would cope with this problem such that a suitable skill is automatically selected by the system, which maps a robot capability to each application specific behavior that is required in the task.

The result of a skill identification phase should be a set of predefined robot skills that a task planner or a human programmer can use to solve a robotic task. However, many frameworks simply propose a set of actions or skills that is task specific without being analyzed to be interpretable by humans. Predefining skills from the technical perspective might be a valid approach, if used for autonomous execution or in task planning approaches. Nevertheless, it is often desired that such skills are legible to describe a robot task in a way that is understandable to end-users.

When it comes to skill recognition, it can be often observed that the models used for inferring an action or a skill are trained on task-specific instances in the motion domain. So called task invariance is often striven for, such that actions can be classified by their motion characteristics with a broad range of variations. However, action characteristics in the force domain are often neglected, although the reign the behavior of contact skills.

The end-user does often not receive the required attention in the research landscape of LfD and PbD. Although there are many works about human factors and human machine interaction, there are too many methods that are reserved for experts because they are too complex to be understood by end-user.

2.7 Conceptual Differences

The main conceptual differences between this thesis and the state of the art are stated as follows.

i) Proven legibility of the proposed skill set

ii) Exclusive usage of proprioceptive data in the task definition and execution

iii) Bootstrapping of decision behaviors without world and object models

This thesis aims at identifying a number of contact skills and proves that the skill's descriptive names are legible and distinguishable by a variety of people without requiring a background in robotics. The identified skills are then used in PbD to be recognized automatically by the robot. Instead of using representations of previously learned motions, this thesis considers features that are derived from both the motion and force data to make the skill recognition independent from reference frames or specific environments. This implies that the recognition generalizes over different tools, materials or objects. Additionally, this thesis presents and approach that merges the advantages of data-driven and symbolic evaluation in skill recognition. With that, only the challenging inference problems are trained on demonstration data and trivial problems are evaluated by symbolic predicates.

This thesis focuses on the proprioceptive measurements of a robot as input modality and does not require visual perception in any of the presented methods. This focus has the advantage that a modern lightweight robot can easily serve as input device, without setting up additional sensors and hardware. A drawback is that visual perception is missing, which reduces the robot's capabilities. However, the intended usage of the proposed methods is rather seen in structured and semi-structured environments, as they occur in the industrial assembly and manufacturing domain.

The presented methods intend to bootstrap robot task only from human demonstration data, wherever applicable. This is also known as no-code robotics and no-code robot programming. With that, the end-user shall be able to define new robot tasks without writing a single line of code. This thesis aims to push the limits of PbD, such that also task decisions and recovery behaviors can be programmed just by demonstration without providing prior knowledge about involved objects and the environment.

This thesis proposes several methods for task definition that work without world representation and object models. Conceptually, this differs from works that require a large amount of prior knowledge about the task, making it infeasible to deploy them in unknown environments. This thesis presents methods that allow to define tasks in entirely unknown environments, as long as they are of static nature or at least partially structured.

Finally, a framework is proposed that unifies different task definition methods for end-users. While many frameworks propose a single, specific teaching or learning method to setup new tasks, the presented framework allows to combine PbD with another programming paradigm, where a GUI is used to manually define the task. Both task definition methods can be considered as no-code robot programming. This framework relies on an ontology to bundle the knowledge obtained from each source with prior knowledge that can be possibly predefined. This knowledge representation allows then to combine expert knowledge that is embedded in the ontology and skill implementations with the parameterization coming from the end-users.

3 Prerequisites and Background

This chapter introduces the prerequisites and the background for the subsequent chapters in Sec. 3.1. Section 3.2 states the demonstration encoding approach used in this thesis. Section 3.3 clarifies that human and robot use different perception modalities when dealing with demonstrations.

3.1 Definitions of the Programming by Demonstration Process

3.1.1 Demonstrations and Time Series

The task definition and learning approaches in the subsequent chapters of this thesis deal with human demonstration data. Usually, the end-user employs kinesthetic teaching to give a single demonstration of the task.

Definition 3 *A demonstration trial is a single human demonstration represented as a multidimensional time series. Synonyms for demonstration trial are demonstration, demo, and trial.*

A major distinction is made between skill identification and skill recognition. Skill identification is made prior of implementing skills.

Definition 4 *Skill identification is the method of labeling a set of distinguishable, generic, and reusable robot capabilities that are interpretable by humans and robots and that follow the definition of a skill (Definition 2).*

Skill recognition uses a set of predefined skill types to classify demonstration data.

Definition 5 *Skill recognition is the process of automatic inference about the performed actions that were observed in a human demonstration. During inference, a human readable label is provided for each recognized skill.*

The process of skill recognition can be performed on a dataset (offline) or during the demonstration on a stream of data (online). If it is applied on a sequence of demonstrated actions, it simultaneously solves the problem of data segmentation and classification. The input data of this process is defined as a multidimensional time series

$$\boldsymbol{X} = \left[\boldsymbol{X}^{(1)}; \ldots; \boldsymbol{X}^{(N)}\right] \in \mathbb{R}^{N \times D} \tag{3.1}$$

with N samples and D dimensions. The samples are vertically stacked in the matrix, which is denoted by a semicolon throughout this thesis. A demonstration sample $\boldsymbol{X}^{(t)}$ at time step t might contain a combination of measurements such as a pose and a wrench.

This thesis considers the demonstration data by the concept of modalities.

Definition 6 *A teaching modality describes the state space of physically related quantities and their derivatives. These quantities can be of continuous or discrete manner. Multiple teaching modalities can be observed simultaneously by different sensor in one demonstration.*

Naturally, the pose modality is recorded in LfD systems. This work expresses all poses in the world frame $\{\mathrm{O}\}$ if not other specified and defines a Cartesian pose $\boldsymbol{\xi}$ as

$$\boldsymbol{\xi} = [\boldsymbol{p}, \boldsymbol{o}] \in \mathbb{R}^7$$

with position $\boldsymbol{p} = [p_1, p_2, p_3]$ and orientation as quaternion $\boldsymbol{o} = [o_x, o_y, o_z, o_w]$. The scalar-last notation is used for quaternions with the vector part $[o_x, o_y, o_z]$ followed by the scalar o_w. Additionally, also the time derivatives of the pose are covered by the pose modality, such as the velocity $\dot{\boldsymbol{p}}$ and the acceleration $\ddot{\boldsymbol{p}}$. A time series of poses is expressed as

$$\boldsymbol{\Xi} = \left[\boldsymbol{\Xi}^{(1)}; \dots; \boldsymbol{\Xi}^{(N)}\right] \in \mathbb{R}^{N \times 7} .$$

This work puts a strong emphasis on the force domain and naturally considers force and torque data in the recognition methodology. The wrench modality is expressed in the world frame $\{\mathrm{O}\}$ if not other specified. This work defines the wrench $\boldsymbol{w}$ as

$$\boldsymbol{w} = [\boldsymbol{f}, \boldsymbol{\varrho}] \in \mathbb{R}^6 \tag{3.2}$$

with force vector $\boldsymbol{f} = [w_\mathrm{x}, w_\mathrm{y}, w_\mathrm{z}]$ and torque vector $\boldsymbol{\varrho} = [w_\mathrm{a}, w_\mathrm{b}, w_\mathrm{c}]$. A time series of wrenches is expressed as

$$\boldsymbol{W} = \left[\boldsymbol{W}^{(1)}; \dots; \boldsymbol{W}^{(N)}\right] \in \mathbb{R}^{N \times 6} .$$

The gripper state of the robot is defined as dedicated modality which is of interest when interacting with objects in this work. For a two finger gripper, it is expressed as

$$\boldsymbol{\vartheta} = [g, h] \in \mathbb{R}^2$$

with $g \in \mathbb{R}$ being the gripper finger opening width and $h \in \{-1, 0, 1\}$ representing the grasp status. The opening width g is a value in the range of $[0 \dots 1]$ where 0 and 1 represent the gripper to be fully closed or opened respectively. The grasp status is defined as

$$h = \left\{ \begin{array}{r} -1 : \text{no object in gripper} \\ 0 : \text{gripper moving} \\ 1 : \text{object in gripper} \end{array} \right\} .$$

A time series of gripper states is expressed as

$$\boldsymbol{G} = \left[\boldsymbol{G}^{(1)}; \dots; \boldsymbol{G}^{(N)}\right] \in \mathbb{R}^{N \times 2} .$$

Revisiting (3.1), the standard demonstration sample used in this work and if not other specified is denoted as $\boldsymbol{X}^{(t)}$ at time step t and it is defined as

$$\boldsymbol{X}^{(t)} = \left[\boldsymbol{\Xi}^{(t)}, \boldsymbol{W}^{(t)}, \boldsymbol{G}^{(t)}\right] \in \mathbb{R}^{15}, \tag{3.3}$$

consisting of pose, wrench, and gripper states. All samples are expressed in the world frame $\{\mathrm{O}\}$ if not other specified..

3.1.2 Multiple Demonstrations

Definition 7 *Multiple demonstrations are forming a set of demonstration trials (Def. 3) for the same task (Def. 1).*

Multiple demonstrations of a task use the index $i \in [1, \dots, I]$, where I is the number of demonstrations. Any variable $\boldsymbol{X}$ that is associated with demonstration i is denoted as $\boldsymbol{X}^{[i]}$. The t-th sample of this variable is referred to as $\boldsymbol{X}^{[i](t)}$.

3.2 Demonstration Encoding with Gaussian Mixture Models

This work encodes multidimensional time series with GMMs [153]. The resulting probabilistic models can be used to exploit their variance in confidence bounds and to reproduce a mean trajectory from multiple demonstrations. There exist state-based models, which reproduce an action conditioned on a state, and time-based models, which reproduce an action conditioned on a time variable. This thesis uses only time-based models, similarly to Dynamic Movement Primitive (DMP)s that rely on a phase variable.

A multidimensional time series $\boldsymbol{X}$ is encoded alongside a time vector $\boldsymbol{u} = [1, \dots, N]^T \in \mathbb{R}^N$. When encoding multiple demonstrations within the same model, they can be stacked along their temporal dimension while the same time vector is repeatedly stacked alongside. This technique exploits the spatial variance among multiple demonstrations, bringing a few advantages. First, it increases the robustness by merging the knowledge from multiple trials with task variations. Second, it smooths the noise inside jerky demonstrations. Third, it allows subsequent execution stages to exploit the model variance, for instance, to specify controller gains or to evaluate confidence bounds used in anomaly detection. Note that the length of each time series and time vector must match. Otherwise, resampled time series can be created to unify their lengths. Possibly, DTW can be applied to compute warping paths that temporally align the time series with each other.

Assume that a GMM encodes multiple time series $\boldsymbol{X}^{[i]} = [\boldsymbol{P}^{[i]}, \boldsymbol{O}^{[i]}] \in \mathbb{R}^{N^{[i]} \times D}$ with $i \in [1, \dots, I]$. Each time series i consists of position $\boldsymbol{P}^{[i]}$ and orientation $\boldsymbol{O}^{[i]}$. All time series have the same length $N = N^{[1]} = N^{[2]} = \dots = N^{[I]}$. The input data $\boldsymbol{R}^{\text{in}}$ consists of all stacked demonstrations alongside the time vector

$$\boldsymbol{R}^{\text{in}} = \begin{bmatrix} \boldsymbol{P}^{[1]} & \boldsymbol{O}^{[1]} & \boldsymbol{u} \\ \vdots & \vdots & \vdots \\ \boldsymbol{P}^{[I]}t & \boldsymbol{O}^{[I]} & \boldsymbol{u} \end{bmatrix} \in \mathbb{R}^{(I \cdot N) \times (D+1)} . \tag{3.4}$$

The Expectation Maximization (EM) algorithm fits a number of Gaussian distributions onto this input data [153]. The resulting model is defined as $\mathcal{M}_{\text{GMM}} = \text{EM}(\boldsymbol{R}^{\text{in}})$. The most important hyperparameter is the number of model components. This work defines the number of Gaussian components to be proportional to the temporal duration of the time series. This is denoted as $N_{\text{g}} \sim \frac{N\lambda}{f_{\text{s}}}$, where λ defines the number of model components per second and f_{s} defines the system's sample frequency.

GMR conditions the model $\mathcal{M}_{\text{GMM}}$ on the time vector $\boldsymbol{u}$. The results are stored in the generalized time series for mean $\boldsymbol{Y} \in \mathbb{R}^{N \times D}$ and covariance $\boldsymbol{Z} \in \mathbb{R}^{N \times D \times D}$ obtained by $(\boldsymbol{Y}, \boldsymbol{Z}) = \text{GMR}(\mathcal{M}_{\text{GMM}}, \boldsymbol{u})$. The samples at time t are specified as

$$\boldsymbol{Y}^{(t)} = [\boldsymbol{Y}_{\text{p}}^{(t)}, \boldsymbol{Y}_{\text{o}}^{(t)}] \in \mathbb{R}^{D}\,, \tag{3.5}$$

and

$$\boldsymbol{Z}^{(t)} = \begin{bmatrix} \boldsymbol{Z}_{\text{p,p}}^{(t)} & \boldsymbol{Z}_{\text{p,o}}^{(t)} \\ \boldsymbol{Z}_{\text{p,o}}^{(t)} & \boldsymbol{Z}_{\text{o,o}}^{(t)} \end{bmatrix} \in \mathbb{R}^{D \times D}. \tag{3.6}$$

The subscripts of $\boldsymbol{Y}$ and $\boldsymbol{Z}$ refer to each modality, with p for position and o for orientation. If the input data contained quaternions, the orientation trajectory quaternions in $\boldsymbol{Y}_{\text{o}}$ are normalized to unit quaternions afterward. The computations for the GMM and GMR are carried out using *pbdlib* [149].

3.3 Skill Transfer Modalities

The technically observable modalities of a robotic system are usually a subset of the modalities that a human actively uses during teaching, and they usually do not match the perception capabilities of the human. For instance, a human naturally employs visual perception, which might not be the capability of a robotic system.

Multiple sets of modalities are important for the task transfer from human to robot, which are defined as follows.

1. *Natural modalities*: Modalities of the physical world, e.g., pose, wrench, vision, acoustics, smell
2. *Teaching modalities*: Modalities that the human actively uses during the demonstration to solve the task efficiently, e.g., pose, wrench, vision
3. *Observable modalities*: Modalities that a technical system can observe with its sensors, e.g., pose, wrench
4. *Utilizable modalities*: Modalities that the system can utilize as feedback in its skill implementations and controllers, e.g., pose

3.3.1 Skill Transfer Example

An exemplary scenario suggests several *natural modalities* that are available from the physical world, which are vision, pose, wrench, acoustics, and smell (Fig. 3.1: *natural*). In this example, a human teacher uses kinesthetic teaching to demonstrate how to wipe a surface by employing a set of *teaching modalities* (Fig. 3.1: *teaching*). At this moment, a human could rely on

i) the control of the hand pose and proprioception of the arm kinematics (pose modality),

ii) the wrench that is perceived when pressing on the surface (wrench modality), and

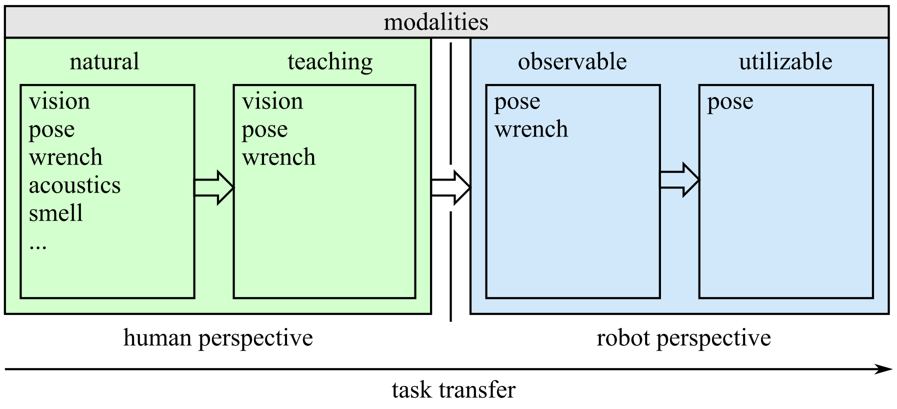


Figure 3.1: Sets of modalities during the task transfer from human to robot. From left to right, a human has natural modalities but might use only a subset of these as teaching modalities. A robot might be able to observe only a subset of these as observable modalities and might utilize a different modality subset during task execution.

iii) the visual perception used to approach the surface and track the wiping path (vision modality).

Since the acoustics and smell might not be task relevant for the human demonstrator, they are not contained in the *teaching modalities*. The robotic system can measure its pose and the applied wrench during kinesthetic teaching. However, being not equipped with a camera, the robot cannot visually observe the task and its environment. Therefore, the vision modality is missing in the *observable modalities* (Fig. 3.1: *observable*). Finally, the robotic system could learn a wiping strategy from the human demonstration. Assuming that the robot has no closed-loop force control capabilities, it could still execute the task with a position controller and a mechanical compliance mechanism at the end-effector. This example then leads to a robot-specific set of *utilizable modalities* without wrench modality and only a pose modality left (Fig. 3.1: *utilizable*).

This example showcases a possible mismatch between human and robot sensing modalities that might be caused by the hardware and software endowment of the robotic system. Consequently, robots that can only use a subset of the human teaching modalities must exhibit different strategies than humans to reach the same goals. Therefore, robot skills can be developed that consider the robot's limited capabilities but still can achieve the task goals as intended by a human. An example is a peg-in-hole task that can be solved visually and haptically by a human and only haptically by a robot without visual perception.

The mismatch between humans' and robots' capabilities requires the robot to understand which capabilities should be used in which context. This requires the identification of a set of skills that can account for different steps within robot tasks. Later, such skills can solve the part of the task as the user intended it, but with the given robot's capabilities. With that background, the next chapter proposes a method for identifying and automatically recognizing contact-based robot skills.

4 Skill Identification and Recognition

This chapter introduces a method for identification of contact skills, where the identified skills can be understood from the human and robot perspective.
A major part of this chapter was published in [154]:

- T. Eiband and D. Lee, "Identification of common force-based robot skills from the human and robot perspective," in *IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, 2021, pp. 507–513.

The author of this thesis developed the proposed methods and conducted the user study and experiments. D. Lee advised in developing research methodologies and analyzing the results and revised the article.

This chapter is structured as follows. Section 4.1 introduces the skill identification process. Section 4.2 explains how a robot recognizes these skills. Section 4.3 presents the experiments with the classification results on the underlying dataset. Finally, Sec. 4.4 discusses and concludes the main findings.

4.1 Skill Identification

A human can naturally select an appropriate strategy to achieve a specific task goal. A human also knows which constraints to consider when interacting with the environment. For instance, a human intends to grasp an object while the object is outside of the field of view. Usually, the human arm moves with adapted velocity toward the expected object location while trying not to hit any obstacles. The arm's feed-forward motion is controlled with low stiffness to reduce possible undesired impacts. When the hand touches the object, the human tries to minimize the contact force such that the object is not unintentionally shifted or dropped. Next, the human searches for a stable grasp pose by tactile exploration. All these steps require specific and adaptive control strategies that account for different constraints. The human can select an appropriate interaction strategy subconsciously from the made experiences. However, a robot cannot naturally rely on such experiences when executing a task.

With that in mind, experts design robot skills that execute a specific behavior with the help of inbuilt constraints. It is emphasized that the research community paid the most attention to robot skills intended to solve position-based tasks [58, 72, 112, 155]. From the human perspective, there is an intuitive need for contact forces in various human actions. Consequently, a robot should exploit contact forces and force feedback to interact with its environment.

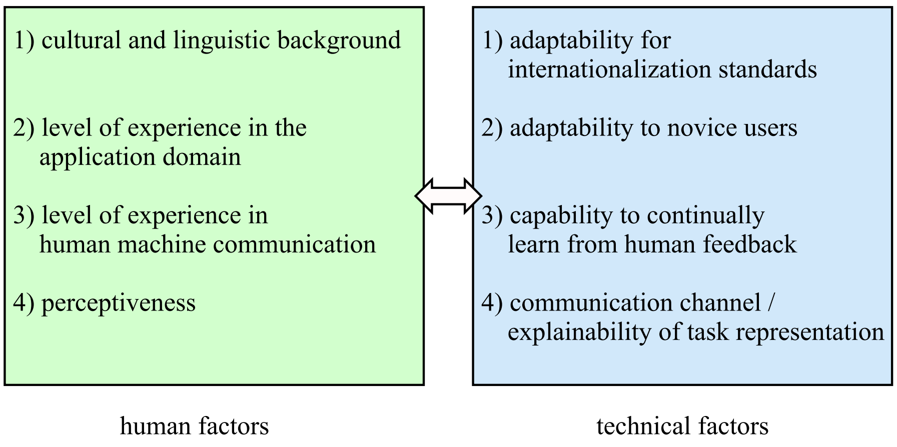

1) cultural and linguistic background
2) level of experience in the application domain
3) level of experience in human machine communication
4) perceptiveness
1) adaptability for internationalization standards
2) adaptability to novice users
3) capability to continually learn from human feedback
4) communication channel / explainability of task representation
human factors
technical factors

Figure 4.1: Human robot communication quality factors. Human factors depend on the individual end-user while technical factors depend on the robotic system characteristics.

Identifying skills that are interpretable by both humans and robots helps to improve the explainability of a technical system.

Definition 8 *Explainability describes how well a system can express itself to the system's user in terms of its knowledge and capabilities.*

Explainability concerns the technical side about how well a system was designed to let users understand what it does and why it is doing.

What precisely a human understands from a system's task representation in the form of skills could be influenced by many factors, such as shown in Fig. 4.1.

4.1.1 Human Factors

Fig. 4.1 shows the human factors: 1) cultural and linguistic background, 2) level of experience in the application domain, 3) level of experience in human machine communication, and 4) perceptiveness.

Considering point 1) the linguistic point of view greatly influences the legibility of skills. For example, a system shows a task representation with a skill labeled in a language the user is not proficient with. Therefore, a system designer must account for the cultural and linguistic background of the target group. Works that analyze the linguistic conception of forces in the English language [156–159] express only a weak connection to physical forces. The discussed usage of the word force and so-called force dynamics points instead to the field of cognitive linguistics and not necessarily to physical forces. For example, the expression "to force something" has only an indirect relation to a physical force. Instead of the works mentioned above that describe force dynamics as a concept from cognitive linguistics, skill identification aims to derive expressions for skills understood as force-based interactions in the physical sense. Although several works describe a set of force-based constraints [51, 160, 161], they do not validate them to which extent they are interpretable by users.

Considering point 2), the level of expertise in the application domain is crucial for qualitative human-machine communication. An example domain could be woodworking, where a carpenter would have a better imagination about the used terms, such as sawing, grinding, screwing, etc., than a novice user who has never worked in this area. Therefore, a system should ideally consider that the target group does not necessarily have a background in the application domain.

For point 3), level of experience in human-machine communication, it is assumed that people continually adapt to a technical system since a) the used terms reappear and become familiar; b) the human sees the consequences that a previous action caused; c) the user adapts to the system's communication behavior, even though some of its expressions are considered wrong in the beginning. Ideally, not only the user adapts to the system but also vice-versa. This behavior is also called co-adaption and would assume that there are learning agents on both sides.

Lastly, addressing point 4), perceptiveness refers to the human ability to understand what the system represents. Here, it is assumed that mental receptivity differs among users for the same task representation and communication channel. An example is that people have different perceptual strengths, which can be visual, auditory, or haptic. Consequently, some people would prefer a graphical task representation, while others would rely on speech and auditory feedback while teaching a robot.

4.1.2 Technical Factors

Fig. 4.1 lists important technical factors that influence the communication quality between humans and robots, which are: 1) cultural and linguistic background, 2) level of experience in the application domain, 3) level of experience in human machine communication, and 4) perceptiveness.

Point 1), adaptability for internationalization standards, refers to a machine's capability to switch to a suitable style of knowledge expression, such as a proper language for a textual task representation or appropriate speech that the user can understand.

A system should ideally adapt to the user's expertise, referring to point 2), adaptability to novice users. For instance, a system could provide hints in a graphical interface about how to be used in an introductory phase. Similarly, a user could start to use a system in a mode with reduced options and gradually move to a more sophisticated interface.

Point 3), capability to continually learn from human feedback, targets again the co-adaption between humans and robots but considers the technical aspect. While a system can incrementally learn to improve its performance generically, it could even adapt to the behavior of specific users by continually interacting with them.

Point 4), communication channel / explainability of task representation, addresses the style and quality of how a system can express itself. Different communication channels can be used to describe the system's knowledge, such as graphical or verbal feedback. Then, the style of how the knowledge is represented influences the explainability. For example, visual feedback can be expressed graphically through sequenced items, graph-like representations, or behavior trees. The labeling or description of these items can be achieved textually or graphically by pictograms.

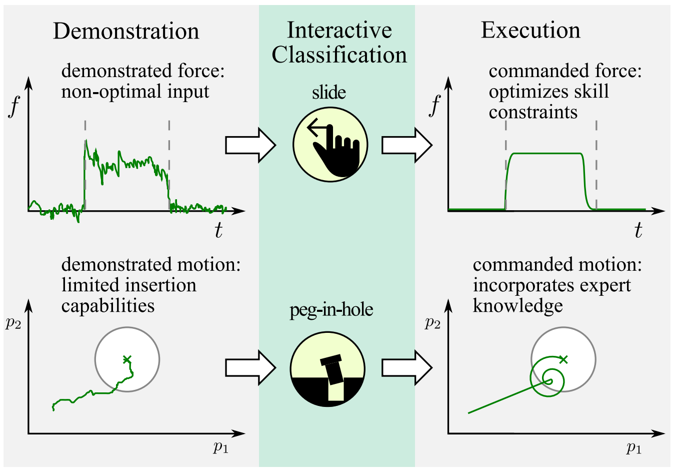

Demonstration
Interactive Classification
Execution
demonstrated force: non-optimal input
f
t
slide
commanded force: optimizes skill constraints
f
t
demonstrated motion: limited insertion capabilities
p_2
p_1
peg-in-hole
commanded motion: incorporates expert knowledge
p_2
p_1

Figure 4.2: Two examples for demonstrations and reproductions using the classified skill. Top: a demonstrated force profile (left) and possible reproduction based on detected *slide* skill (right). Bottom: a demonstrated peg-in-hole trajectory achieved with user's visual feedback (left) and possible reproduction of a *peg-in-hole* skill using a spiral search primitive to identify the insertion point (right).

4.1.3 Skill Constraints and Properties

The skill characteristics are formulated such that a hybrid force-position or force-impedance controller can reproduce them. A skill might have associated constraints during each phase of the execution. A constraint could be of kinematic or dynamic nature, such as described in the task frame formalism [161]. For instance, the robot controller would track the position in one subspace and track the wrench in another subspace [148].

The execution of a skill could further involve different states arranged in the form of a finite state machine. These states are either sequentially arranged or exploit a more complex architecture. Each state could then enable different control modes and control parameters. For instance, this is required when switching from position to force control after contact is made [162]. Each skill in this chapter also refers to multiple exemplary applications and related implementations from the state of the art. Such mapping between skill and implementation allows to reuse existing methods in the proper context without re-implementing robot behaviors repeatedly.

The following paragraphs introduce a set of skills that is analyzed throughout this chapter. At first, a skill is introduced that humans use for haptically identifying the state of the environment.

Touch is intended for a slight contact between the robot tool and the environment. Exemplary usages are to detect physical constraints in the world, the presence of expected objects, and for haptic search strategies [42, 96, 97, 163]. While the forces in this skill shall be limited, it might be desired to reproduce a specific force onto the environment.

Press achieves a desired force onto a counterpart, such as press fitting in assembly [37, 164, 165]. Pressing shall reach the desired force on a single point of application.

However, one could require to move a tool along a surface while maintaining the same force, as explained in the following.

Press and Slide enables polishing tasks [166], surface smoothing [34], grinding [167], or gluing [168] where usually a contact force normal to the surface is maintained. Tools can be also applied on free-form countours as part of many workpieces.

Contour targets to follow a free-form path, for instance, used in deburring tasks or edge smoothing of mechanical parts [169–172]. The requirement is to maintain a force normal to the edge such that the tool applies constant pressure onto the counterpart. Motion around a center of rotation is contained in many technical mechanism, requiring either the robot tool or a mechanical part to be rotated, depending on where the desired center of rotation lies in.

Turn aims for rotational motions, for instance, to screw or unscrew parts like a bottle cap, or to rotate a valve or lever [105, 173, 174]. More sophisticated robot behaviors are required for insertion tasks, also known as peg-in-hole tasks.

Insertion is designed for physically constrained mating operations, also referred to as peg-in-hole, where a variety of algorithms were presented [33, 145, 175]. A comparison of algorithms is presented in [176] and [177]. In a collaborative workcell, hand-overs between human and robot become possible.

Hand-over targets situations where an object shall be passed between robot and user. Examples can be found in [178], where actions such as “receiving” or “handing over” are defined. Such events can be, for instance, triggered by touching the robot’s structure. A robot can not only pass around or manipulate objects by grasping, but also by pushing them with any part of its structure.

Push can be employed in nonprehensile manipulation, meaning to handle objects without grasping them. For instance, used to shift an object over a table without using a gripper. A survey about robot pushing is presented in [179].

4.1.4 User Study for Skill Identification

The aim of this study is to identify human interpretable skill names for different human actions. These skill names are found by showing the users a video demonstration for different force-based interactions between the human hand or a tool within the human hand and the environment. In order to generalize over each of the previously introduced skills, two different videos were provided per skill. This results in 16 prerecorded videos that are referred to as video samples. Each video shows a force-based interaction between a human hand and its environment. An interaction can be achieved via a tool that is hold in the hand or directly via the human hand. Some examples are shown in the Figures from 4.3 to 4.10.

Each interaction class was shown in two different situations. For instance, touching was performed in the following two configurations: 1) the human hand holds a tool and touches one object (in this example, the tool is a pen as shown in Fig. 4.3); 2) the human uses the fingers directly to touch an object. Although the human holds a tool in the first configuration but not in the second one, a user might still consider that these actions belong to the same type. All video examples can be seen in the accompanying video of [154].

This study involved 26 subjects (10 female, 16 male, average age of 28.6 (SD = 4.04)). Prior to the study, they rated their level of robotics experience (RE) with an average of 2.35 (SD = 1.20)) in the range from 1 (no experience) to 5 (in-depth experience). Additionally, they rated their English skills (ES) with an average of 3.88 (SD = 0.71) in the range from 1 (basic) to 5 (native language). Each participant was instructed to propose a suitable textual label for each of the 16 videos via a browser interface. No restrictions were given about how to label a video, meaning that the user could either provide a single word or multiple words as designation.

The 26 subjects provided a textual label for each of the 16 interactions, resulting in 416 labels. The goal of this study was to find unique and human interpretable labels for each of the eight skills. To reach this goal, similarities have to be identified in order to maximize the number of matching designations. Therefore, all labels were pre-processed by lemmatization, employing the lexical database of WordNet [180]. Lemmatization makes use of a dictionary in order to remove the inflectional ending of a word. An example is the word *touching*, which results in the word *touch* after lemmatization. A more complex example are irregular verbs, such as *slide*, whose past tense is *slid*, which results in the word *slide* after lemmatization.

The results of the labeling study are shown in the bar charts of Figures 4.3 to 4.10. Each bar chart reports the 10 most frequently mentioned names. The users were rather consistent with labels for the *touch* skill, with 10 mentions of *touch*, followed by 4 mentions of *tap* 4.3. The press skill *press* was labeled 10 times with *press*, followed by two mentions of *push* 4.4. In the case of *hand-over*, 10 users mentioned the same skill name, while one user each labeled it as *give and take*, *grab*, and *pass*. The *insert* skill received also a rather obvious labeling with 10 mentions of the same skill name and only one mention each of the labels *stick in, fit,* and *place*. Another rather clear labeling received the *push* skill, with 10 mentions of the same name, three mentions of *slide*, two mentions of *move*, and one mention of *push object*.

Some of the skills received a larger variety of labels. The *press and slide* skill was termed five times *slide*, two times *press and slide*, and two times *stroke*. The *contour* skill was termed only three times with its expected name. Further, it received two mentions of *follow contour*, *follow edge*, *scrap*, and *trace* respectively. The *turn* skill received also a large variety of labels, where users mentioned only five times the label *turn*, four times *twist*, three times *screw*, and two times *rotate*. From these results, each skill is termed with the most frequently mentioned label. Figures 4.3 to 4.10 show the scenes of the interactions that were presented in the user study. The bar plot beside each figure shows the 10 most frequently mentioned labels.

4.1.5 User Study for Skill Classification

In this user study, the previously extracted labels are used as skill names in a classification task performed on the same video samples as the previous study. The difference to

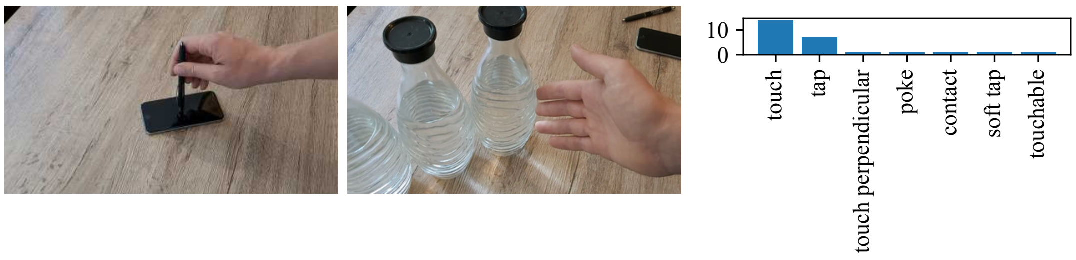


Figure 4.3: *touch*

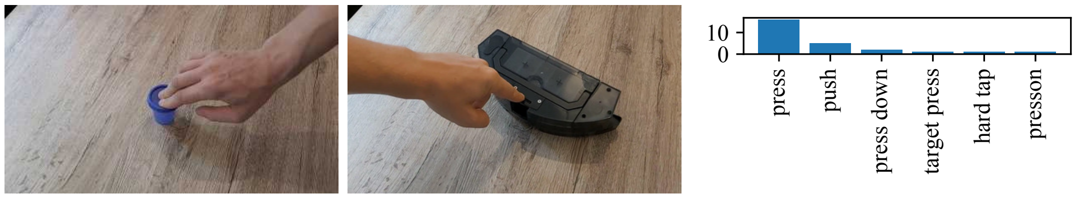


Figure 4.4: *press*

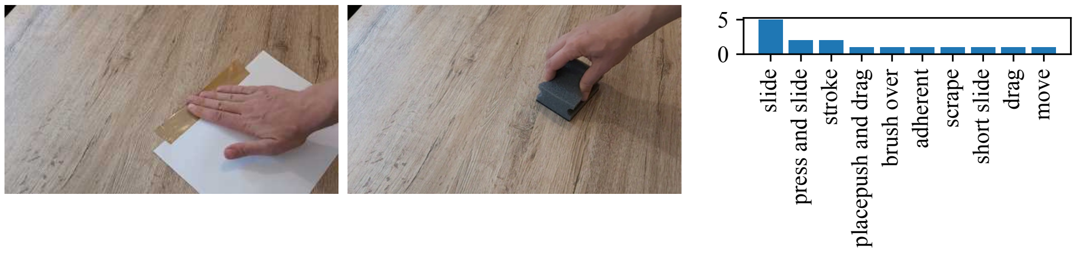


Figure 4.5: *slide*

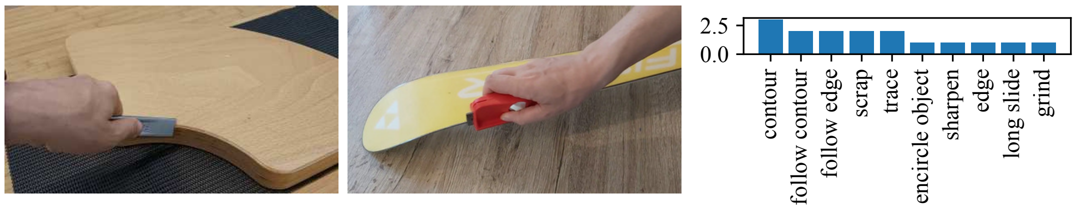


Figure 4.6: *contour*

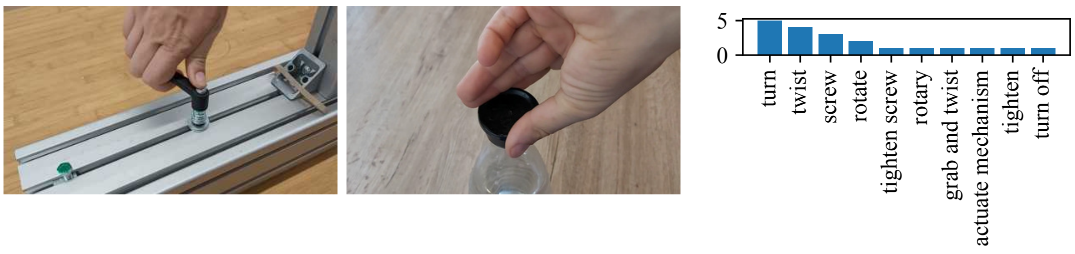


Figure 4.7: *turn*

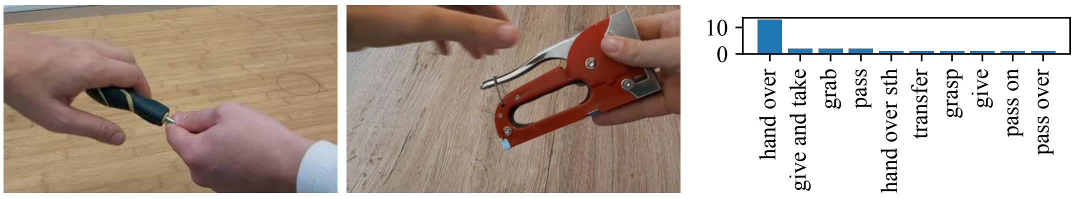


Figure 4.8: *hand-over*

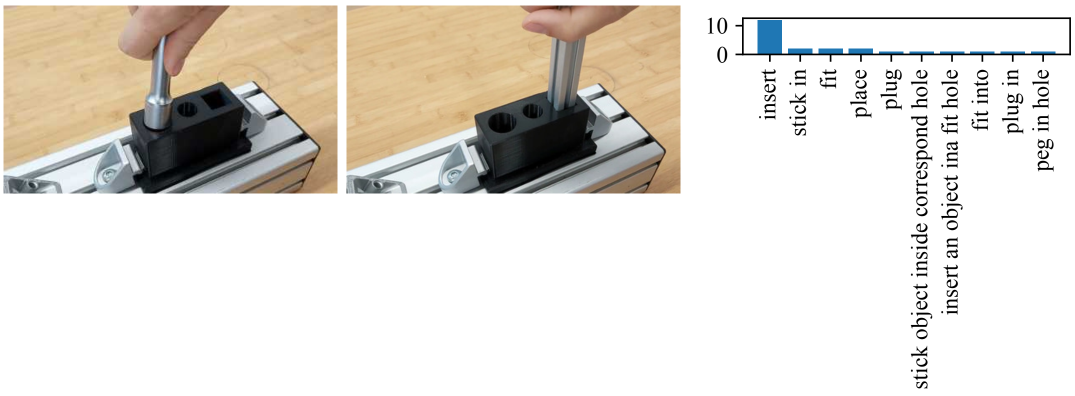


Figure 4.9: *insert*

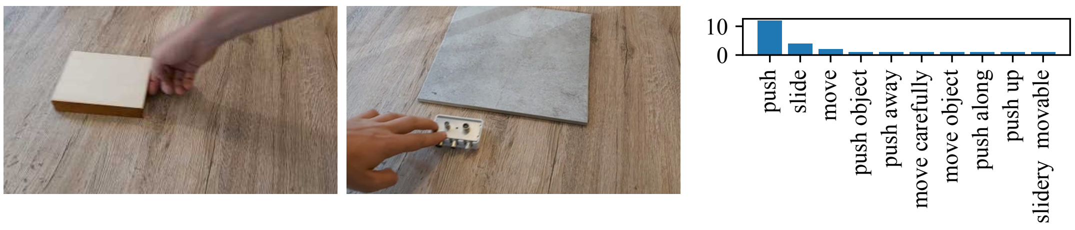


Figure 4.10: *push*

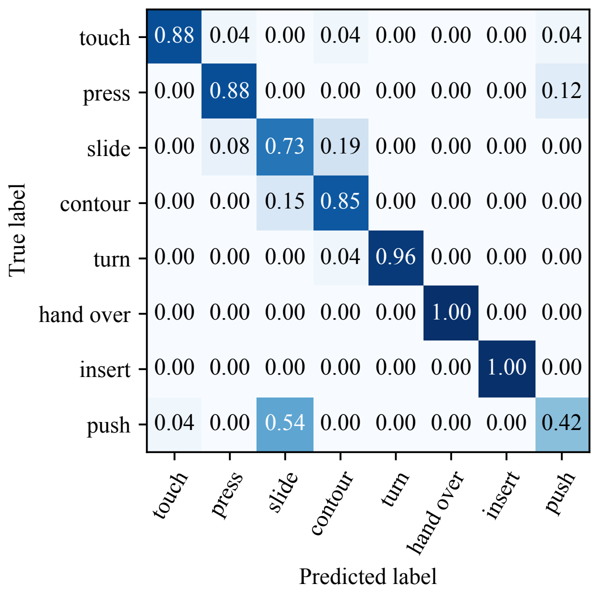


Figure 4.11: User study for skill classification: Confusion matrix of the human classified skills with original names.

the previous study is that now, the users could not freely provide a textual description but had to choose from a list of eight skill names that fit best to each sample.

The first user study about skill identification produced the most frequently mentioned labels for each skill, which are now used as labels in a classification task. A group of 15 subjects participated in this study (8 female, 7 male, average age 33.3 ($SD = 5.43$), $RE = 2.13$ ($SD = 1.13$), $EE = 3.13$ ($SD = 0.83$)). None of the subjects was involved in the previous user study. The subjects watched the same 16 videos as in the previous study and were asked to select one textual label that suits best to each video. The set of provided labels was {*touch, press, press and slide, contour, turn, insert, hand-over, push*}.

This study can be seen as testing a classifier. The classification model is represented by people's common comprehension of the classification task. Since the subjects could not communicate with each other and had never seen these video samples before, it is expected that there is a common understanding of force-based physical interactions embedded in the subjects' mental models and their linguistic cognition.

From the classification study, 15 users selected the labels for 16 video samples, resulting in 240 labels. The classification results are shown in the confusion matrix in Fig. 4.11. The average accuracy of the subjects' labeling is 0.89. In other words, the users could recognize about 9 out of 10 skills correctly. The *push* skill faced the lowest accuracy of 0.42. This means that less than half of the group could recognize its expected label. Moreover, the subjects most often confused it with *slide* (in the proportion of 0.54) as they might have associated the sliding of an object over a surface stronger with a *slide* skill. However, *slide* was intended for an interaction where pressure is applied onto a surface and maintained while moving a tool. After examination of this issue, it was assumed that the main confusion in the case of *push* and *slide* suggested proposing a different skill name for *slide* since it was even more frequently selected than *push* itself. Therefore, a different label for *slide* was used in a follow-up study for the skill classification.

4.1.6 User Study for Skill Classification with Modified Label

The third and final study revisits the skill classification task as it was designed in the previous study. However, the limited accuracy in the recognition of the *slide* that was mainly confused with the *contour* skill suggested to run a study with a modified skill label to possibly increase the legibility of the whole skill set.

The difference to the previous classification study is that the skill *slide* is renamed to *press and slide*. This modified label reached the second highest number of mentions, namely two, as shown in the bar-plot of Fig. 4.5. Although the label *stroke* also reached two mentions, it was hypothesized that *press and slide* has a higher interpretability than *stroke*. This study involves a number of 10 subjects (2 female, 6 male, average age 28.8 ($SD = 4.13$), $RE = 3.00$ ($SD = 1.51$), $ES = 3.75$ ($SD = 0.71$)). These subjects have neither participated in any other of the previous studies nor seen any of the video samples before the experiment.

The classification results are again shown as a confusion matrix in Fig. 4.21a. The average accuracy is reported as 0.89, which is comparable to the previous classification study. However, the main goal of this study was to improve the interpretability of the *push* skill, whose recognition accuracy could be increased from 0.43 to 0.70. This change also had a positive effect with respect to the *slide* skill that has been renamed to *press and slide*. Its recognition accuracy could be increased from 0.73 to 1.00, and the confusion between *push* and *press and slide* could be decreased from 0.54 to 0.2. In other words, only two out of 10 subjects confused these two skills compared to previously 14 out of 26 subjects.

Considering the improved accuracy of the push skill and the reduced confusion between *push* and *press and slide*, it is concluded that the skill labels are reasonably well interpretable by a general audience. Therefore, the final set of proposed skills is shown in Fig. 4.12 and a compact overview is given as {*touch, press, press-and-slide, contour, turn, insert, hand-over, push*}.

4.2 Robotic Skill Recognition

A human shall be able to interpret how the robot will behave when executing a skill. Analogously, a technical system shall be able to recognize a skill in a human demonstration. After identifying the skills, they can be considered commonly interpretable by both humans and robots and therefore serve as recognizable skill classes. Consequently, a skill recognition algorithm can infer the observed skill class from the demonstration data. Fig. 4.12 shows such concept of a skill demonstration and recognition system.

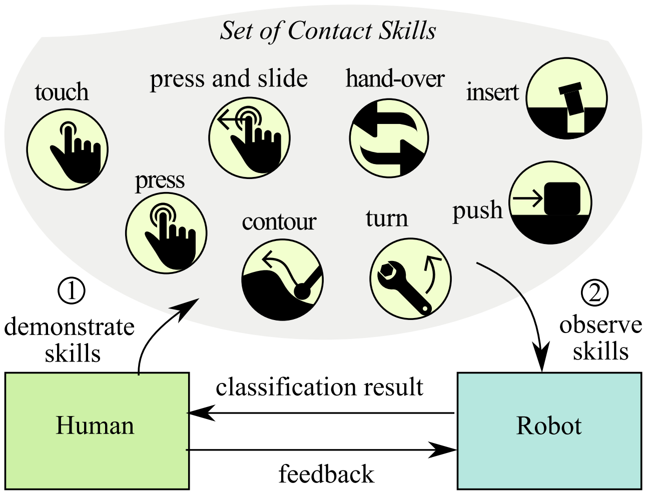


Figure 4.12: Concept overview of a PbD process where the human demonstrates and the robot recognizes skills.

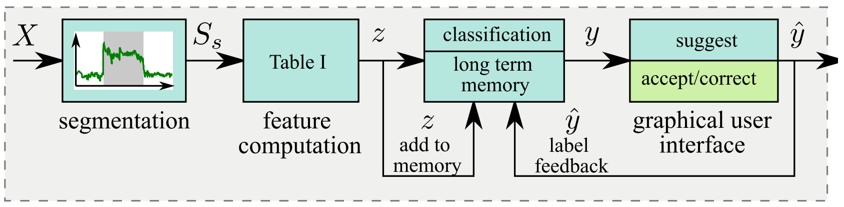


Figure 4.13: An overview of the system architecture and data flow.

First, a human provides a kinesthetic demonstration that is observed by the robot's proprioception. It is assumed that the provided demonstration contains skills the system can interpret. This process is called skill recognition employing a feature-based time series classification. It is a feature-based approach since it computes manually designed features from the raw measurements. The underlying problem is time series classification because the robot measurements that resemble a skill have the form of a multi-dimensional time series. This section introduces an algorithm for robotic skill recognition and proposes a system architecture used in the experiments. The components of this system are shown in Fig. 4.13.

First, a segmentation module observes the demonstration data $\boldsymbol{X}$, segments the time series based on the tool's contact state, and extracts so-called contact segments $\boldsymbol{S}_s$. A feature computation module (Sec. 4.2.3) reads these contact segments and outputs a feature vector $\boldsymbol{z}$ per segment $\boldsymbol{S}_s$. This feature vector has two purposes: 1) to predict a skill label y in the classification module (Sec. 4.2.4); 2) to serve as a data sample for the long-term memory of the classification module. The user can see the predicted label on a GUI. Finally, the system assumes either the user's agreement if no action is undertaken or accepts a manually corrected label y_{user}. This process is further described in the User Interaction Concept in Sec. 4.2.5. In case the user does not take any action, the final label $\hat{y}$ becomes $\hat{y} = y$, or in case the user corrects the label, it is $\hat{y} = y_{\text{user}}$. The final label $\hat{y}$ and its associated feature vector $\boldsymbol{z}$ is then added to the system's long-term memory.

4.2.1 Task Demonstration

The user demonstrates the task via kinesthetic teaching. The robot must be back-driveable and equipped with a wrist FTS, mounted between the robot and tool. The robot records the demonstration data time series denoted as $\boldsymbol{X} \in \mathbb{R}^{N_{\text{d}} \times 15}$ with N_{d} samples. It consists of position and orientation, force and torque, gripper finger distance, and grasp status. See Sec. 3.1 and Equation (3.3) for more details.

4.2.2 Contact State Segmentation

At this stage, the contact state segments the demonstration time series. It is expected that a task demonstration $\boldsymbol{X}$ consists of multiple alternating segments, which are: 1) unconstrained motion without tool contact (abbreviated as no-contact: NC); 2) tool contact with the environment (abbreviated as in-contact: IC).

The contact segmentation approach is introduced in more detail in Sec. 6.2.2. In brief, it logically combines threshold evaluation on absolute force and torque measurements to differentiate between free motion and contact state. The criterion for splitting the demonstration data is

$$\left(\left\|\boldsymbol{f}^{(t)}\right\| < \delta_f \wedge \left\|\boldsymbol{f}^{(t+1)}\right\| > \delta_f\right) \vee \left(\left\|\boldsymbol{\varrho}^{(t)}\right\| < \delta_\varrho \wedge \left\|\boldsymbol{\varrho}^{(t+1)}\right\| > \delta_\varrho\right),$$

given the force and torque vectors $\boldsymbol{f}^{(t)}$ and $\boldsymbol{\varrho}^{(t)}$ at time step t and their respective thresholds δ_f and δ_ϱ. The notation of $\|\cdot\|$ is the Euclidean norm and $\wedge$ and $\vee$ refer to mathematical conjunction and disjunction, respectively.

After the segmentation, there is a post-processing stage where all NC segments in between IC segments that are below the duration of 250 ms are removed. This rule exists for two reasons: 1) users have to deal with the robot's inertia during kinesthetic teaching and therefore cause undesired contact losses during a contact-rich operation. Examples are small lift-offs when the tool moves along a complex surface and the user cannot fully compete with the robot's inertia behavior. 2) the proposed algorithm assumes that the whole contact skill is demonstrated in one piece while the tool maintains contact with the environment. If the contact loss duration becomes too long, it lets a new NC segment emerge.

The segmentation module produces new segments online while observing the demonstration. A segment is denoted as $\boldsymbol{S}_s$ with $s \in \{1, .., M\}$ where M is the number of segments per demonstration. Whenever the system identifies that a contact state changes, the next segment $\boldsymbol{S}_s$ is passed to the feature computation module, introduced in the next section. Fig. 4.14 shows an exemplary demonstration with multiple contact segments.

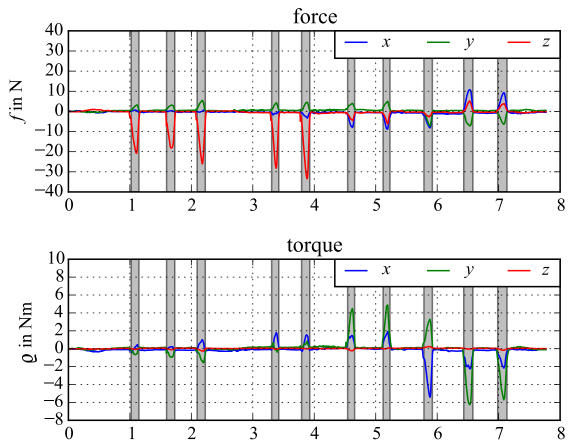


Figure 4.14: Segmented demonstration of multiple touch events, showing the in-contact segments in gray. From left to right over time, first the force and later the torque signals caused the segmentation to identify contact states.

4.2.3 Feature Design

This section introduces 30 features (Table 4.1) that are computed on each segment's motion and wrench data $\boldsymbol{S}_s$. The feature design required:

1. pose invariance, i.e. invariance to a task-specific reference frame without mattering in which Cartesian frame a skill demonstration lies;
2. shape invariance, i.e. invariance to various geometries of the contacted object or environment;
3. scaling invariance, i.e. invariance to the spatial scaling of a skill demonstration;
4. tool frame invariance, i.e. the feature computation does not assume a specific contact point on the tool, which can arbitrarily change during the demonstration.

In the following example, the number of each of the requirements above can be found in parentheses. A *contour* skill could be demonstrated in an arbitrary task frame (1.), has a motion trajectory whose geometric shape is not contained in the training set (2.), uses a larger spatial scaling than in the training data (3.). Additionally, the robot's tool follows a contour while its contact point constantly changes (4.). The classification method shall still be able to predict the correct skill while facing these challenges.

Table 4.1 shows the proposed features. The leftmost column contains an ID of each feature, referenced later in the feature analysis. The middle column shows the feature names trailed by a set of possible input variables. For example, "Path length $(\boldsymbol{p}, \boldsymbol{o})$" describes two independent features with IDs 1 and 2 using $\boldsymbol{p}$ and $\boldsymbol{o}$ as input, respectively. The rightmost column shows the feature equations, where $\boldsymbol{x}$ is a placeholder for each input variable. The dot notation $\dot{\boldsymbol{x}}$ refers to the first derivative over time. A segment comes with a number of samples N. The Euclidean norm is written as $\|\cdot\|$.

The proposed features consider the physical relation between the motion and wrench modalities. For instance, there are features such as *Work* (ID 17, 18), *No. of Zero-power*

Table 4.1: Contact Features

ID	Feature	Equation or Description
0	Duration	$N = \text{length}(\boldsymbol{X})$
1..2	Path length $(\boldsymbol{p}, \boldsymbol{o})$	$d_\text{x} = \sum_{t=1}^{N-1} \lVert \dot{\boldsymbol{x}}^{(t)} \rVert$
3..4	Path length ratio $(\boldsymbol{p}, \boldsymbol{o})$	$\frac{\lVert \boldsymbol{x}_N - \boldsymbol{x}_1 \rVert}{d_\text{x}}$
5..6	Distance $(\boldsymbol{p}, \boldsymbol{o})$	$\lVert \boldsymbol{x}^{(N)} - \boldsymbol{x}^{(t)} \rVert$
7..10	Time to max. $(\dot{\boldsymbol{p}}, \dot{\boldsymbol{o}}, \boldsymbol{f}, \boldsymbol{\varrho})$	$\underset{t}{\text{argmax}}\{\boldsymbol{x}^{(t)}\}$
11..14	Mean absolute value $(\dot{\boldsymbol{p}}, \dot{\boldsymbol{o}}, \boldsymbol{f}, \boldsymbol{\varrho})$	$\frac{1}{N} \sum_{t=1}^{N} \lVert \boldsymbol{x}^{(t)} \rVert$
15..16	Normalized sum $(\boldsymbol{f}, \boldsymbol{\varrho})$	$\frac{1}{N} \left\lVert \sum_{t=1}^{N} \boldsymbol{x}^{(t)} \right\rVert$
17..18	Work $((\dot{\boldsymbol{p}}, \boldsymbol{f}), (\dot{\boldsymbol{o}}, \boldsymbol{\varrho}))$	$w = \sum_{t=1}^{N} \boldsymbol{f}^{(t)T} \cdot \dot{\boldsymbol{p}}^{(t)}$
19..20	Mean power	$\frac{1}{N} w$
21	No. of Zero-power crossings (ZPC)	$\left\lvert \text{ZPC}(\boldsymbol{f}^{(t)T} \cdot \dot{\boldsymbol{p}}^{(t)}) \right\rvert$
22..23	Linear regression correlation $((\dot{\boldsymbol{p}}, \boldsymbol{f}), (\dot{\boldsymbol{o}}, \boldsymbol{\varrho}))$	coefficient of determination: R^2
24	Position linearity	v_e1
25	Position planarity	$v_\text{e1,e2}$
26..27	Relative spatial variance $(\boldsymbol{p}, \boldsymbol{o})$	$\frac{1}{d_\text{x}} \lVert \text{var}(\boldsymbol{x}_{1..N}) \rVert$
28..29	Relative wrench variance $(\boldsymbol{f}, \boldsymbol{\varrho})$	$\frac{1}{\max(\lVert \boldsymbol{x} \rVert)} \lVert \text{var}(\boldsymbol{x}) \rVert$

crossings (ID 21) and the *Linear regression correlation* (ID 22, 23) between position and force. Related works also apply so-called template matching, where a parametric model of prior demonstrations is learned and compared to the actual demonstration, making the model task-specific. The requirements above and the designed features intend to make the skill recognition invariant to specific environments.

Next, the computation of certain features is clarified in more detail. *Zero-power crossings (ZPC)* describes the number of events where the power in the physical sense changes its sign. This feature counts how often the energy flow between the robot and the environment reverses. *Position linearity* v_{e1} is the ratio of explained variance of the first principal component when applying a Principal Component Analysis (PCA) on the position trajectory. It is a metric describing how linear a trajectory is in the task space. Similarly, *position planarity* describes the ratio of explained variance in the PCA's first two principal components, which scores how well a trajectory lies in a two-dimensional plane in the task space. These variables are computed by

$$v_{\mathrm{e1}} = \frac{\mathrm{tr}(\boldsymbol{V}) - \boldsymbol{V}_{1,1}}{\mathrm{tr}(\boldsymbol{V})} \quad \text{and} \quad v_{\mathrm{e1,e2}} = \frac{\mathrm{tr}(\boldsymbol{V}) - \boldsymbol{V}_{1,1} + \boldsymbol{V}_{2,2}}{\mathrm{tr}(\boldsymbol{V})},$$

where $\boldsymbol{V}$ is the diagonal matrix of eigenvalues computed on the position $\boldsymbol{p}$, which are sorted in descending order. The *Linear regression correlation* is the coefficient of determination R^2 as computed in [181]. It is obtained by the multivariate linear regression function

$$\boldsymbol{Y}^{(t)} = f(\boldsymbol{X}^{(t)}, \boldsymbol{\beta}) + \boldsymbol{e}^{(t)},$$

where $\boldsymbol{Y}^{(t)}$ denotes the dependent variable, $\boldsymbol{X}^{(t)}$ denotes the independent variable, $\boldsymbol{\beta}$ are the estimated parameters, and $\boldsymbol{e}^{(t)}$ describes the error terms at time step t. More specifically, the features with ID 22 and 23 fit the force $\boldsymbol{f}$ as a function of linear velocity $\dot{\boldsymbol{p}}$, and torque $\boldsymbol{\varrho}$ as a function of angular velocity $\dot{\boldsymbol{o}}$.

A mutual information analysis applied to the feature set assesses their importance. It relies on non-parametric entropy estimation methods as proposed in [182] and computes a score of dependency between features and dependent variables in the dataset. Fig. 4.15 shows these results. The feature's score is then sorted in descending order. In an ablation study, the subsets of the k-best scoring features are used in a cross validation using the support vector classification described in the following subsection. Figure 4.16 shows the resulting classification accuracies converging to about 0.96.

4.2.4 Support Vector Classification

The examined problem is a multi-class classification with continuous features, which can be solved by an SVM. This algorithm was in favor of a Neural Network (NN) designed as Multi-layer Perceptron (MLP) that could not outperform the SVM in preliminary tests. The NN could perform better on more complex data without using the proposed features, but it might also require a much larger dataset and depend on data augmentation methods. There are other advantages of using an SVM, which are:

- simple optimization due to a small number of hyperparameters, which are usually kernel width and a regularization parameter;

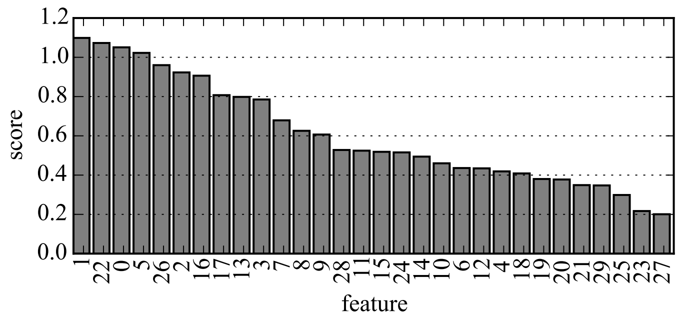


Figure 4.15: Score of each feature based on the mutual information analysis.

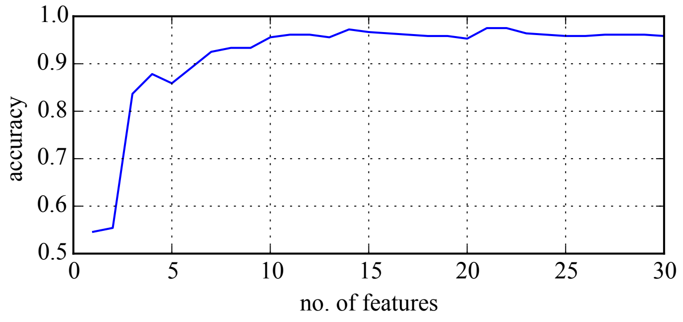


Figure 4.16: Classification accuracy over the reduced feature sets used in the SVM.

- a very short prediction time compared to NNs; and
- availability of confidence measures for each class' membership that allow scoring of the class assignment

In the training process, features that are computed on all contact segments are normalized to values between $[0, 1]$. The values of this normalization step are saved for the prediction step. Computing the features over a segment $\boldsymbol{S}_s$ yields a feature vector $\boldsymbol{z} \in \mathbb{R}^{30}$. The kernel of the SVM is chosen as Radial Basis Function (RBF) using the implementation of [183]. A hyperparameter search using a 5-fold cross-validation identified the regularization parameter of the training loss function to be $C = 100$. Finally, the trained SVM can predict the label y of the skills given a contact segment obtained from a human demonstration. Further, it can score the prediction concerning each class using a decision function $d_c(\boldsymbol{z})$ for a class c given the feature vector $\boldsymbol{z}$. The decision function returns the sample's distances to each separation hyperplane defined by the previously trained support vectors. This distance metric allows ordering the predicted class membership scores that the user can choose from.

4.2.5 User Interaction Concept

Whenever the user demonstrates a new task, the system segments the incoming data, computes the feature vector $\boldsymbol{z}$ on each contact segment, and predicts a candidate skill denoted as y. The candidate skill is then forwarded to the GUI to be presented to the user (Fig. 4.17 left). The user now has two options:

Accept The system's suggestion is added to the new task definition. This means there is consent between the system and the user about the skill to be specified, allowing the user to continue with the task definition ($y = \hat{y}$). Notably, the system can learn from the user's feedback. Accepting a suggestion is treated as an approval of the predicted label, which allows to add the unseen sample $\boldsymbol{z}$ and accepted label $\hat{y}$ to the system's long-term memory (Fig. 4.13).

Correct When the user is unsure about the proposed skill or is concerned that a misclassification occurred, a correction can be made. At this moment, a new skill $\hat{y}$ is selected from the proposed list ordered by the skill's class membership scores (Fig. 4.17 right). Examples of skill decision scores are shown in the experiment section in Fig. 4.22(b) and Fig. 4.23(b)). This scheme comes with the advantage that the user inherently labels new data. Analogously as in the case above, a new sample $\boldsymbol{z}$ plus associated label $\hat{y}$ is added to the system's long-term memory.

The system's prediction capabilities can be improved over time by retraining the classifier given newly labeled samples. In this case, a relatively small dataset is sufficient to bootstrap the system, which were 400 manually labeled contacts. With this interactive scheme, a new sample and label is obtained after each prediction because both actions of *accept* and *correct* achieve that. The burden for the user is relatively small because only a small fraction of misclassified predictions must be actively labeled.

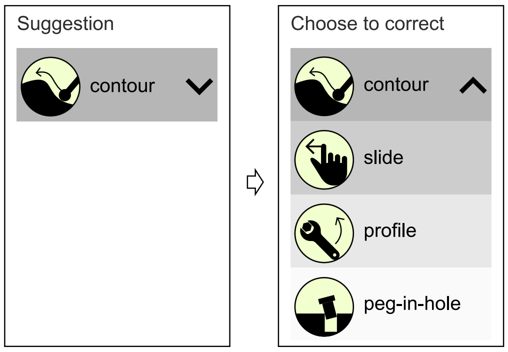


Figure 4.17: First, the system suggests a skill on the interactive GUI (left). Next, the user corrects the system's suggestion by opening the drop-down list. Now, probabilistically sorted suggestions for alternative skills are presented to the user (right). The user finally selects a skill to correct the system.

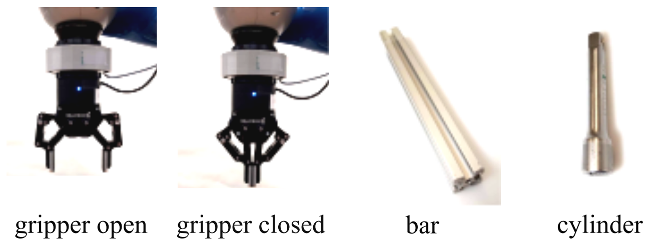


Figure 4.18: The tools used to collect the dataset. Bar and cylinder have been grasped by the gripper during demonstration.

4.3 Robot Experiments

4.3.1 Dataset

The dataset that facilitates this research is named KROCO (Kinesthetic Robot Contacts), which is publicly available[1] It contains a collection of demonstrated contact segments for each introduced skill. Figure 4.18 shows a variety of tools and Fig 4.19 shows different environments used in the data collection process. The dataset consists of about 400 labeled interactions and exemplary samples for each skill can be seen in the accompanying video of [154]. The bar and cylinder tools in Fig. 4.18 have been grasped lengthwise to point away from the gripper during all trials. Trajectories were demonstrated at varying locations and orientations of the task space. See Fig. 4.20 for examples where either the tool or the task frame varies while a circular tool is in contact with an object. Furthermore, the skill demonstrations in the dataset exhibit a large variance in temporal length, spatial scaling, and contact forces. For instance, there were demonstrations on multiple surface types with varying friction coefficients for the *slide* skill. The skills of *contour* and *push* were demonstrated on the wood form and the box.

[1]Description: https://teiband.github.io/KROCO/
Dataset: https://dx.doi.org/10.21227/nhea-gt59

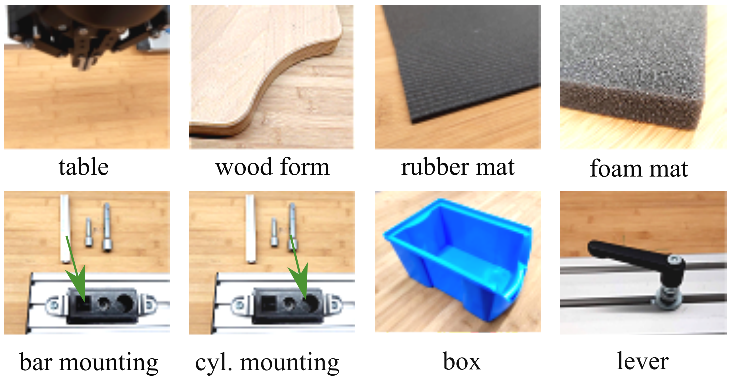


Figure 4.19: Different environments in dataset collection.

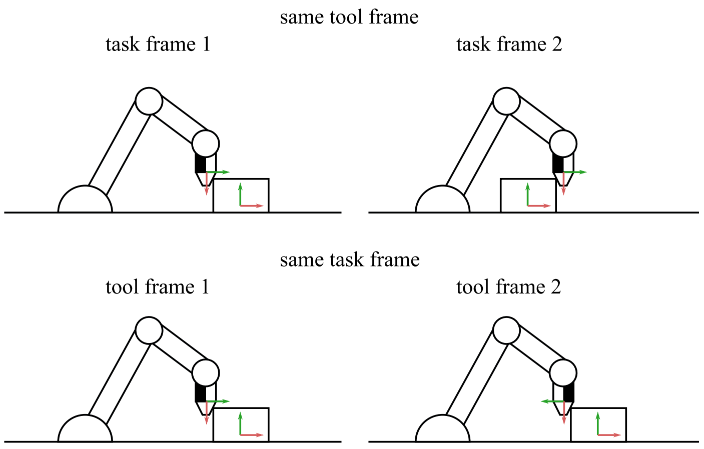


Figure 4.20: Examples about varying tool frames and task frames where the contact point on the tool can lie anywhere on the circumference of a circular tool, such as a cylinder or a cone.

The *turn* skill was demonstrated on lever rotations (see lever in Fig. 4.19). The user interactions for the *hand-over* skill were recorded by holding the end-effector with one hand and touching it with the other hand, leading to a small position displacement.

4.3.2 Robot Classification Results

As introduced in Fig. 4.13, the demonstration is segmented into free motion and contact segments. Next, the feature vector is computed over each segment, leading to a stacked feature matrix of $\boldsymbol{Z} \in \mathbb{R}^{400\times30}$ with associated label vector $\boldsymbol{y}^* \in \mathbb{R}^{400}$. After that, the described SVM classifier is trained on feature matrix $\boldsymbol{Z}$ and label vector $\boldsymbol{y}^*$. The classifier is evaluated offline with the dataset based on a 5-fold cross-validation. The accuracy reaches 0.96 with a training time of $1.39\,s$ and an overall prediction time of $3 \cdot 10^{-4}\,s$. Noticeable is the fast training and prediction time that allows to run the algorithm in online scenarios.

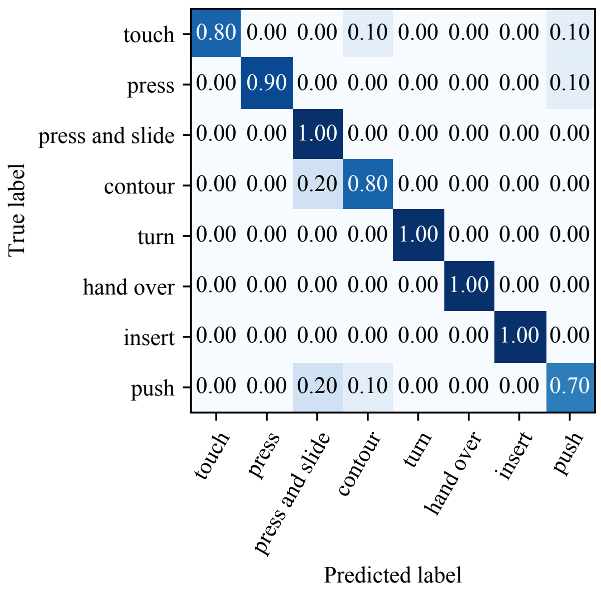
Fig. 4.21b shows the confusion matrix of the robot's classification performance. The classifier's accuracy of 0.96 is higher than the human's performance with an accuracy of 0.88. The human confusion matrix is shown below in Fig. 4.21a. Both confusion matrices show obvious similarities, suggesting that humans and the robot perceived the same characteristics of the skills. Interestingly, both the robot and the group of subjects share a stronger confusion concerning the *contour* skill. It is mostly confused with the *press and slide* skill in 0.20 of the robot cases and 0.27 of the cases for the human. This particularity suggests that a high similarity in the motion and wrench data corresponds to a high similarity in the human's visual perception.

4.3.3 Online Classification with User Feedback

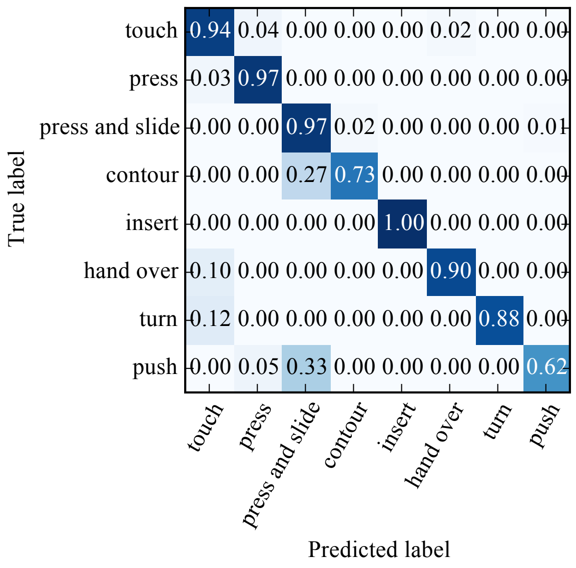
This section handles the classification of unseen interactions not contained in the dataset. The generalization capability is therefore tested with tools and environments that are not contained in the dataset.

Insertion A new cylindrical tool was introduced for the *insertion* task, which has different diameter ($d = 18\,\text{mm}$) and length ($l = 75\,\text{mm}$) as the cylinder tool ($d = 24\,\text{mm}$, $l = 125\,\text{mm}$) used in the dataset. Additionally, the mating geometry of the hole leads to a tight fit with the new tool. The hole's insertion depth is shorter (25 mm) than the one from the training set (50 mm). See Fig. 4.22a for the experimental setup. The classification predicted the skill *insertion*, with the skills' decision function scores shown in Fig. 4.22b. Fig. 4.22c shows the experimental position and force data.

Screw Tightening A screw-tightening task evaluates the rotation of an object with the *turn* skill. The demonstration tool is a grasped socket wrench insert of 13 mm used to tighten a metric M8×20 screw (see Fig. 4.23a. This task differs from previously shown motions and force profiles in the dataset, including no screw-tightening samples. Instead, the *turn* skill was trained on lever rotations (see the lever in Fig. 4.19). Although screw tightening and lever rotation are different in their motions, they share common features,

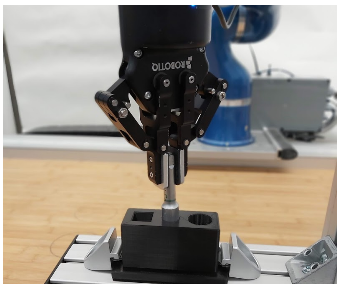


(a) User study with modified label: Confusion matrix of the human classified skills with name modification.

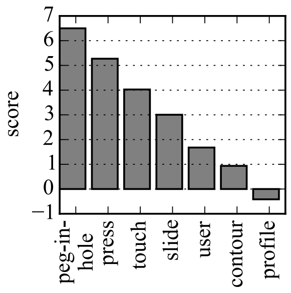


(b) Confusion matrix of the robot's feature-based time series classification.

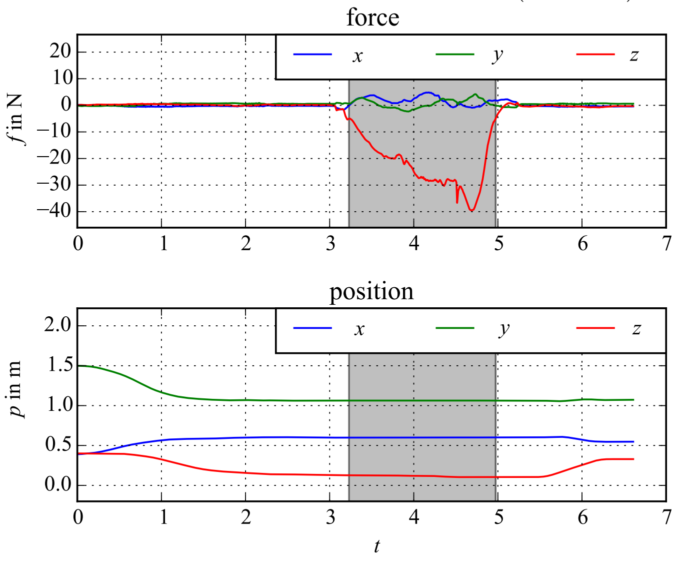
Figure 4.21: Confusion matrices for the human (a), (b), and machine classification (c).



(a) The unseen test tool and its hole for the *insertion* skill detection.

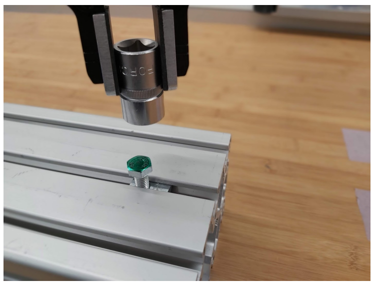


(b) The decision function scores of the classification (Sec. 4.2.4).

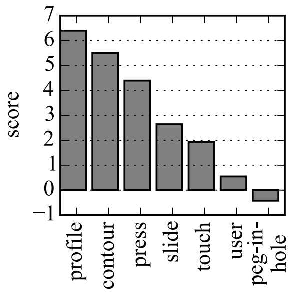


(c) Force and position trajectory with in-contact segment (grey) found by the contact state segmentation (Sec.4.2.2) (Sec. 4.2.4).

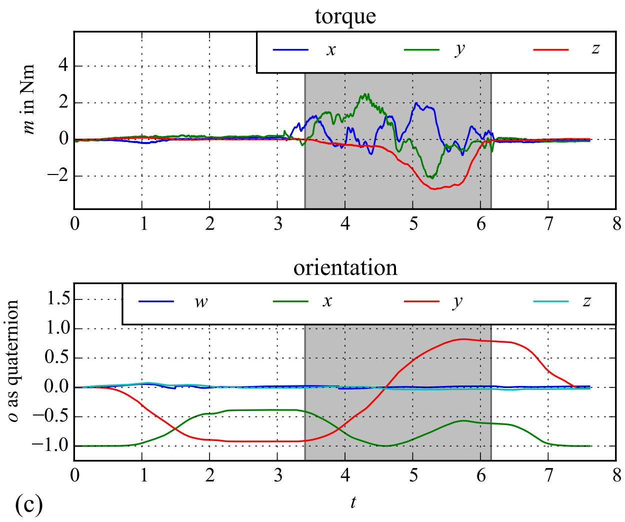
Figure 4.22: Peg-in-hole insertion experiment.



(a) The unseen test setup for the robotic screw tightening as *turn* skill.



(b) The decision function scores of the classification (Sec. 4.2.4).



(c) Torque and orientation trajectories with in-contact segment (grey) found by the contact state segmentation.

Figure 4.23: Screw tightening experiment.

for example the applied torque in conjunction with angular velocities. The classification predicted the skill *turn*, with the skills' decision function scores shown in Fig. 4.23b. Fig. 4.23c shows the position and force trajectories of the demonstration, including the segmentation results.

4.4 Discussion and Conclusion

4.4.1 Discussion

Comparing the results of the human and robot classification in Figure 4.21 shows that the *push* skill had a lower accuracy in the initial classification in Fig. 4.11. In the human classification case, this might be caused by the fact that objects were pushed along the table, which could also be interpreted as sliding an object over a surface.

From the linguistic perspective, it suggests that the English verbs of *push* and *slide* are interpreted similarly. Physically, both skills require a constant force while moving along a path. However, changing the name from *slide* to *press and slide* improved the interpretability on the human side (Fig 4.21a). Since misclassifications likely occur in a real system, the robot could suggest skills to the user via the interaction concept. The user has the option to accept or correct each of it. The corresponding class labels could then be used to incrementally retrain the classifier with new incoming data in a continual learning setup.

This research about skill identification and recognition from the human perspective is constrained to the English language. However, the proposed strategy for skill identification could be used to extract the same information in other languages.

4.4.2 Conclusion

The proposed set of contact skills can be interpreted by users in the role of robot programmers, as evaluated in a user study. The proposed classification method enables the robot to classify force-based interactions from only the motion and force data without requiring additional sensors. Contact skill detection is an essential direction in LfD, which shifts the burden of defining specific constraints or the parameterization of a controller for a particular behavior from the user to the system. As a prerequisite, a system designer would implement the required behaviors as skills to reproduce the desired behaviors. A system distributor could then ship a skill library along with the hardware. The proposed interactive classification scheme uses a feedback mechanism about the detected skill, which has the potential to improve the system's performance. It further enhances the system's transparency and explainability and lets the user understand the system's capabilities.

Successfully recognizing a basic set of skills paves the way to detect a wider variety of specific contact-based skills from motion and force data, for instance, snap-fit or press-fit connections or other surface processing techniques such as brushing, polishing, or sanding. An open issue at this stage is the automatic extraction of control parameters and compliance frames from the detected skills. Additionally, the contact skills should be recognizable within manipulation tasks that require unconstrained manipulation skills such as pick and place.

5 Skill Sequence Recognition

The previous chapter concerned the recognition of individual skills from a demonstration. In this chapter, recognition of a sequence of skills is considered as task definition method. The main idea of the presented skill recognition pipeline is that symbolic and data-driven approaches are merged in an online fashion to provide immediate feedback to the user during the demonstration phase.
This chapter includes the content of the following publication [184]:

- T. Eiband, J. Liebl, C. Willibald, and D. Lee, "Online task segmentation by merging symbolic and data-driven skill recognition during kinesthetic teaching," *Robotics and Autonomous Systems*, vol. 162, p. 104367, 2023.

The author of this thesis proposed the concept of merging symbolic skill recognition, using pre and post-conditions, with data-driven skill prediction, using support vector machines for skill classification. J. Liebl designed the merging algorithm that uses a segments pool, implemented it, and conducted the user study. C. Willibald supported the software development as well as the user study. D. Lee advised in developing research methodologies and analyzing the results and revised the article.

This chapter is structured as follows. Section 5.1 introduces the fundamentals, Sec. 5.2 proposes the method, and Sec. 5.3 shows the experiments. Section5.4 presents a user study that evaluates the intuitiveness of the method for end-users, and Sec. 5.5 concludes the findings.

5.1 Fundamentals

PbD is intuitively understandable to non-experts and requires less background knowledge in robotics or traditional programming [185]. While PbD allows non-experts to program robots, it does not imply that all human demonstrators are good teachers. The demonstrations non-experts give to machines are often sub-optimal [186], which stems from a mismatch between the mental model that the users have of the robot and the robot's real knowledge [187]. Therefore, it is helpful to provide the users with feedback about what the robot has learned to improve their teaching capability [188]. Furthermore, an understandable representation of the robot's knowledge allows it to explain itself. This kind of explainability helps users to build trust in robots, which is necessary to make non-experts feel at ease when using robots and gives them the insight required to debug incorrect robot programs [189].

Skill recognition is defined as a technique to segment demonstration data and classify the resulting segments to obtain a sequence of meaningful, understandable steps that are

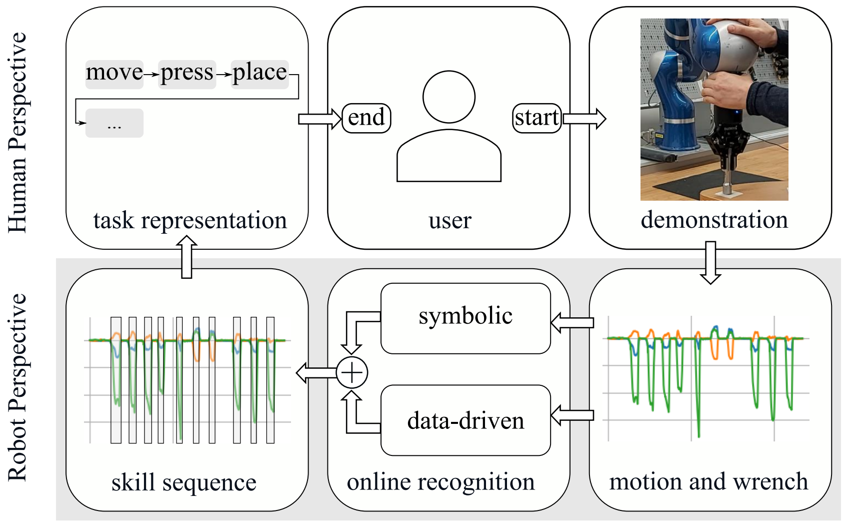


Figure 5.1: An overview of the information flow in the framework. The user provides a demonstration segmented by the system to recognize appropriate robot skills. At the same time, the user can monitor the evolving task representation that is editable at any time.

represented as robot skills. This is achieved by solving a segmentation and classification problem simultaneously. The combination of symbolic and data-driven skill recognition extracts a rich task representation consisting of manipulation and force-based skills. Fig. 5.1 shows an overview of this approach, where the user demonstrates a task while monitoring how the robot builds up its knowledge. This chapter proposes an approach for task segmentation that combines symbolic evaluation with supervised data-driven methods to recognize manipulation and contact skills. With that, the system builds a task representation during kinesthetic teaching. This task representation reflects how the system interprets the demonstration and helps the user to understand the robot's knowledge. Furthermore, it offers future options to correct it and thus supports non-expert users in their role as programmers.

The framework in this chapter:

1. reduces the amount of training data by 'outsourcing' simple skills like pick and place, which can be easily described in terms of symbolic preconditions/effects;
2. exploits only proprioceptive data and end-effector forces while avoiding vision-related issues such as problematic lighting conditions, occlusion, and inaccuracies due to calibration;
3. segments the demonstration online by two parallel segmentation pipelines;
4. reduces over-segmentation of data-driven classifications by the usage of a segment pool and reconsideration of combined segments candidates;
5. provides live feedback in the form of a human-readable task representation; and

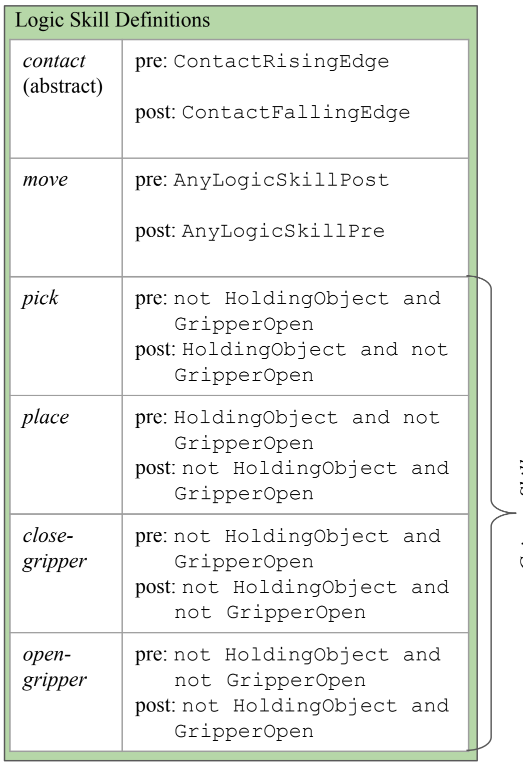
6. enables the user to debug the graphical task representation via a graphical user interface, evaluated in a user study.

5.2 Online Skill Recognition

The information flow in the framework is shown in Fig. 5.1. The process starts with a user that demonstrates a task to the system. The robot observes the human demonstration and collects samples $\boldsymbol{X}^{(t)}$ as defined in eq. (3.3) at each time step t. The output of the algorithm is a sequence of skill labels along with their corresponding demonstration data segments.

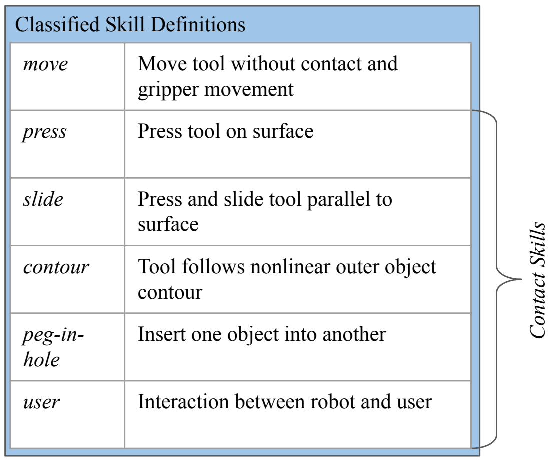
5.2.1 Skill Definition

There are two sets of skills defined, which are called *Logic Skills* and *Classified Skills* (Fig. 5.2). *Logic Skills* can be symbolically described (Fig. 5.2a). These are designed



Logic Skill Definitions	
contact (abstract)	pre: `ContactRisingEdge` post: `ContactFallingEdge`
move	pre: `AnyLogicSkillPost` post: `AnyLogicSkillPre`
pick	pre: `not HoldingObject and GripperOpen` post: `HoldingObject and not GripperOpen`
place	pre: `HoldingObject and not GripperOpen` post: `not HoldingObject and GripperOpen`
close-gripper	pre: `not HoldingObject and GripperOpen` post: `not HoldingObject and not GripperOpen`
open-gripper	pre: `not HoldingObject and not GripperOpen` post: `not HoldingObject and GripperOpen`

(a) *Logic Skills*



Classified Skill Definitions	
move	Move tool without contact and gripper movement
press	Press tool on surface
slide	Press and slide tool parallel to surface
contour	Tool follows nonlinear outer object contour
peg-in-hole	Insert one object into another
user	Interaction between robot and user

Contact Skills

(b) *Classified Skills*

Figure 5.2: Logic and classified skill definitions.

by a human understandable precondition (pre:) and postcondition (post:). It is known that manipulation skills such as *pick* and *place* are commonly used in robotics [65, 67], leading to the set in Fig. 5.2a. The abstract skill *contact* is recognized by evaluating its pre- and post-conditions, which are described by the symbols `ContactRisingEdge` and `ContactFallingEdge`. Their binary values are obtained by

$$\texttt{ContactRisingEdge} = \left(\left\|\boldsymbol{f}^{(t)}\right\| < \delta_f \wedge \left\|\boldsymbol{f}^{(t+1)}\right\| > \delta_f\right) \vee \left(\left\|\boldsymbol{\varrho}^{(t)}\right\| < \delta_\varrho \wedge \left\|\boldsymbol{\varrho}^{(t+1)}\right\| > \delta_\varrho\right),$$

and

$$\begin{aligned}\texttt{ContactFallingEdge} = \bigg(&\left(\left\|\boldsymbol{f}^{(t)}\right\| > \delta_f \wedge \left\|\boldsymbol{f}^{(t+1)}\right\| < \delta_f\right) \wedge \boldsymbol{\varrho}^{(t+1)} < \delta_\varrho\bigg) \vee \\ &\bigg(\left(\left\|\boldsymbol{\varrho}^{(t)}\right\| > \delta_\varrho \wedge \left\|\boldsymbol{\varrho}^{(t+1)}\right\| < \delta_\varrho\right) \wedge \boldsymbol{f}^{(t+1)} < \delta_f\bigg),\end{aligned}$$

with the force and torque signals $\boldsymbol{f}^{(t)}$ and $\boldsymbol{\varrho}^{(t)}$ at time step t that are compared to thresholds δ_f and δ_ϱ respectively.

These symbols are evaluated by checking if the absolute force and torque measurement exceed or fall below a threshold. These thresholds were set to 5N for force and 2Nm for torque based on preliminary experiments and as inspired by [42]. The skill is marked as abstract in Fig. 5.2, which means that it will only appear in the processing pipeline and finally be refined by the algorithm, which is explained in more detail later. A *move* skill defines a demonstration section where the end-effector is in motion without physical contact while the gripper fingers are not actuated. The precondition `AnyLogicSkillPost` is automatically triggered when the preceding skill reached its postconditions. Similarly, the postcondition `AnyLogicSkillPre` is automatically triggered when any subsequent skill has valid preconditions.

Consequently, a *move* skill fills gaps in the time series between other available *Logic Skills*. The rest of the described skills involve gripper operations and are termed *Gripper Skills*.

Classified Skills (Fig. 5.2b) are challenging to describe by manually defined rules. The presented set is mainly based on the identification in Chapter 4. The *move* skill is also part of this set because a classifier can better predict free motion and contact as a symbolic predicate can do, which only evaluates a fixed threshold in the force domain. The skills in the group labeled as *Contact Skills* perform a force-based interaction with the environment. Since these interactions are hard to evaluate by manually defined conditions, they are classified by a data-driven model.

5.2.2 Symbolic Skill Recognition

The symbolic pipeline evaluates each skill's pre- and post-conditions to recognize a skill during the demonstration. The symbolic pipeline using exemplary data is shown in Fig. 5.3, step (1a). The input of the algorithm is a stream of measurements $\boldsymbol{X}$, marked as <`in`> in the figure. The output lists segmentation points and skill labels, marked as <`out`>.

The pre- and post-conditions for each skill specified in Fig. 5.2a are evaluated at each time step t. These conditions are computed from the movement of the robot's gripper fingers and the measurements from the FTS mounted between the gripper and the robot. As these symbols do not suffice to distinguish between different *Contact*

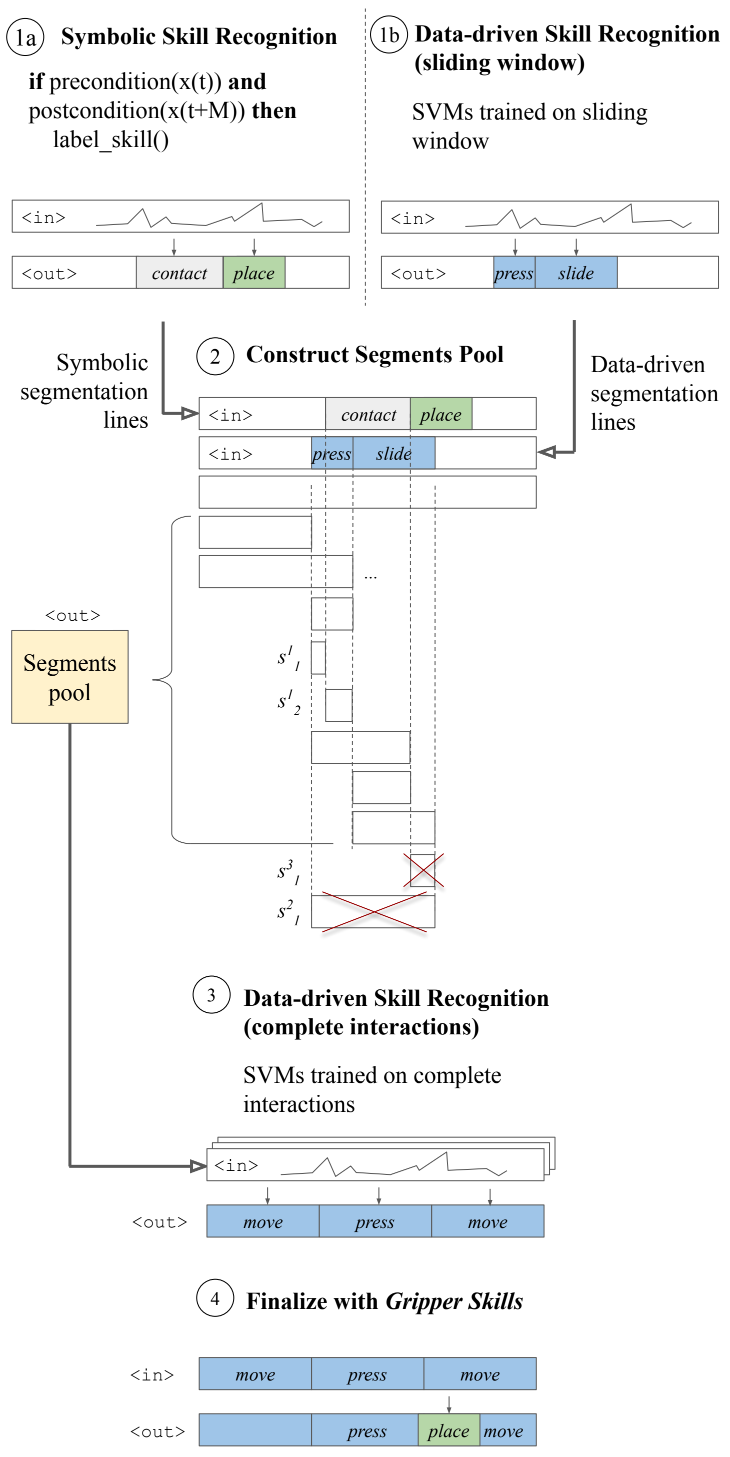

1a Symbolic Skill Recognition
if precondition(x(t)) and postcondition(x(t+M)) then label_skill()
1b Data-driven Skill Recognition (sliding window)
SVMs trained on sliding window
<in>
<out>
contact
place
press
slide
Symbolic segmentation lines
2 Construct Segments Pool
Data-driven segmentation lines
...
Segments pool
s^1_1
s^1_2
s^3_1
s^2_1
3 Data-driven Skill Recognition (complete interactions)
SVMs trained on complete interactions
move
press
move
4 Finalize with Gripper Skills
move
press
place
move

Figure 5.3: Skill Recognition Pipeline. The color code is the same as in Fig. 5.2, where the logic skill definitions are in green, and the classified skill definitions are in blue

Skills, the symbolic approach is limited to detecting an abstract *contact* skill instead. In summary, the symbolic segmentation can detect the skills specified in Fig. 5.2a.

Algorithm 1 describes the process of evaluating the pre- and post-conditions in the symbolic recognition pipeline. All skill candidates start with their status set to `IDLE` until the symbolic conditions indicate that the preconditions are met for any skill. Then, the status of the skill is set to `ACTIVE` as long as the post-conditions are not yet met. Once its post-conditions are met, its status is set to `DONE` and added to the list of detected skills with the information about where its data segments start and end. If the `ACTIVE` state temporally overlaps for multiple skills, the one that first fulfills its post-conditions dominates the selection and causes a reset of the `ACTIVE` state of all other skills. An exemplary output of the algorithm is shown in Fig. 5.3, step (1a), labeled as <`out`>.

Algorithm 1 Symbolic Recognition

```
Require: Measurements: $\boldsymbol{X}$, Set of Logic Skills (Fig. 5.2): LS
  while $\boldsymbol{X}^{(t)}$ do
    $condvals \leftarrow$ evaluate_conditions($\boldsymbol{X}^{(t)}$)      ▷ Compute value of each predicate
    for $s \in LS$ do
      update_status($s, condvals$)      ▷ set IDLE, ACTIVE, or DONE
      if $s.status ==$ DONE then
        $segments([s.start : s.end]) \leftarrow s.name$      ▷ label each sample with skill name
        for $s \in LS$ do
          $s.status \leftarrow$ IDLE
        end for
        break
      end if
    end for
  end while
Ensure: $segments$
```

5.2.3 Data-Driven Skill Recognition on Time Window

Figure 5.3, step (1b) shows the data-driven recognition pipeline. The input (labeled as <`in`>) is the stream of measurements $\boldsymbol{X}$ and the output (labeled as <`out`>) is a list of skill labels. The recognition module employs a support vector machine (SVM) with a sliding window approach similar to [82]. At each time step t, a feature vector $\boldsymbol{F}^{(t)}$ is computed from the measurements $\left[\boldsymbol{X}^{(t-W+1)}; \ldots; \boldsymbol{X}^{(t)}\right]$ in the current time window of length W. The SVM predicts on $\boldsymbol{F}^{(t)}$ which skill is most likely performed in the given time window.

The features in $\boldsymbol{F}^{(t)}$ consist of two parts. The first part shows features extracted with the library *tsfresh* [190], using the proposed minimal feature set that consists of standard deviation, sum of values, maximum, median, minimum, variance, and mean. The feature extraction function f_{tsfresh} maps a univariate time series of length W to seven feature values, defined as

$$f_{\text{tsfresh}} : \mathbb{R}^W \mapsto \mathbb{R}^7.$$

Table 5.1: Additional Contact Skill Features

Feature	Definition
mean translation power	$p_f = \frac{1}{L}\sum_{t=1}^{N} \boldsymbol{f}^{(t)T} \cdot \dot{\boldsymbol{p}}^{(t)}$
mean rotation power	$p_\varrho = \frac{1}{L}\sum_{t=1}^{N} \boldsymbol{\varrho}_t^T \cdot \dot{\boldsymbol{o}}^{(t)}$
slope to max force	$\Delta f = \frac{\max\left(\hat{f}^{(1)},\ldots,\hat{f}^{(N)}\right)-\hat{f}^{(1)}}{\underset{n}{\operatorname{argmax}}\left\{\hat{f}^{(n)}\right\}}$ with $\hat{f}^{(t)} = \lvert\lvert\boldsymbol{f}^{(t)}\rvert\rvert_2$
slope from max force	$\nabla f = \frac{\hat{f}^{(N)}-\max\left(\hat{f}^{(1)},\ldots,\hat{f}^{(N)}\right)}{N-\underset{n}{\operatorname{argmax}}\left\{\hat{f}^{(n)}\right\}}$
contact duration	N
contact distance	$d = \lVert\boldsymbol{p}^{(N)} - \boldsymbol{p}^{(1)}\rVert$
force-torque correlation	Coefficient of determination R^2, obtained from a linear function approximation $f : \boldsymbol{\varrho} \longrightarrow \boldsymbol{f}$.

Consider the time series

$$\boldsymbol{X}_{\text{tsfresh}} = \left[\boldsymbol{X}_{\text{tsfresh}}^{(t-W+1)}; \ldots; \boldsymbol{X}_{\text{tsfresh}}^{(t)}\right] \in \mathbb{R}^{W\times 4}$$

with a number of W samples obtained from a sliding window. A single sample is defined as

$$\boldsymbol{X}_{\text{tsfresh}}^{(t)} = [\lvert\lvert\dot{\boldsymbol{p}}\rvert\rvert, \lvert\lvert\dot{\boldsymbol{o}}\rvert\rvert, \lvert\lvert\boldsymbol{f}\rvert\rvert, \lvert\lvert\boldsymbol{\varrho}\rvert\rvert] \in \mathbb{R}^4 ,$$

where the velocity $\dot{\boldsymbol{p}}$ is computed from the Cartesian position $\boldsymbol{p}$ and the angular velocity $\dot{\boldsymbol{o}}$ from the orientation $\boldsymbol{o}$ of the end-effector frame. The variables f and ϱ refer to the force and torque at the end-effector. The Euclidean norm is computed over these variables, denoted as $\lVert\ldots\rVert$. The feature vector $\boldsymbol{F}_{\text{tsfresh}}^{(t)} \in \mathbb{R}^{28}$ is obtained by applying f_{tsfresh} column-wise on $\boldsymbol{X}_{\text{tsfresh}}$ and by stacking the results row-wise.

The second part consists of manually designed features mainly taken from Sec. 4.2, denoted as $\boldsymbol{F}_{\text{add}}^{(t)}$ (Tab. 5.1). They address physical relations between motion and force as expected in the *Contact Skills*. A combined feature vector is constructed as $\boldsymbol{F}^{(t)} = \left[\boldsymbol{F}_{\text{tsfresh}}^{(t)}, \boldsymbol{F}_{\text{add}}^{(t)}\right]$. Finally, $\boldsymbol{F}^{(t)}$ is normalized dimension-wise to mean $\mu = 0$ and standard deviation $\sigma = 1$ for improved numerical stability in the classification.

As the strength of the data-driven segmentation lies in detecting skills by their force and torque profile, the SVMs are trained to predict the *Classified Skills* specified in Fig. 5.2b. The *Gripper Skills* specified in Fig. 5.2a (*pick*, *place*, *open-gripper*, *close-gripper*) are not classified in a data-driven manner, as the symbolic segmentation can robustly and efficiently identify them without any uncertainty. The SVMs use a Radial Basis Function (RBF) kernel, whose regularization parameter and the kernel bandwidth are found through a grid search. They are trained on data of the skills, both consisting of the whole skill as well as sub-segments of the skills that lie within the sliding window.

Algorithm 2 shows the computational steps required for the data-driven recognition pipeline. While the window is sliding over the incoming measurement stream, the SVM keeps on predicting each sample, and the predicted label is appended to a list. The prediction rate easily achieves 50 Hz since SVMs are known to be computationally very efficient.

Algorithm 2 Data-driven Segmentation & Recognition

```
Require: Measurements: $\boldsymbol{X}$, Set of Classified Skills (Fig. 5.2): CS
  $y_{\text{last}} \leftarrow$None
  $t_{\text{last}} \leftarrow 0$
  while $\boldsymbol{X}^{(t)}$ do
      $\boldsymbol{F}^{(t)} \leftarrow$ compute_features($[\boldsymbol{X}^{(t-W+1)}; \ldots; \boldsymbol{X}^{(t)}]$)    ▷ Compute features over batch of
                                                                         measurements with length $W$
      $y$ = SVM.predict($\boldsymbol{F}^{(t)}$)                                    ▷ Predict skill label
      if $y_{\text{last}}$ != $y$ then
          $segments[t_{\text{last}} : t] \leftarrow y_{\text{last}}$
          $y_{\text{last}} \leftarrow y$
          $t_{\text{last}} \leftarrow t$
      end if
  end while
Ensure: segments
```

5.2.4 Combined Recognition

This section describes the stages $\textcircled{2}$, $\textcircled{3}$, and $\textcircled{4}$ of the recognition pipeline in Fig. 5.3. Now, both recognition pipelines from $\textcircled{1a}$ and $\textcircled{1b}$ are merged into a common pipeline.

Construction of Segments Pool Stage $\textcircled{2}$ combines the output of both recognition pipelines. The inputs are the segmentation points and skill labels of both symbolic $\textcircled{1a}$ and data-driven $\textcircled{1b}$ recognition pipelines. The task of this stage is to fill a *segments pool*, and its output is a list of segment candidates constructed by the following three rules:

1. A segment that is intersected by a segmentation point of the other recognition source leads to the addition of both split parts as segment candidates. An example is the addition of the segment candidates in Fig. 5.3 labeled as s_1^1 and s_2^1.
2. If two segments from the same recognition source, which is either symbolic or data-driven, follow each other, e.g., *press* and *slide*, it leads to the addition of the concatenated segment. An example is the segment candidate in Fig. 5.3 labeled as s_1^2. This enables the recognition of skills that consume more time and are composed of multiple steps, such as *peg-in-hole*.
3. If a segment is intersected by the segmentation point of a *Gripper Skill*, it does not lead to the addition of new segment candidates and the overlapping segment

candidate is removed from the pool. An example for excluded segments is given in Fig. 5.3), step $\textcircled{2}$, with the crossed out segment candidates, labeled as s_1^2 and s_1^3.

Rule 3. emerged since the recognition of *Gripper Skills* is considered to be highly reliable, and therefore avoids to create new segment candidates at regions that overlap with any of the *Gripper Skills*.

Data-driven Skill Recognition of Complete Interactions When the segment candidates are available in the *segments pool*, they can be classified by the data-driven stage in step $\textcircled{3}$. It uses the same implementation as in stage $\textcircled{1b}$. However, it is operated with the following differences:

1. The SVM is trained solely on complete examples of *Classified Skills* instead of examples observed in a fixed length time window.
2. The SVM predicts skills based on features computed from the segments in the segments pool instead of features computed on a fixed length time window.

The segment candidates that serve as input to this stage vary in duration. However, computing the features over one segment s always leads to a single feature vector $\boldsymbol{F}_s$ of fixed length. Each feature is designed to produce a scalar when computed on a time series of arbitrary length.

A skill is recognized by scoring all segment candidates for each possible skill class using the results of the SVM's decision function, given as

$$D_{\mathrm{SVM}} : \mathbb{R}^{N_{\mathrm{F}}} \mapsto \mathbb{R}^6 \quad \text{with } D_{\mathrm{SVM}}(\boldsymbol{F}_s) = \boldsymbol{z}_s \,. \tag{5.1}$$

Above function maps the feature vector $\boldsymbol{F}_s$ with N_{F} elements to a vector $\boldsymbol{z}_s$ that holds the prediction scores for each of the six possible *Classified Skills*. The skill with the highest score is extracted by

$$\overset{*}{s} = \underset{s}{\operatorname{argmax}}\{\boldsymbol{z}_s\} \,. \tag{5.2}$$

The resulting skill is inserted into the task representation. All other segments that overlap this segment are removed from the *segments pool*. This step is repeated until the whole demonstration is segmented into *move* skills and *Contact Skills*.

Finalization with Gripper Skills Step $\textcircled{4}$ adds the *Gripper Skills* to the task representation. These skills are directly taken from the symbolic recognition pipeline results of stage $\textcircled{1a}$ and overwrite the existing labels in the task representation. Exemplary, the input of this step is labeled as `<in>`, coming from the data-driven results of step $\textcircled{3}$ and the output is labeled as `<out>`. A final post-processing step reduces over-segmentation by splitting all segments below a predefined minimum length in the middle and merging the resulting parts into the segments before and after.

5.2.5 Online Segmentation with Immediate Feedback

The previously described method of combining the symbolic and data-driven recognition pipelines requires the demonstration to be completed, i.e., fully observed by the system. This only allows the approach to run offline. To overcome this limitation, an online segmentation algorithm is presented (Alg. 3). The main idea is to run the whole recognition pipeline whenever the system is confident about a past sub-sequence of skills. This confidence is given once a *GripperSkill* is detected by the symbolic recognition pipeline, which is a reliable segmentation point that will not change anymore throughout the following stages in the skill recognition.

Algorithm 3 Online Segmentation

```
Require: Measurements: $\boldsymbol{X}$, Skill Classes: $skills$
  while $\boldsymbol{X}^{(t)}$ do
    $s \leftarrow$ get_symbolic_segmentation_results($\boldsymbol{X}^{(t)}$)
    if $s$.state == DONE and $s$.type == $contact\ skill$ then
      show_data_driven_results()                                ▷ Fig. 5.4a
    else
      show_preliminary_results()                                ▷ Fig. 5.4b
    end if
    if $s$.type == $gripper\ skill$ then
      do_combined_recognition($s$)                              ▷ Sec5.2.4
    end if
    $skill\_sequence$.append(s)
    show_combined_results()                                     ▷ Fig. 5.4c
  end while
Ensure: $skill\_sequence$
```

The symbolic recognition detects online which skill is currently performed. When it detects a *contact* skill, the SVMs of the data-driven approach predict the most likely *Contact Skill*, as shown in the example in Fig. 5.4a. Once the symbolic skill segmentation detects the end of a *Gripper Skill*, such as *pick* or *place*, an intermediate fixed segmentation point is set since it will not change during the rest of the recognition process. Therefore, the demonstration part that ends with this point can already be treated as a complete demonstration part and thus be processed with the combined recognition algorithm. The results of the combined recognition then replace the preliminary recognition in the task representation (Fig. 5.4b). This technical feature enables the task representation to continuously develop, where past skills of the demonstrations are already finalized while the most recent skills are displayed as preliminary results. This process continues until the demonstration is complete and fully processed with the combined recognition algorithm as illustrated in Fig. 5.4c.

Due to the nature of the online skill recognition, the user receives immediate feedback about what the robot has already learned during the demonstration. The confidence of the robot can be visually indicated by a color for each recognized skill as exemplified in Fig. 5.4, showing how the confidence evolves from Fig. 5.4a to Fig. 5.4c.

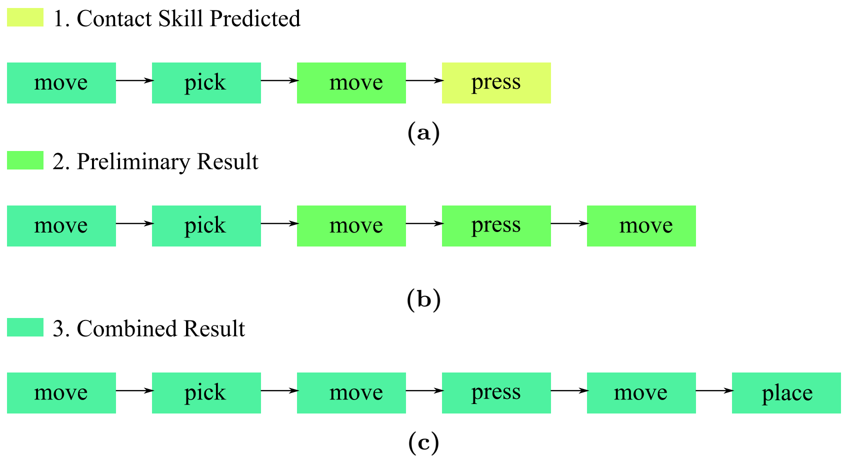


Figure 5.4: Consolidation of task representation during online kinesthetic teaching. The colors allow the user to monitor to which extent each skill is consolidated, i.e., which step of the online recognition has already passed. Yellow: SVMs currently predict the label. Green: The preliminary recognition is done. Turquoise: The combined recognition is finished.

Table 5.2: Parameter Values used in the Experiments

Parameter Name	Value
sampling rate	50 Hz
SVM feature window length W	0.4 s
SVM regularization parameter C	100
SVM RBF kernel coefficient `gamma` $= \frac{1}{2\sigma^2}$	0.01
minimum skill length before merging	0.4 s

5.3 Experiments

The framework is first assessed offline using a dataset of demonstrations (Sec. 5.3.1). Next, two benchmarks of its segmentation and skill recognition performance follow. On the one hand, it is compared to a purely symbolic approach. On the other hand, it is compared to a purely data-driven approach. This comparison uses an exemplary task demonstration (Sec. 5.3.2). Table 5.2 shows the parameter values used throughout the experiments.

5.3.1 Skill Recognition Performance

The recognition algorithm was trained and evaluated on a dataset based on the demonstrations of three subjects. The dataset contains approximately 100 demonstrations per skill recorded in batches, meaning that each user demonstrated ten times the *press* skill, then ten times the *slide* skill, and so forth. The skills were separated by demonstrating a free motion without contact.

The recognition algorithm was tested in a five-fold cross-validation, with the data split into 80% training data and 20% test data. The ground truth was obtained by annotating the skills manually. The approach is evaluated based on two commonly used metrics [191], which are temporal tolerance and classification by data point label. The temporal tolerance assesses how close the algorithmic segmentation points lie to the ground truth, without considering the labels of each segment. All algorithmic segmentation points that lie in a region of $\pm t_{\mathrm{err}}$ around the ground truth are considered as True Positive (TP) and False Positive (FP) otherwise. If a ground truth segmentation point has no corresponding algorithmic segmentation point within its region, it is considered a False Negative (FN). The region margin $\pm t_{\mathrm{err}}$ was chosen as 0.2s in this evaluation. The overall metric is computed by the $F1$-score

$$F1 = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}\,. \tag{5.3}$$

The framework segmented the dataset with an F1-score of 0.93, which combines the measures of precision and recall in a balanced manner. The placement of the segmentation points matters, as even if a skill has been mislabeled, the user could correct the label of the skill without having to further adapt the segmentation result.

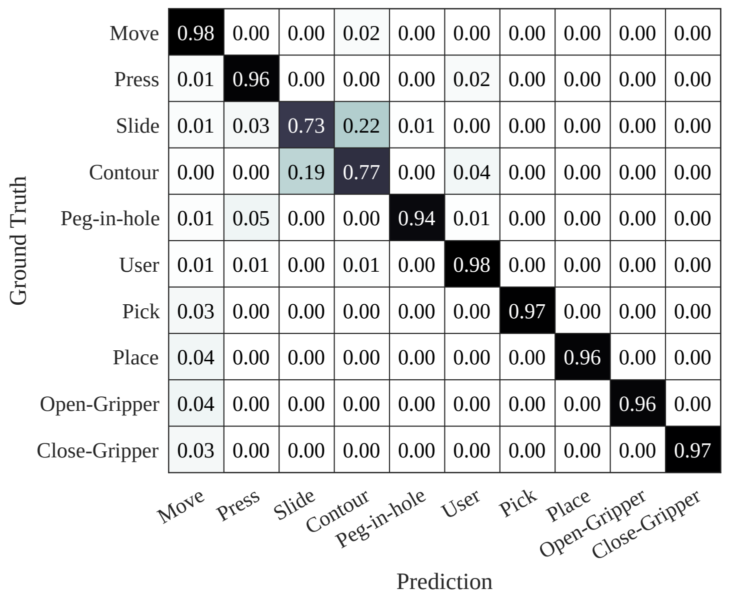
The second metric evaluates the classification accuracy by data point label, given as

$$C = N_{\mathrm{correct\_labels}} / N_{\mathrm{all\_labels}}, \tag{5.4}$$

which represents the ratio of data point labels l that have been chosen correctly by the algorithm. Consequently, a confusion matrix can be calculated showing what percentage of ground truth labels l_{g} have been labeled with label l_{a} by the algorithm (Fig. 5.5). The average recognition accuracy on the dataset was 92.2 %, which means that this ratio of data points was correctly labeled.

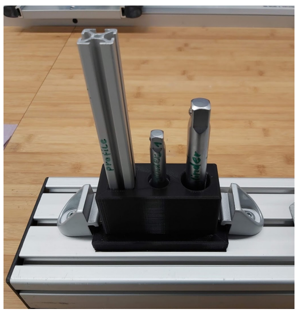
5.3.2 Comparison of Approaches

This section compares the results of the symbolic approach from stage (1a), the data-driven approach from stage (1b), and the merged approach from stage (4). The experimenter demonstrated multiple skills in a versatile environment to collect experimental data. Fig. 5.6a shows a part of this environment, which was also used for demonstrating the *peg-in-hole* skill shown in Fig. 5.6b. The segmentation results are shown in Fig. 5.7. The three rows in the plot represent each recognition approach. First, in the symbolic recognition results, the *Gripper Skills* were correctly recognized, but the areas of contact led to an abstract *contact* skill. This approach is not able to predict the specific skill type of the *Contact Skills*, and it does not resolve multiple sequenced skills within one *contact* block, for instance, in the contact area starting at time = 25 s. Next, the data-driven approach tends to over-segment the demonstration. It produces numerous artifacts in the form of very short skills. For example, starting at time = 10 s, the skills

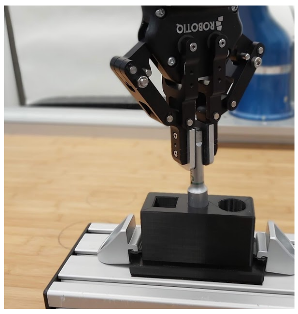


Ground Truth \ Prediction	Move	Press	Slide	Contour	Peg-in-hole	User	Pick	Place	Open-Gripper	Close-Gripper
Move	0.98	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00
Press	0.01	0.96	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00
Slide	0.01	0.03	0.73	0.22	0.01	0.00	0.00	0.00	0.00	0.00
Contour	0.00	0.00	0.19	0.77	0.00	0.04	0.00	0.00	0.00	0.00
Peg-in-hole	0.01	0.05	0.00	0.00	0.94	0.01	0.00	0.00	0.00	0.00
User	0.01	0.01	0.00	0.01	0.00	0.98	0.00	0.00	0.00	0.00
Pick	0.03	0.00	0.00	0.00	0.00	0.00	0.97	0.00	0.00	0.00
Place	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.00	0.00
Open-Gripper	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.00
Close-Gripper	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.97

Figure 5.5: Confusion matrix for the classification by data point label.



(a) Three different peg-in-hole problems with square, small round, and large round peg. The left and right ones are contained in the SVM training set.



(b) Exemplary peg-in-hole task with unseen objects.

Figure 5.6: Experimental environment for data collection.

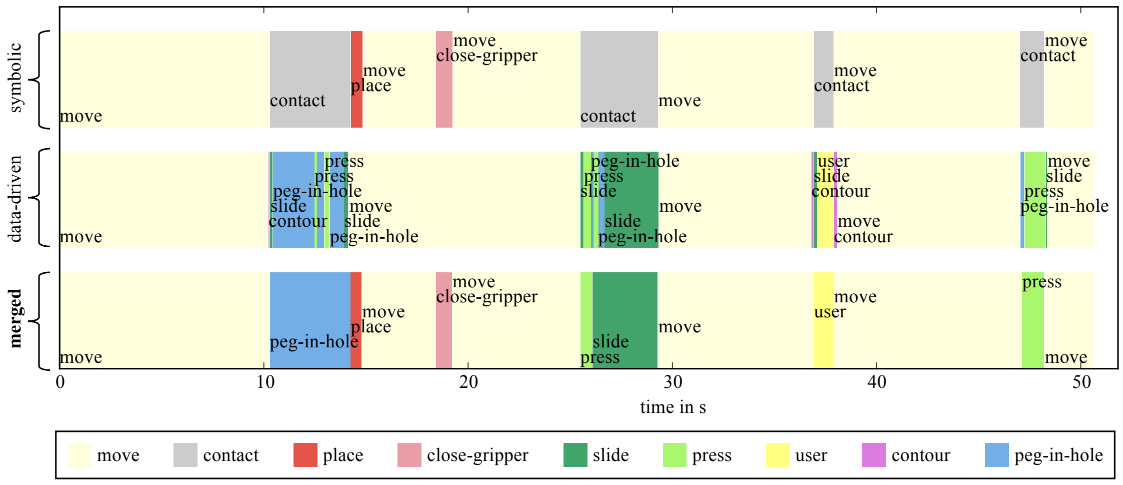


Figure 5.7: Comparison of approaches.

of *contour*, *slide*, and *peg-in-hole* were recognized sequentially, although the demonstration contained only a single *peg-in-hole* skill in this interaction. This issue could be caused by the fact that a single *peg-in-hole* interaction could consist of a sequence of other skills. The greedy search principle might then favor such skills, given that only a small time-frame can be observed in the sliding window. Finally, the results of the merged approach are shown in the bottom row. Due to the usage of the segments pool, which possibly contains longer data segments compared to the one in the sliding window, the data-driven algorithm was also able to classify adjacent skills in the same contact and found the most likely candidate. This feature allowed to handle situations where the observed skills significantly differed in their duration. Consequently, the proposed algorithm handles the over-segmentation problem of the data-driven approach well while recognizing the full palette of skills with instances of varying duration.

5.3.3 Discussion

The skill recognition performance of the framework was assessed based on a dataset (Sec. 5.3.1) and by an exemplary demonstration, comparing a symbolic, a data-driven, and the presented approach with each other. The framework uses only proprioceptive sensor values and does not require object recognition or tracking. Object independence enables the framework to be deployed in new setups without predefining the involved objects. However, the system would not recognize changes in the scene due to the missing visual input.

One limitation is the robustness towards large demonstration speed in combination with the minimum allowed skill duration. If a user would demonstrate in a too fast pace, the overall duration of a data segment could lie below the minimum skill length (Tab. 5.2). Then, recognizing the skill would become impossible. This could be tackled by decreasing the minimum skill length, potentially leading to very small segments. Consequently, the user could use a GUI that supports decisions about which of these skills shall be kept or merged together. The data-driven approach uses the combined features from $\boldsymbol{F}_{\text{add}}$ and $\boldsymbol{F}_{\text{tsfresh}}$. An ablation study could improve the feature set to extract only the highly relevant ones, increasing the classification accuracy.

The ability to perform online segmentation distinguishes the presented approach from purely data-driven approaches such as [59, 84–86]. By combining symbolic and data-driven segmentation, the framework can efficiently evaluate the segmentation result, as described in section 5.2, at each time step during user demonstration. The methods in [84] and [86] use a sample-based inference approach that is not feasible to be solved online. In [85], the segmentation result of different demonstrations is passed to a classification task, making it unsuitable for online segmentation. Moreover, the proposed method can segment a single task demonstration, while the approaches in [59,63,84,85] require multiple demonstrations of the same task. The BP-AR-HMM based segmentation approaches [63,84] use the variation in skill endpoints from different demonstrations to identify the relevant coordinate frame of each skill. Similarly, [59] analyzes the skill endpoints from different demonstrations to obtain transition states for segmentation. The presented approach uses a predefined and pre-trained skill library to avoid the need for multiple user demonstrations. This may limit flexibility in detecting unknown skills, but it is expected that reusable, domain-specific skills are known in advance in a production environment and suffice for most situations.

5.4 User Study

The user study has two aims: 1) Assess the framework's segmentation accuracy and correctness of the recognized skill sequence in a laboratory study with six subjects; 2) Evaluate if users can teach a robotic task and debug incorrect robot programs with the help of the task representation that the framework automatically constructs from the recognized skills.

All participants have some technical background in different fields of robotics. Still, they have not used the presented framework or participated in the data collection for the training set.

5.4.1 Procedure of the Laboratory Study

The laboratory study consisted of a teaching phase (points 1..5) and a debugging phase (points 6..9) as specified in the following procedure:

1. Experimenter shows instruction video for method A
2. Experimenter shows instruction video for task 1
3. Subject demonstrates task 1 with method A
4. Experimenter shows instruction video for task 2
5. Subject demonstrates task 2 with method A
6. Experimenter explains that a task can be debugged given a task representation of method A

7. Experimenter asks subject for expected task outcome of debug task $D(x, y)$ and lets subject correct it if desired using method A
8. Experimenter asks subject again for expected task outcome and lets subject correct if desired
9. Subject observes the task execution according to the current task representation
10. Subject evaluates overall framework by questionnaire concerning method A
11. Subject evaluates comparison between methods A and B by questionnaire

Tasks 1 and 2 are placeholders for the stamping and assembly tasks correspondingly. Methods A and B are placeholders for two different task representations, which are the skill-based and time-line representations. A detailed description can be found in Sec. 5.4.4. The steps above from 1 to 10 were then reiterated but using Method B instead of Method A. Throughout the study, the sequential order of tasks $\in$ 1, 2, debug tasks $D(x, y) \in$ D(1,1), D(1,2), D(2,1), D(2,2), and methods $\in$ A, B were permuted through a Latin Square design [192]. Before commencing the experiment, the participants were offered a familiarization phase with the robot in gravity compensation control.

In the teaching phase (points 1..5), the participant engaged in programming two robot tasks by utilizing two distinct forms of visual feedback, thereby generating a total of four programming sequences. The participant was aware that the robot would replay each human demonstration to capture the robot's repetition of the task. For this, a Cartesian impedance controller tracks the demonstration trajectory. Gaussian Mixture Models were constructed for each skill jointly using the demonstration and repetition data acquired from the participant and robot. Subsequently, Gaussian Mixture Regression was applied based on these models to derive a trajectory for task execution. More details about this procedure are described in [193].

In the debug phase (points 6..9), the participant examines previously programmed task representations demonstrated by the experimenter, where the tasks were either demonstrated correctly or intentionally incorrectly. These debug tasks utilize the same environment as the stamping and assembly task, resulting in four different debug task conditions with associated task representations, namely: D(1,1): stamping task - correct; D(1,2): stamping task - incorrect; D(2,1): assembly task - correct; D(2,2): assembly task - incorrect. The incorrect tasks were designed to exhibit a wrong skill sequence with respect to the correct skill sequence of the stamping and assembly tasks. The goal of the subject was to validate if those task representations correctly perform the tasks. It was of interest if the task representation alone provided sufficient information. Therefore, the users were asked to give their opinion on the correctness before they were allowed to watch the robot executing the program. If they expected a task representation to be incorrect, they were asked to correct it by selecting the skill that appeared to be faulty in the task representation. Subsequently, the robot moved to the correct start location. Finally, the user could provide a new demonstration that was subsequently segmented, resulting in an updated task representation.

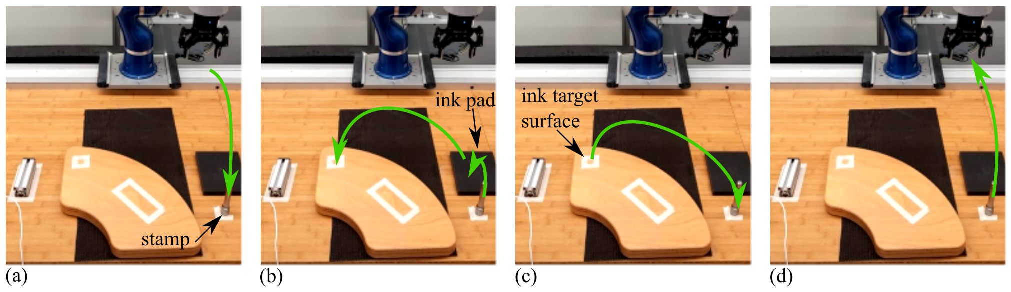


Figure 5.8: Stamping task instructions. (a) pick the stamp, (b) push it into the ink pad and then push it into the ink target surface, (c) place it at its original location, and (d) move the gripper up to the home position. The task demonstration is shown in the accompanying video for the stamping task.

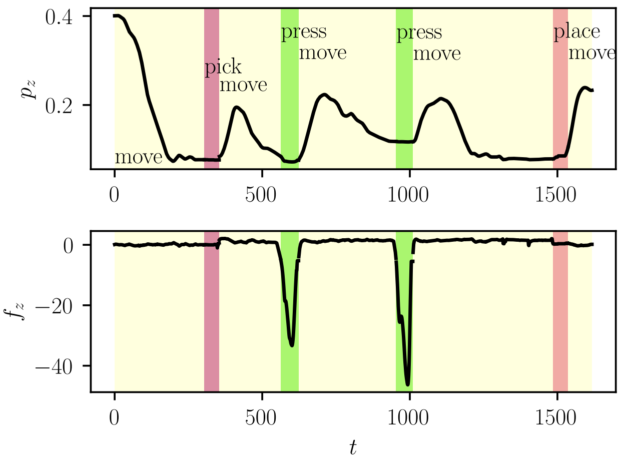
5.4.2 Skill Recognition for Online Task Representation

This section evaluates the task representation that the framework builds as a sequence of skills while the user is teaching the robot. Therefore, using the presented framework, two users were asked to program a DLR LWR IV [6] robot. The robot was equipped with a Robotiq 85 two-finger gripper connected to a wrist-mounted FT sensor that measures the wrench acting on the gripper. Fig. 5.8 shows this setup. The user transferred the task by kinesthetic teaching and used two buttons, one to start and stop the demonstration and another to open and close the gripper. The following paragraphs describe two experimental programming tasks used in the evaluation.

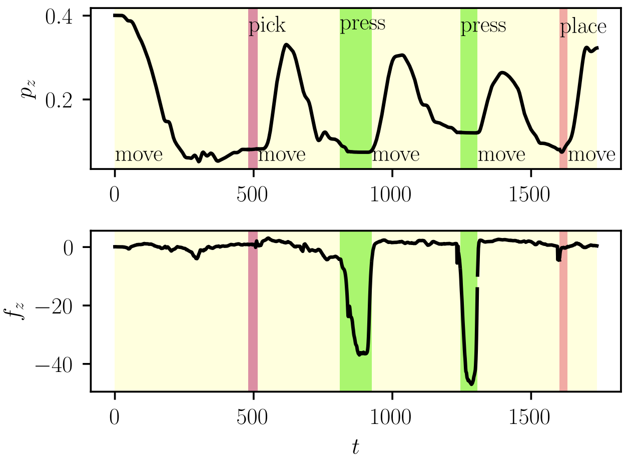
Stamping Task Fig. 5.8 shows the visual task description for the stamping task, and its caption specifies the textual user instructions to demonstrate a task that involves two force-based interactions with the environment. The first interaction requires pressing the stamp into the ink pad. The second interaction requires pressing the stamp onto the surface of the ink target. The system is expected to recognize a *press* skill in both interactions.

The skill recognition results are shown as colored segments overlaid on the gripper position and applied force in the z-axis (Fig. 5.9). The users achieved a segmentation accuracy of 96.9 % (f1-score=0.75) and 98.2 % (f1-score=1) respectively. The skill recognition results show that the robot moved to the stamp (*move*), picked it (*pick*), moved to the ink pad (*move*), pressed on it (*press*), moved to the ink target surface (*move*), pressed on it (*press*), moved to the stamp's place location (*place*), placed the stamp (*place*), and moved up to the robot's home pose (*move*). Both force-based interactions were recognized as *press* skills, referring to the regions of large force magnitude in the bottom plot of each user.

Assembly Task In this task, users were asked to demonstrate to the robot how to apply glue to a surface and then attach an object to it. Fig. 5.10 presents a visual task description, and the textual user instructions can be found in the caption.



(a) Subject 1



(b) Subject 5

Figure 5.9: Stamping task results for subject 1 and 5. p_z shows the position in z-axis and f_z the force in z-axis.

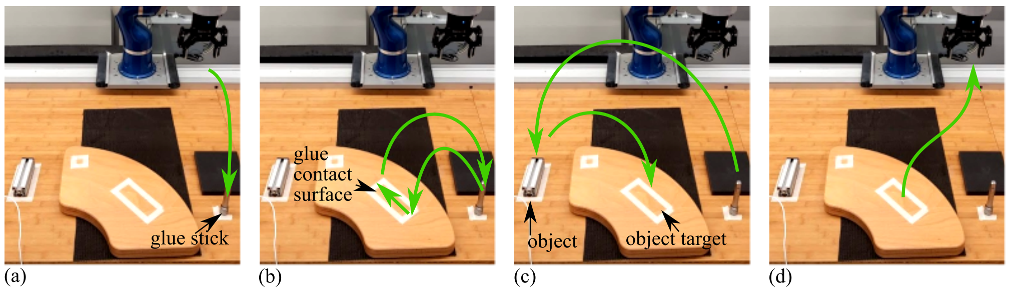


Figure 5.10: Assembly task instructions. (a) pick the glue stick, (b) slide it over the contact surface and place it back at its original location, (c) pick the object and press it onto the object target, (d) move the gripper up to the home position. The task demonstration is shown in the accompanying video for the assembly task.

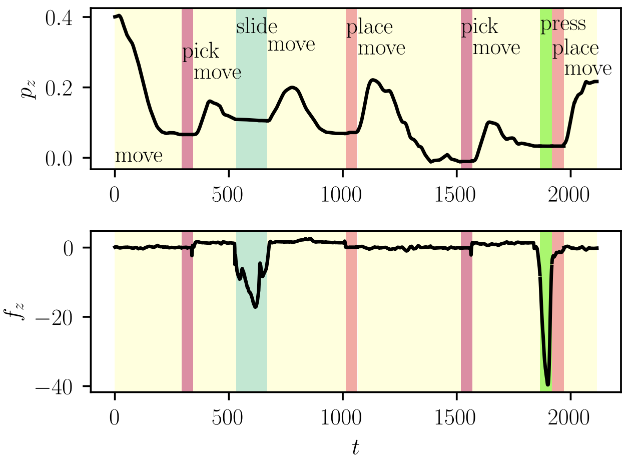
The skill recognition results for both users are overlaid as colored segments in Fig. 5.11, which also presents the end-effector position and force in the z-axis.

For subject 1, a validly programmed task is shown in which the glue stick is picked up (*pick*), then slides over the adhesive surface (*slide*), and is placed back in the original position (*place*). The object is then picked up (*pick*) and pressed onto the sticky surface (*press*). While still in contact with the environment, the gripper is opened, leading to the detection of a *place* skill. Finally, the robot is moved to its initial position (*move*). The user decided to press on the object while it was still gripped, corresponding to the task description of assembling it by pressing on the glue bonding. Considering the strategy of subject 1, the skill recognition seamlessly detected the transition from *press* to *place* without an intervening *move* (Fig. 5.11a). It is highlighted that the presented framework can detect transitions from forceful interactions to grasp actions, where the opening of the grasp itself causes the contact break with the environment.

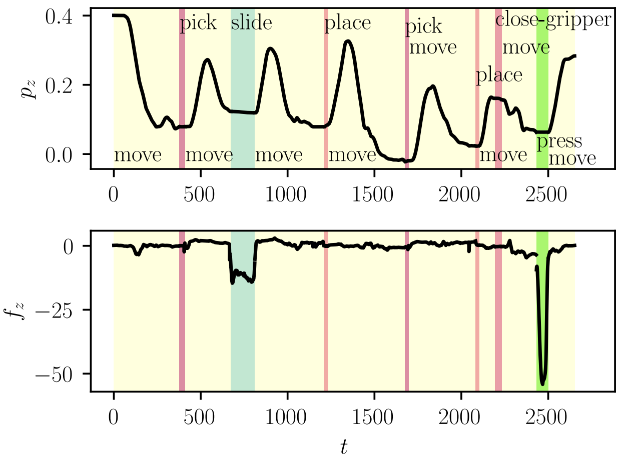
Subject 5 followed the strategy of subject 1 until the object was grasped. Instead of using the sequence of (*press*) and (*place*), subject 5 placed the object without significant force application (*place*), then moved the gripper over the object (*move*), closed the gripper without the object (*close-gripper*), moved downward towards the object (*move*), pressed on the object with the empty and closed gripper (*press*), and finally moved the end-effector to the initial position (*move*). Although the strategy was different compared to subject 1, both users achieved the task goal of attaching the object to the target object by pressing on the glue bonding.

5.4.3 Performance Metrics

All participants saw the same video instructions about how to program the stamping and assembly task, which is represented as a storyline in Figs. 5.8 and 5.10. Regarding the assembly task, no specific gluing strategy was required, allowing the user to perform the sliding action as desired. For instance, some users decided for a single and possibly curved sliding interaction, while others decided on multiple sliding interactions with the glue contact surface.



(a) Subject 1



(b) Subject 5

Figure 5.11: Assembly task results for subject 1 and 5. p_z shows the position in z-axis and f_z the force in z-axis.

Table 5.3: Task representation results

Subject ID	**Task Representation**	**Succ.**
stamping GT (as instructed)	*move*, *pick*, *move*, *press*, *move*, *press*, *move*, *place*, *move*	
1, 2, 3, 5	*move*, *pick*, *move*, *press*, *move*, *press*, *move*, *place*, *move*	✓
4	*move*, *pick*, *move*, *press*, *move*, *press*, *move*, *press*, *place*, *move*	✓*
6	*move*, *pick*, *move*, *press*, *move*, *place*, *move*	–
assembly GT (as instructed)	*move*, *pick*, *move*, *slide*, *move*, *place*, *move*, *pick*, *move*, *press*, *place*, *move*	
1	*move*, *pick*, *move*, *slide*, *move*, *place*, *move*, *pick*, *move*, *press*, *place*, *move*	✓
2	*move*, *pick*, *move*, *slide*, *move*, *press*, *move*, *place*, *move*, *pick*, *move*, *press*, *place*, *move*, *close_gripper*, *move*, *press*, *move*	✓**
3	*move*, *pick*, *move*, *slide*, *move*, *slide*, *move*, *place*, *move*, *pick*, *move*, *place*, *move*, *close_gripper*, *press*, *open_gripper*, *move*	✓**
4	*move*, *pick*, *move*, *slide*, *move*, *press*, *move*, *place*, *move*, *pick*, *move*, *place*, *move*, *press*, *move*	✓***
5	*move*, *pick*, *move*, *slide*, *move*, *place*, *move*, *pick*, *move*, *place*, *move*, *close_gripper*, *move*, *press*, *move*	✓**
6	*move*, *pick*, *move*, *slide*, *move*, *place*, *move*, *pick*, *move*, *place*, *move*	–

*, **, and *** refer to special cases that are discussed in Sec. 5.4.6.

Table 5.4: Overall performance metrics

	stamping	assembly	average
$F1$ score (see. Equation 5.3)	0.90	0.84	**0**.87
C (accuracy, see. Equation 5.4)	0.97	0.94	**0**.96

Table 5.3 shows the task representation results of the stamping and assembly task for each subject numbered from one to six. The ground truth (GT) is stated for a successful skill sequence, as it was instructed to the participants.

Even though the initial instruction video suggests a skill sequence that solves the task, other skill sequences originating from the user's personal strategy can still lead to a successful task execution. Successful executions are marked in column "Succ." with ✓and unsuccessful ones with –. Exceptional cases marked with (*) refer to the discussion section 5.4.6. Table 5.4 summarizes the quantitative performance of the framework when confronted with the untrained subjects of this study. The metrics were already introduced in Sec. 5.3.1. The laboratory study achieved an F1-score of 0.87 and an accuracy of 96 %.

5.4.4 Evaluation of Task Representation

The evaluation consists of two parts. The first part is a laboratory study, as described in Sec. 5.4.1, in which six participants program a robot using the framework. The second part is a remote study involving 26 users to evaluate the comprehensibility of the proposed approach. There were different participants recruited in each of these parts. The

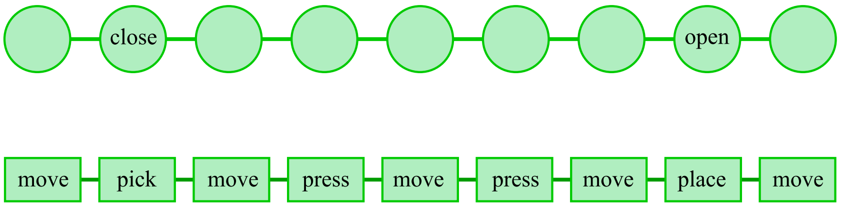


Figure 5.12: Top: stamping task in the time-line representation; bottom: stamping task in the skill-based representation.

type of the presented task representation is termed skill-based. A so-called time-line task representation was defined as a baseline that does not provide a skill annotation for each demonstration segment. This representation was introduced in [168], and its segmentation approach originally uses the Douglas-Peucker line simplification algorithm to detect notable points of a trajectory, which are then used to encode the trajectory. As these points have no semantic meaning, the resulting task representation describes the learned task through a time-line, showing the detected points and gripper movements. Since the focus was on evaluating the task representation, the same segmentation points found with the proposed approach were used to construct the time-line task representation. The time-line task representation shows the segmentation result without skill annotation. Additionally, the opening and closing events of the gripper are displayed. The same task, but in different task representations can be seen in Fig. 5.12. The *contact skills* used in the experiments were limited to *press* and *slide* to decouple the classification results from the evaluation of the task representations. The time-line representation, like the skill-based representation, is updated online when a new task segment is found. It also displays the currently active segment while the task is being replayed. It further allows the robot to be moved to a selected point in the task representation or to delete points through the user interface, e.g., for debugging purposes.

After performing all steps for one of the task representations in the laboratory study, the users were asked to fill out questionnaires. The NASA-TLX was used to measure the workload, and the ISO 9241-110 questionnaire for Interaction Principles was used to evaluate the quality of interaction between the robot's user interface and the human. To further investigate the explainability of the task representations, based on [189], questions with regards to three more categories were used: trust in the robot, ease of debugging, and match of the user's mental model to the real robot.

The remote study was planned to expand the laboratory study, focusing on the debug phase and employing the same robotic task scenario. Instead of programming a physical robot, users were requested to associate a task description, presented by a video, with a corresponding task representation. In contrast to the previous approach of correcting the robot program, the participants were instructed to label the task outcome as correct or incorrect. The subjects were split into two groups of equal size, each only seeing one of the task representations evaluated through the ISO 9241-110 questionnaire and an interview.

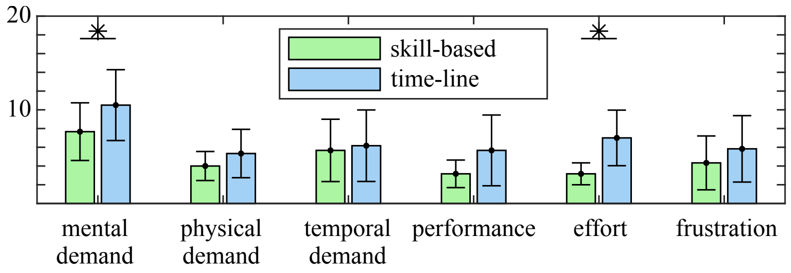


Figure 5.13: The results of the NASA-TLX questionnaire obtained from the laboratory study participants. The users scored the categories on a scale from 1 (best) to 20 (worst). The plots show the mean scores for the individual categories for the skill-based task representation used in the presented approach (blue) and the time-line task representation (red). The $*$ marks that a p-value of $p < 0.05$ was found with a Wilcoxon signed rank test.

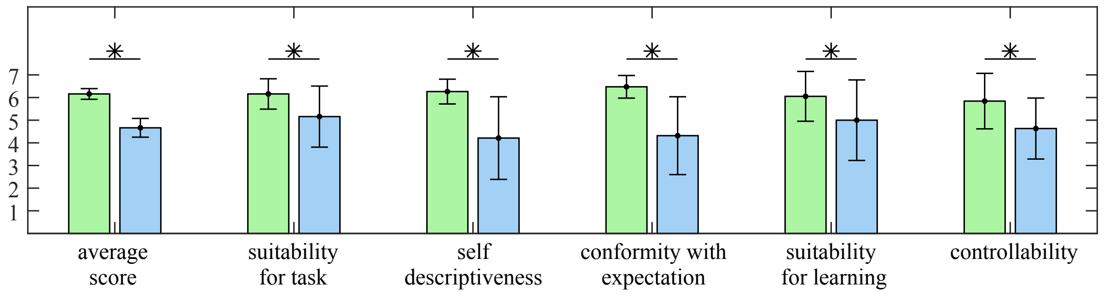


Figure 5.14: The combined results of both the online and the laboratory study for the ISO questionnaire. The scores rate how much the users agree with the category from 1 (strongly disagree) to 7 (strongly agree). For details about the $*$ that marks significance and colors please refer to Fig. 5.13.

5.4.5 Subjective Results

A number of subjective results were collected from the user study with the help of questionnaires. Fig. 5.13 shows the perceived workload of the approaches with different task representations. A clear reduction of the mental workload and effort is visible for the use of the skill-based representation in comparison to the time-line representation.

Figure 5.14 opposes the intuitiveness and usability of the different representations, as assessed by the ISO 9241-110 questionnaire. The users rated the usability of the skill-based task representation 32% higher than that of the time-line representation. Notably, in terms of intuitiveness, the skill-based task representation significantly outperformed the time-line task representation in the categories of self-descriptiveness (48% higher) and conformity with the user's expectation (50% higher). Fig. 5.15 shows how the presentation of the robot's knowledge influenced the teaching procedure. They are grouped in the abovementioned categories as proposed in [189]. In the debugging category, the users were asked to rate the ability to detect errors in the robot's learned program with the help of the task representation. The results suggest a clear preference for the skill-

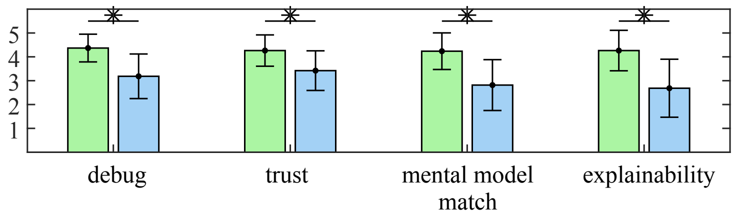


Figure 5.15: The combined results of both the online and the laboratory study for the explainability questions. For details about the ∗ that marks significance and colors please refer to Fig. 5.13. The questions were grouped in the categories 'Ease of debugging,' 'Trust in the robot,' 'Match of the user's mental model to the robot's real knowledge,' and 'Explainability of the task representation,' shown from left to right. The users rated on a 5-point Likert scale from 1 (strongly disagree) to 5 (strongly agree). The higher the result, the better.

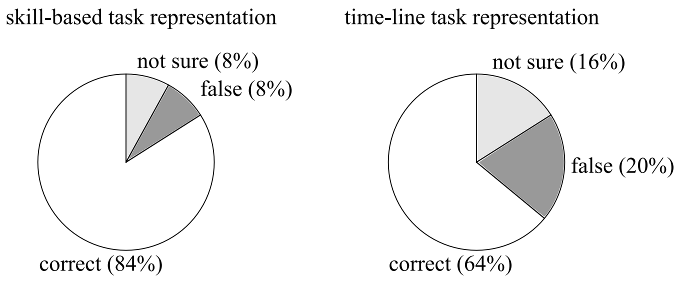


Figure 5.16: The results of the debug tasks. The participants were asked whether a task representation would execute the task successfully. The diagrams show the percentage of answers where the user's predicted outcome matched the task outcome. The answers were evaluated to be correct, false, or undecided (not sure) for the skill-based task representation (left) and the time-line task representation (right).

based task representation. The users' self-evaluation is also confirmed by the number of times the users could predict if a task representation would perform a task as desired, shown in the results in Fig. 5.16. It shows that, with the help of the skill-based task representation, 84% of the participants recognized that the visualized task representation would not match the intended task goal. In contrast, only 64% could do so with the time-line task representation. In the categories *match of mental model* and *explainability*, the skill-based task representation also outperformed the time-line task representation drastically (see Fig. 5.15). The influence of the task representation was also noticeable in the laboratory study. Most participants caused the detection of a *press* skill before the placing of objects, as they would forcefully set them on the workspace. In the time-line representation, none of the users chose to correct this behavior, which could be seen in the robot's execution of the task. In contrast, in the skill-based task representation, 50% of the users removed this *press* skill, as the graph did not match their expectation of how the task should be executed.

Furthermore, the users exhibited a 24% increase in trust towards the robot, although the execution of the taught programs was unrelated to the task representation. In the final interview of the laboratory study, the participants favored the skill-based task representation. When asked about their initial impressions, half of the participants found the time-line representation to be more comprehensible when seeing it for the first time, as it conveyed less information than the skill-based representation. However, when asked which one they found easier to use in practice, their preference shifted towards the skill-based representation.

5.4.6 Discussion

The user study showed that the generated robot programs among the subjects differed for the same task. However, most subjects reached the task goals with their strategy in each task. Variations between task description and the user's teaching behavior are marked with the symbol * in Table 5.3. Regarding the stamping task, the symbol * denotes that one user pressed the tool twice on the ink target surface, which was correctly recognized by the system, although the task description did not require it. Regarding the assembly task, the symbol ** marks users who decided on another strategy by closing the gripper before applying pressure on the object. In this case, the system also correctly recognized the skills, and the task was performed successfully. User 4 (***) demonstrated a skill sequence that involved two additional press skills caused by contacting the environment with the tool. This issue might be caused due to low expertise in kinesthetic teaching but did not lead to unsuccessful task execution.

Considering the human and robot perspective, numerous benefits emerge from the obtained task representation. First, it contains the knowledge about the recognized skills, which can be used by the robotic system to efficiently reproduce them. Here, the robot can rely on a library of existing skill implementations to execute the skills using an optimized strategy available from the state of the art. Second, the user could adapt skill-related parameters given a graphical interface without demonstrating the task again. An example is to adjust the pressing force in the *press* skill. Third, the user could adapt the task representation without providing a new demonstration, for instance, exchanging skill types that fit better to the task requirements. For example, the system identified a *slide* skill although the user wants to employ a skill of type "polish." Then, the associated data segment could be reused to parameterize the desired skill implementation. Fourth, the user can observe what the robot is currently performing, which helps the robot to explain itself and the user to gain trust in the system.

The results of the user study indicate that a skill-based task representation has a significantly better explainability and usability as a time-line task representation. With the skill-based representation, users were able to identify errors in the task representation at first glance. In contrast, the time-line representation forced users to go through every step of the task and estimate how many steps this might correspond to in the time-line.

In the user study phase of debugging a faulty program, the users inserted new demonstrations in the time-line task representation at points where the gripper opened or closed, as these were the easiest to detect. On the contrary, the skill-based task representation allowed the users easier access to correct a task since each represented skill had an understandable meaning. Furthermore, the users of the laboratory study often did not notice the necessity of correcting a program when it was presented in the time-line task representation. Especially in the case mentioned above of the *press* skills before the *place* skills, users only noticed the necessity to correct this mistake in the skill-based representation. The stronger engagement with the skill-based task representation suggests that the user has finer control over the knowledge of the robot.

5.5 Conclusion

The presented online task segmentation and skill recognition algorithm combines symbolic segmentation by evaluating pre-conditions and post-conditions and data-driven segmentation and recognition by a classifier that predicts appropriate skills. This combination enables detecting a more comprehensive set of skills, involving *Contact Skills*, while still allowing the algorithm to run online.

The experimental results confirm the recognition capabilities and showcase how task representations emerge from kinesthetic user demonstrations. These task representations are consolidated on the fly whenever the incoming data increases the confidence of the system. With that, the users receive immediate feedback about the currently processed skills and finally obtain a consolidated task representation about how the robot interpreted their demonstration. Immediate feedback for the end-user is expected to build trust by closing the gap between the user's mental model of the robot's knowledge and its actual knowledge.

Future work could focus on the integration of additional input modalities beside kinesthetic teaching during the skill recognition process. However, using only proprioceptive data is cheap, easy to achieve with a collaborative robot, and does not suffer from known problems of visual perception, such as occlusion or sensitivity to lighting conditions. Beside that, it could be explored what is a feasible task complexity that users would still like to address with a single demonstration. There must be a trade-off between ease of task definition by a one-shot demonstration and the mental load that comes with it. For a too complex task, users might be overloaded with demonstrating it in a single shot. Then, partial task demonstrations or individual skill demonstrations might be preferred, which is too be explored.

6 Contact-based Exploration

This chapter proposes a method for robotic exploration of an uncertain workspace by sensing object locations using force-sensing capabilities at the end-effector. The presented task definition method relies on multiple demonstrations as input to define an adaptive task execution behavior, which allows the robot to adapt its motions online based on the explored state of the environment.
The chapter includes the content of the following publication [42]:

- T. Eiband, M. Saveriano, and D. Lee, "Learning haptic exploration schemes for adaptive task execution," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7048–7054.

The author of this thesis developed the proposed methods and conducted the user study and experiments. D. Lee advised in developing research methodologies and analyzing the results and revised the article.

The chapter is outlined as follows. Section 6.2 describes the approach for segmentation and skill identification. Section 6.3 introduces the method for learning skill relations. Section 6.4 presents the experiments. Conclusions and future work are stated in Sec. 6.5.

6.1 Introduction

In contact-based exploration, two familiar terms are tactile and haptic sensing. From the human perspective, tactile sensing is considered to be a sub-type of haptic sensing, which can be of two different forms: tactile and kinesthetic. Tactile sensing is known to be the detection of forces on the skin. The human tissue is equipped with a myriad of sensors that allow the sensation of touch and pressure, surface texture type, and vibrations. Kinesthetic sensing is close to what is known as proprioception, but it assumes that there is a physical connection between body parts and the environment such that forces can be exchanged. It incorporates sensing a body part location with respect to external objects and sensation of forces due to changes in muscle strength. Considering the term feedback in place of sensing, haptic feedback combines tactile and kinesthetic feedback. This work addresses the exploration of the environment by a robotic tool, where the most appropriate term is identified to be haptic exploration.

Revisiting the human perspective, it is evident that interaction forces play an essential role when detecting and exploring the surroundings in daily life scenarios. Besides visual observations, humans use haptic feedback as an additional source of information when exploring or manipulating an uncertain environment [194–197]. Planning an exploratory action is achieved by relying on the sense of touch in a region of the human's spatial imagery where physical contact is expected [198].

There are several cases where a human cannot rely on vision alone, which makes haptic sensing purposeful to gain feedback about the state of the environment. In contrast to humans, technical systems are not naturally equipped with an integrated vision system. Furthermore, using a vision system is not always applicable. The following ten points list known limitations of visual perception in robotic manipulation:

1. occlusion and self-occlusion, i.e., the object of interest is not visible in the current frame of vision, e.g., because it is behind an object or hidden behind the robotic structure or tool.
2. poor eye-hand coordination, i.e., the object is visible in the workspace, but the eye-hand coordination behavior does not provide an appropriate camera pose to observe the scene, e.g., the camera moves too slowly relative to the hand movement.
3. adverse eye-hand kinematic, i.e., the object cannot be observed due to constraints of the eye-hand kinematic chain, e.g., joint limits prevent the camera from observing the scene where the arm is acting on.
4. weak illumination, i.e., poor or missing light sources do not allow visual perception.
5. low color gradients, i.e., the object of interest is in the line of sight but is not recognized due to missing contrast to its surroundings.
6. poor perception and inference capability, i.e., the perception algorithm has low accuracy in object recognition or frame extraction.
7. rough environmental conditions, i.e., hazards of the environment such as dust or mechanical impacts prevent the usage of a camera system.
8. task observation constraints, i.e., a scene must be visually observed while the arm acts simultaneously in another region out of sight, e.g., catching something requires observation of the flying object rather than the catching hand.
9. real-time constraints, i.e., the image processing speed given a required image resolution does not fulfill the task requirements.
10. resolution constraints, i.e., the image resolution or quality is too low to detect fine-grained geometrical features or spatial discontinuities.

Service robots or collaborative robots usually have to deal with unstructured or uncertain environments. An unstructured environment is defined as a world in which not all objects are known or where the locations of existing objects are unknown. Such environments exist or were usually built without consideration about how a robot would act in them. An uncertain environment is defined as a world in which all objects are known, but their locations can be uncertain, meaning that the estimated object location is a sample of a probability distribution around the actual object location.

Unstructured environments are outside the scope of this approach since they often require visual perception to deal with them. Although haptic exploration strategies exist for whole workspaces [96, 97], this work focuses on the intuitive bootstrapping of

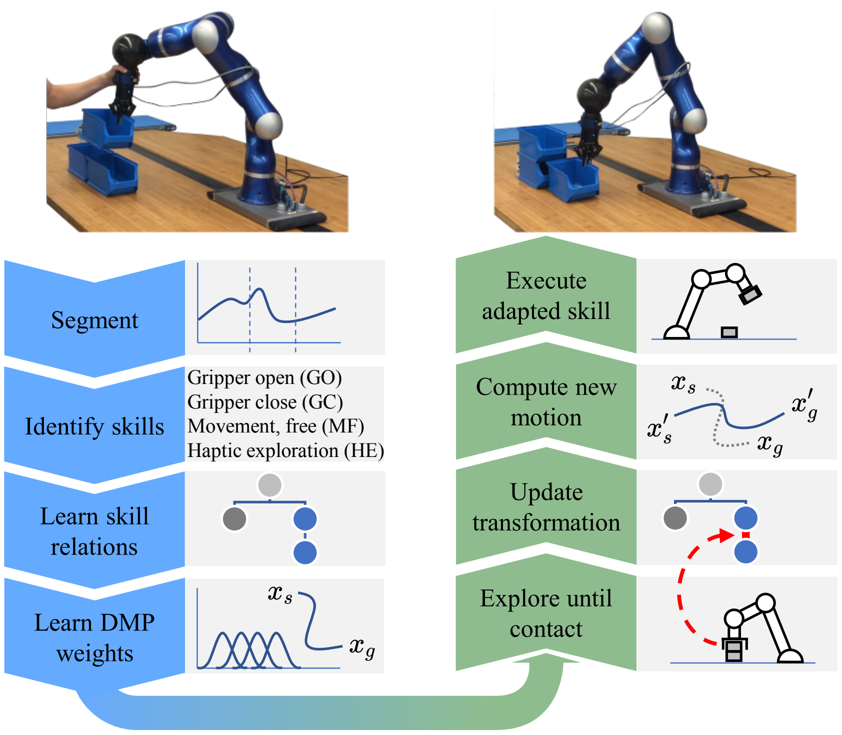


Figure 6.1: The skill demonstration, learning, and execution framework to execute parameterized motions. Previously observed constraints influence motion start and end points during exploration of the environment. All behaviors are extracted from human demonstrations via kinesthetic teaching.

an automation behavior, which targets uncertain environments as they could occur in a human-robot shared workspace. Examples are pick and place positions that depend on an object's position, such as a manually filled object store or a staple that varies in height.

Manually programming a robotic exploration strategy is cumbersome and requires expert knowledge about the programming method and the robot's capabilities and sensors. LfD is employed to simplify this procedure by extracting the desired behavior from the demonstration. In the proposed method, the user demonstrates the task consisting of exploration and manipulation motions multiple times. Each demonstration requires a change in object locations within their expected region during task execution. Pick and place actions are demonstrated by opening and closing the gripper with additional human input.

The presented approach segments at least two demonstrations of a task into parametrizable robot skills. If the task requires exploratory actions that the robot can haptically sense, so-called haptic exploration skills take care of environmental variations during task execution. The developed method extracts the position-based transformations between skills based on the variance observed from multiple demonstrations. Furthermore, a method is presented to learn the motions for the exploratory behavior, which is also extracted from the variations among multiple demonstrations.

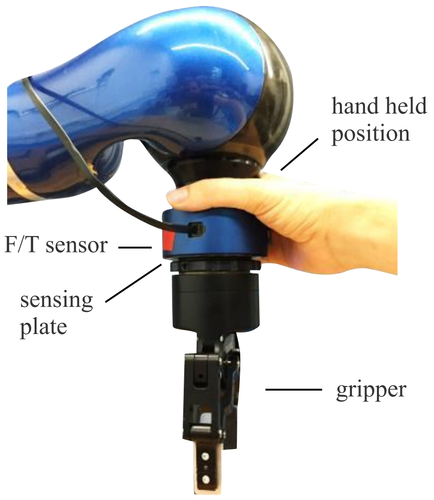
6.2 Skill Recognition from Human Demonstrations

The basic building blocks of the framework are robot skills that are predefined robot behaviors parameterized by a human demonstration via kinesthetic teaching. Fig. 6.1 shows an overview of the framework with the task demonstration starting at the top left. The task demonstration system is explained in Sec. 6.2.1. The task segmentation module explained in Sec. 6.2.2 takes demonstration data as input and produces labeled segments as output. The skill identification step explained in Sec. 6.2.3 constructs a sequence of skills from the previously labeled segments.

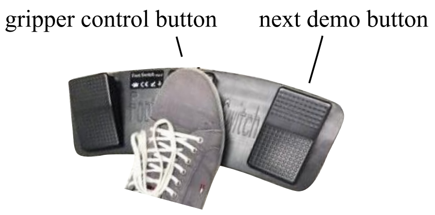
6.2.1 Task Demonstration System

The robot is guided by kinesthetic teaching where the user grasps the robot on a hold above the sensing plate of the FTS (Fig. 6.2a. The user can demonstrate exploratory movements by touching a part of the environment with any part of the end-effector, such as the gripper fingers. At this time, the gripper can be in opened or closed state. The contact forces between the gripper and environment are recorded by an FTS, mounted between the robot flange and gripper.

Each demonstration trial (Definition 3) is represented as the time series $\boldsymbol{X}^{[i]} \in \mathbb{R}^{N_i \times 15}$ as defined in eq. (3.1) with N_i samples corresponding to demonstration trial i. A demonstration sample at time t is denoted as $\boldsymbol{X}^{(t)} \in \mathbb{R}^{15}$ as defined in eq. (3.3). A number of I demonstration trials of the same task is stacked in the set $\boldsymbol{T} = \{\boldsymbol{X}^{[1]}, \ldots, \boldsymbol{X}^{[I]}\}$. The presented method requires a minimum of two demonstrations ($I \geq 2$) to exploit the variance among environmental changes. Ideally, the user demonstrations cover the



(a) End-effector and hand hold position



(b) Foot pedal for additional user input

Figure 6.2: Kinesthetic teaching by manually guiding the robot (a) and controlling gripper actions and demo recording by a foot pedal (b).

changes in the environment that are expected during task operation since the distance of an exploration path is derived from the boundaries faced in the demonstrations. The demonstration trials can differ in length depending on the user variations and teaching performance.

6.2.2 Task Segmentation

Pre-Processing The set of demonstrations $\boldsymbol{T}$ is pre-processed before segmentation into skills. This process is required to align temporal differences and equalize the demonstration trial length to find common segmentation points over all trials at once. In the first step, all phases in which neither the robot's linear nor angular velocity exceeds their respective thresholds are removed from the data. This optimizes the task reproduction since unnecessary waiting times are purged. Then, each wrench measurement time series $\boldsymbol{W}^{[i]}$ of trial i is filtered by a first-order low-pass Butterworth filter with a cut-off frequency of $f_{\text{co}} = 1\,\text{Hz}$.

DTW is used to temporally align the demonstration trials with each other [199]. The DTW standard formulation computes a warping path and a warping distance only between a pair of time series. In this work, each trial from the set $\boldsymbol{T}$ is aligned with a medoid trial $\boldsymbol{X}^{[m]}$, except the medoid trial itself that is part of the set. This medoid is found in two steps. First, the distances between all possible pairs of trials (i, j) are computed as

$$d_{\text{DTW}}^{[i,j]} = f(\boldsymbol{X}^{[i]}, \boldsymbol{X}^{[j]})$$

for $i, j \in [1, \dots, I]$. Second, the trial m with a minimum sum of squared distances to all other demos is found by

$$\boldsymbol{X}^{[m]} = \underset{k}{\operatorname{argmax}} \sum_{i=1}^{I} (d_{\mathrm{DTW}}^{[i,k]})^2 .$$

Finally, a warping path is computed for all trials with respect to the medoid trial. The warped trials are stored in $\boldsymbol{C} = [\tilde{\boldsymbol{X}}_{\mathrm{p}}, \tilde{\boldsymbol{X}}_{\mathrm{o}}, \tilde{\boldsymbol{W}}, \tilde{\boldsymbol{G}}] \in \mathbb{R}^{I \times N^{[m]} \times N_{\mathrm{d}}}$ with a number of I demonstration trials, $N^{[m]}$ samples, and N_{d} dimensions. Warped position and orientation time series are represented as $\tilde{\boldsymbol{X}}_{\mathrm{p}}$ and $\tilde{\boldsymbol{X}}_{\mathrm{o}}$ respectively and the warped wrench time series is denoted as $\tilde{\boldsymbol{W}} = [\tilde{\boldsymbol{F}}, \tilde{\boldsymbol{T}}]$ with forces $\tilde{\boldsymbol{F}}$ and torques $\tilde{\boldsymbol{T}}$. Further, the warped gripper finger distance is represented as $\tilde{\boldsymbol{G}}$.

The averages of $\boldsymbol{F}$ and $\boldsymbol{T}$ over demonstrations $i \in [1, \dots, I]$ are calculated as

$$\boldsymbol{F}_{\mathrm{m}} = \frac{1}{N^{[m]}} \sum_{i=1}^{I} \tilde{\boldsymbol{F}}, \quad \text{and} \quad \boldsymbol{T}_{\mathrm{m}} = \frac{1}{N^{[m]}} \sum_{i=1}^{I} \tilde{\boldsymbol{T}} .$$

Using these average force and torque trajectories in the subsequent segmentation reduces the impact of involuntary human actions. Here, the number and quality of demonstration trials influence the robustness of the segmentation. A larger number of demonstrations increase the robustness of the learning process as undesired forces are canceled out. On the other hand, querying too many demonstrations increases the risk of introducing an involuntary change in the sequence of actions compared to existing demonstration trials. Such involuntary actions are, for instance, multiple bumps when the user establishes contact with a rigid environment, accidentally touching the tool below the FT sensor with a human body part, or unintended force interactions between the tool and environment while picking and placing an object.

Segmentation of Contact States The contact state segmentation aims to differentiate between contact states and free movement. Here, two things are tackled: 1) finding segmentation points between no-contact and in-contact states, and 2) labeling the found segments with in-contact (IC) and no-contact (NC). Finding the segmentation points is achieved by predefined force and torque thresholds δ_f and δ_ϱ that are compared with the Euclidean norm of force and torque measurements, respectively. Algorithm 4 detects rising and falling edges in the force and torque domain. Whenever a rising edge is detected in one of the domains, the subsequent segment is labeled as IC. For every falling edge, the subsequent segment is labeled as NC.

The independent comparison of force and torque domains increases segmentation robustness as contact events can be triggered by any of these signals when establishing contact with the environment. Reaching the threshold in one domain is sufficient to trigger segmentation of an IC state, while both domains need to be below their thresholds to reset it, leading to the start of an NC segment. This strategy prevents a single contact from being over-segmented. For example, a single perpendicular contact of the gripper with an object can lead to torques and forces in the FTS where a rising and falling edge might be observed several times in each domain. While the gripper is still in contact, the algorithm would produce only one segment.

Algorithm 4 IC segmentation

```
Require: $\boldsymbol{F}_m$
 1: Initialization :
 2: $c \leftarrow 1$
 3: $IC \leftarrow$ false
 4: for $n = 1 \ldots N_n$ do
 5:     Detect rising edge:
 6:     if $(\|\boldsymbol{F}_{\mathrm{m,f}}(t-1)\| < \delta_f$ and $\|\boldsymbol{F}_{\mathrm{m,f}}(t)\| > \delta_f)$ or
 7: $(\|\boldsymbol{F}_{\mathrm{m,t}}(t-1)\| < \delta_\varrho$ and $\|\boldsymbol{F}_{\mathrm{m,t}}(t)\| > \delta_\varrho)$ and not
 8: $IC$ then
 9:         $\boldsymbol{IC}_{\mathrm{s}}(c) \leftarrow n$                ▷ add segmentation index to list
10:         $IC \leftarrow$ true                          ▷ set in-contact flag
11:         $c \leftarrow c + 1$
12:     end if
13:     Detect falling edge:
14:     if $(\|\boldsymbol{F}_{\mathrm{m,f}}(t)\| < \delta_f$ and $\|\boldsymbol{F}_{\mathrm{m,t}}(t)\| < \delta_\varrho)$ and
15: $IC$ then
16:         $\boldsymbol{IC}_{\mathrm{e}}(c) \leftarrow n$                ▷ add segmentation index to list
17:         $IC \leftarrow$ false                       ▷ reset in-contact flag
18:     end if
19: end for
20: return $\boldsymbol{IC}_{\mathrm{c}}, \boldsymbol{IC}_{\mathrm{e}}$
```

Segmentation of Gripper States This segmentation stage has the aims of 1) finding segmentation points where the gripper has been actuated within the NC segments from the previous segmentation stage and 2) labeling the found segments with gripper open (GO) and gripper closed (GC). All event's indices for gripper open (GO) $n_{\mathrm{GO}}^{[i]}$ and gripper close (GC) $n_{\mathrm{GC}}^{[i]}$ are extracted for each demo i and then averaged over demos, resulting in the averaged indices $\bar{n}_{\mathrm{GO}} = \frac{1}{I}\sum_{i=1}^{I} n_{\mathrm{GO}}^{[i]}$ and $\bar{n}_{\mathrm{GC}} = \frac{1}{I}\sum_{i=1}^{I} n_{\mathrm{GC}}^{[i]}$ respectively. Segments before these indices are labeled as GO or GC, accordingly. The demonstration data for segment s is stored in the matrix $\boldsymbol{S}_s = [\boldsymbol{P}_s, \boldsymbol{O}_s, \boldsymbol{W}_s] \in \mathbb{R}^{I \times N_s \times 13}$, with N_s samples from I demos. It contains position $\boldsymbol{P}_s^{[i]}$, orientation $\boldsymbol{O}_s^{[i]}$, and wrench $\boldsymbol{W}_s^{[i]}$ for each demonstration i. The segmentation result is stored as a set of all segments $\boldsymbol{S} = \{\boldsymbol{S}_1, \ldots, \boldsymbol{S}_{N_{\mathrm{S}}}\}$ with a number of N_{S} segments.

6.2.3 Skill Recognition

The system can identify a sequence of predefined robot skills, which are moving in free space (MO), haptic exploration (HE), and motions ending with gripper open (GO) or gripper close (GC). The skill recognition uses the previously labeled segments to infer a skill from the mapping shown in Fig. 6.3. A HE skill is recognized from the sequence of an NC and an IC segment. The skills of GC and GO are directly inferred from their corresponding GC and GO segments. An MO skill is recognized from a NC segment.

A recognized skill is denoted as $r \in [1, \ldots, N_{\mathrm{r}}]$. If an NC and an IC segment occur successively, a HE skill r is recognized and provided with the stacked data $\boldsymbol{R}_r = [\boldsymbol{S}_{\mathrm{NC}}, \boldsymbol{S}_{\mathrm{IC}}]^T$

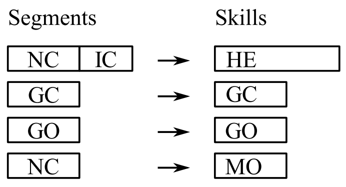


Figure 6.3: Mapping between segments and skills

of both segments. Using the data from both segments is required as the NC segment provides information to learn the motion and exploration strategy, and the IC segment provides information to infer the direction of the exploration of the HE skill. The remaining skills (GC, GO, MO) are provided with the data of their mapped segments (GC, GO, NC) according to Fig. 6.3. This mapping approach leads to a number of skills N_r that is less or equal to the original number of segments N_s, i.e., $N_\mathrm{r} \leq N_\mathrm{s}$. The identified skills are stored in the sequence $\boldsymbol{R} = (\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{N_\mathrm{r}})$.

6.3 Learning Adaptive Task Structures

6.3.1 Skill Relation Learning

After the skill recognition stage, the goal is to build an adaptive task structure that can adapt to changes in the environment. This goal is accomplished by using relative transformations between skill motions. More specifically, a skill motion can be executed 1) relative to another skill's motion frame or 2) relative to the origin $\{O\}$ (*root* frame). Therefore, a metric is proposed that pairwise compares all skills' motion frames to identify how strong they are spatially related to each other.

First, the goal position of a skill r is extracted for each demonstration i as

$$\boldsymbol{p}_{\mathrm{g},r}^{[i]} = \boldsymbol{R}_r^{[i](N_r)} \in \mathbb{R}^3,$$

where N_r is the number of samples of $\boldsymbol{R}_r$. This leads to I goal points for skill r, stored in the key-points matrix

$$\boldsymbol{K}_r = [\boldsymbol{p}_{\mathrm{g},r}^{[1]}, \ldots, \boldsymbol{p}_{\mathrm{g},r}^{[I]}]^T \in \mathbb{R}^{I\times 3}.$$

All key-points are added to a set of available key-points given by $\boldsymbol{L} = \{\boldsymbol{K}_1, \ldots, \boldsymbol{K}_{N_\mathrm{r}}\}$.

Next, a dimension-wise metric for the relation distance between skills m and n with their key-points $\boldsymbol{K}_m$ and $\boldsymbol{K}_n$ in dimension d over all demonstrations is given by

$$v_{m,n}^d = \mathrm{Var}\left(\begin{bmatrix} \boldsymbol{K}_m(1,d) - \boldsymbol{K}_n(1,d) \\ \vdots \\ \boldsymbol{K}_m(I,d) - \boldsymbol{K}_n(I,d) \end{bmatrix}\right) \quad \text{for } m \neq n. \tag{6.1}$$

The notation $K_m(1,d)$ refers to the element in matrix K_m in row 1 and column d. Similarly, the dimension-wise relation distance between a single skill m with key-point

$\boldsymbol{K}_m$ and the coordinate origin $\{O\}$ is given by

$$v_{m,n}^d = \mathrm{Var}\left(\begin{bmatrix}\boldsymbol{K}_m(1,d)\\ \vdots \\ \boldsymbol{K}_m(I,d)\end{bmatrix}\right) \quad \text{for } m = n. \tag{6.2}$$

The relation distance over all dimensions for all m, n is computed using the root mean squared error

$$\boldsymbol{V}(m,n) = \sqrt{\frac{1}{3}\sum_{d=1}^{3} {v_{m,n}^d}^2} \quad \text{for } \forall\, m, n. \tag{6.3}$$

Each skill now has a relation distance to any subsequent skill in the sequence. Consider a skill m whose behavior is influenced by a preceding skill k. Consequently, skill k minimizes the relation distance to the subsequent skill m in the skill sequence. The skill k is found from the relation distance matrix $\boldsymbol{V} \in \mathbb{R}^{N_\mathrm{r} \times N_\mathrm{r}}$ by

$$k = \underset{k \in \{1,\ldots,m\}}{\operatorname{argmin}} \{\boldsymbol{V}(m,k)\} \quad \text{with } k \leq m\,. \tag{6.4}$$

The notation $\boldsymbol{V}(m,k)$ refers to the matrix element of V in row m and column k. Constraining the search by $k \leq m$ considers only skills in the demonstrated sequence before skill m and excludes non-causal relations to future skills. In other words, a relation between skills is only searched for in the past since past events can influence future task behavior but not vice versa. The skill k is stored in a relation matrix $\boldsymbol{R}_\mathrm{rel} \in \mathbb{R}^{N_\mathrm{r} \times N_\mathrm{r}}$ by $\boldsymbol{R}_\mathrm{rel}(m,k) = 1$. In the case $m = k$, the skill is relative to the coordinate origin $\{O\}$ by setting $\boldsymbol{R}_\mathrm{rel}(m,m) = 1$, as it does not depend on any other skills. All other entries in $\boldsymbol{R}_\mathrm{rel}$ are set to 0.

A so-called relation tree is constructed from the relation matrix $\boldsymbol{R}_\mathrm{rel}$, representing a hierarchical structure of relative coordinate transformations. The transformed coordinates affect the skill's goal points. An exemplary relation tree is shown in Fig. 6.9, where each node represents a skill, and each arrow represents the direction of a coordinate transformation. This work considers only the translations between the skill's goal points. The translations are directed from a parent skill goal point to a child skill goal point and can be adapted during the execution by a HE skill. Since the HE skill has the capability to explore unknown environments, it will find a new contact point online, which leads to an adapted transformation to its attached children skills.

Skills that are directly attached to the *root* node (acting as the coordinate origin $\{O\}$) are executed with absolute motions, such as HE0 and GO3. The first skill of a task is always attached to *root* as any other preceding skill cannot influence it anymore. Skills attached to any other parent skill are executed with relative motions to their parent skills. For instance, the goal point of GC2 is defined to be relative to the explored contact point of HE1. Consequently, a single HE skill can affect multiple child skills. It is important to note that child skills can be executed right after its parent skill but also at any time later in the task. This means that the relation tree does not represent the sequence of skills but only the transformations between their frames. The sequential skill order is indicated by the number at the end of each skill label in Fig. 6.9

The relation tree is built by Algorithm 5. It uses a mean key-point

$$\boldsymbol{\mu}_p = \frac{1}{I}\sum_{i=1}^{I} \boldsymbol{K}_p^{[i]}$$

of a parent skill p to transform the position data $\boldsymbol{P}_c$ and key-points $\boldsymbol{K}_c$ of the child skill c.

Algorithm 5 Constructing the skill relation tree

```
Require: $\boldsymbol{R}, \boldsymbol{R}_{\text{rel}}$
1: for each child_parent_relation $c, p$ in $\boldsymbol{R}_{\text{rel}}(c,p)$ do
2:     $\boldsymbol{T}_{K_p}^{\mu_p} \leftarrow [\boldsymbol{\mu}_p, \dots, \boldsymbol{\mu}_p]^T - \boldsymbol{K}_p$      ▷ parent key-point translations
3:     $\boldsymbol{K}_c \leftarrow \boldsymbol{K}_c + \boldsymbol{T}_{K_p}^{\mu_p}$      ▷ transform child key-points
4:     $\boldsymbol{P}_c \leftarrow \boldsymbol{P}_c + \left[\boldsymbol{T}_{K_p}^{\mu_p}, \dots, \boldsymbol{T}_{K_p}^{\mu_p}\right]$      ▷ transform skill data
5:     ${}_p\boldsymbol{t}^c \leftarrow \text{mean}(\boldsymbol{K}_c - \boldsymbol{K}_p)$      ▷ parent child translation
6:     connect_nodes_with_trafo$(c, p, {}_p\boldsymbol{t}^c)$
7: end for
8: return $\boldsymbol{R}$
```

6.3.2 Skill Learning

The skills that execute free motions (MO, GO, GC) are encoded as DMPs that are learned from the data of multiple demonstrations. The HE skill is intended for contact-based exploration of uncertain environments where the contact point is unknown before execution. Therefore, a collision-aware exploration strategy is necessary, and the originally demonstrated motions must be modified.

Modeling the Exploration Motion An exploration motion allows the robot to explore the workspace outside the demonstrated region. It consists of two sections: 1) an approach path (AP) to reach the exploration area and 2) an extended exploration path (EEP) inside the exploration area (Fig.6.4). The subscript *HE* refers to any HE skill in the following. It is assumed that the variance in the demonstrations resembles the uncertainty in the environment, captured by the distribution of key-points. This uncertainty is modeled as normal distribution

$$\boldsymbol{K}_{\text{HE}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{HE}}, \boldsymbol{\Sigma}_{\text{HE}})$$

with mean and covariance

$$\boldsymbol{\mu}_{\text{HE}} = \frac{1}{I}\sum_{i=1}^{I} \boldsymbol{K}_{\text{HE}}^{[i]}; \qquad \boldsymbol{\Sigma}_{\text{HE}} = \text{Cov}(\boldsymbol{K}_{\text{HE}})\,.$$

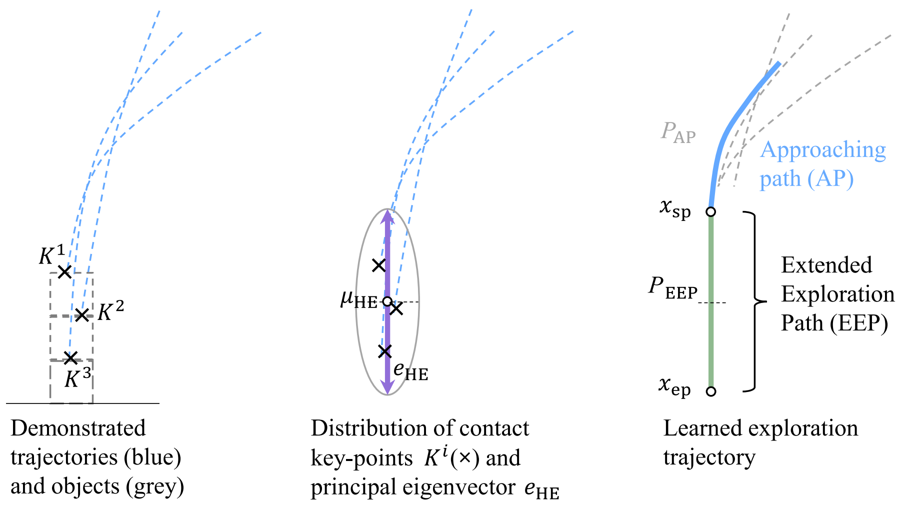

K^1
K^2
K^3
Demonstrated trajectories (blue) and objects (grey)
μ_{HE}
e_{HE}
Distribution of contact key-points $K^i(\times)$ and principal eigenvector e_{HE}
P_{AP}
Approaching path (AP)
x_{sp}
P_{EEP}
Extended Exploration Path (EEP)
x_{ep}
Learned exploration trajectory

Figure 6.4: Schema of multiple demonstrations in a changing environment (left), the distribution over extracted key-points with principal component eigenvector (middle) and learned exploration trajectory (right).

The proposed algorithm requires at least two demonstrations to exploit the variance in the data. More than two demonstrations might be intended to cover a wider exploration area or to transfer a more complex exploration behavior. The HE skill uses a linear exploration path to find an object in the workspace. This exploration path is identified with the help of a principal component analysis, using the covariance matrix $\boldsymbol{\Sigma}_{\mathrm{HE}}$. Its principal eigenvector $\boldsymbol{e}_{\mathrm{HE}}$ is termed as main exploration vector (Fig. 6.4 middle). However, the principal eigenvector's direction might be anti-parallel to the demonstrated paths and desired linear exploration direction. The true direction of the main exploration vector shall be aligned with the demonstrated motions and is found by the direction of the contact force vector of the corresponding data segment. The maximum force vector is denoted as $\hat{\boldsymbol{F}}_{\mathrm{IC}}^{[i]}$ for each demo i. Examples of these vectors can be found in Fig. 6.10. Then, the average force vector over all demos is computed as

$$\bar{\boldsymbol{F}}_{\mathrm{IC}} = \frac{1}{I}\sum_{i=1}^{I} \hat{\boldsymbol{F}}_{\mathrm{IC}}^{[i]} .$$

The alignment with the main exploration vector $\boldsymbol{e}_{\mathrm{HE}}$ is ensured using the sign of the dot product $\bar{\boldsymbol{F}}_{\mathrm{IC}} \cdot \boldsymbol{e}_{\mathrm{HE}}$. This allows to determine if the exploration direction points against the expected contact force. If positive, the main exploration vector $\boldsymbol{e}_{\mathrm{HE}}$ must be inverted to align with the approaching path. The position trajectory $\boldsymbol{P}_{\mathrm{EEP}}$ of the EEP is constructed by connecting the exploration start point $\boldsymbol{x}_{\mathrm{sp}} = \boldsymbol{\mu}_{\mathrm{HE}} - k_{\mathrm{e}}\boldsymbol{e}_{\mathrm{HE}}$ and the exploration endpoint $\boldsymbol{x}_{\mathrm{ep}} = \boldsymbol{\mu}_{\mathrm{HE}} + k_e\boldsymbol{e}_{\mathrm{HE}}$ with linearly spaced points, where k_e is a parameter to scale the exploration region (Fig. 6.4 right). In contrast to the position, the orientation is kept fixed during the whole EEP, taken from the quaternions average [200] at the contact points.

The motion of the HE skill consists of two phases. In the first phase, the robot follows the AP until it reaches the exploration region. The data $\boldsymbol{P}_{\mathrm{AP}}$ for the AP is constructed from samples from the beginning of the corresponding segment until the index where a sample falls below a distance to the exploration start point $\boldsymbol{x}_{\text{sp}}$ (Fig. 6.4 right). In the second phase, the robot follows the EEP where $\boldsymbol{x}_{\text{sp}}$ is a desired via-point before the robot reaches the exploration endpoint $\boldsymbol{x}_{\text{ep}}$. The final definition of the input data of a HE skill is then $\boldsymbol{P}_{\text{HE}} = [\boldsymbol{P}_{\text{AP}}, \boldsymbol{P}_{\text{EEP}}]$. An example can be found in Fig. 6.10.

Motion Encoding Each skill r is encoded by a GMM as described in Sec. 3.3. The model encodes multiple demonstrations $i \in [1, \dots, I]$ consisting of position $\boldsymbol{P}_r^{[i]}$ and orientation $\boldsymbol{O}_r^{[i]}$, and a time vector $\boldsymbol{t} \in \mathbb{R}^{N_r}$ in a single model. Expectation Maximization fits a number of Gaussian distributions on the input data R_r^{in} that consists of all stacked demonstrations

$$\boldsymbol{R}_r^{\text{in}} = \begin{bmatrix} \boldsymbol{P}_r^{[1]} & \boldsymbol{O}_r^{[1]} & \boldsymbol{t} \\ \vdots & \vdots & \vdots \\ \boldsymbol{P}_r^{[I]} t & \boldsymbol{O}_r^{[I]} & \boldsymbol{t} \end{bmatrix} \in \mathbb{R}^{N_r \times 8}. \tag{6.5}$$

GMR (Sec. 3.3) reproduces the generalized trajectory

$$\boldsymbol{R}_r^{\text{out}} = [\boldsymbol{P}_r^{\text{out}}, \boldsymbol{O}_r^{\text{out}}] \in \mathbb{R}^{N_r \times 7}\,. \tag{6.6}$$

The quaternions in the orientation trajectory $\boldsymbol{O}_r^{\text{out}}$ are normalized to unit quaternions after the GMR.

The trajectory obtained in $\boldsymbol{R}_r^{\text{out}}$ is then used to learn a DMP, which is used to produce a stable motion that is guaranteed to converge to a desired goal point [201]. This work adopts the approach from [202] to learn DMPs for position and orientation based on quaternions. It uses the quaternion logarithmic and exponential map, which does not violate the constraints of the orientation group ($SO3$) manifold. The DMPs enable the adaptation of their start and goal point given the transformations that are stored in the relation tree. The DMP weights for a skill r are learned from its motion data $\boldsymbol{R}_r^{\text{out}}$ generated by the GMR.

6.3.3 Task Execution and Skill Sequencing

The adaptive execution scheme is implemented in Algorithm 6, which generates motions with parameterized start $\boldsymbol{x}_{\text{s}}$ and goal point $\boldsymbol{x}_{\text{g}}$. The coordinate origin is termed as $\{O\}$. During the execution of a HE skill, a contact is detected when the exploration force or torque exceeds a defined threshold. Whenever a contact is registered, the controller stops the current motion. The contact point $\boldsymbol{x}_c$ is extracted, and the goal point translation between this skill and its parent skill is updated in the relation tree (Alg. 6, line 13). If required, object positions can be explored in multiple dimensions. For instance, this could be necessary for a structured arrangement of objects or a single object in a plane. As relative skill's motions are adapted by their parent skills, multiple HE skills can be chained to allow more complex exploration behaviors, which is shown in the following experiments section.

Algorithm 6 Adaptive task execution

```
 1: for $r = 1 \dots N_\mathrm{r}$ do
 2:     $p \leftarrow$ get_parent_skill($r$)
 3:     if skill $r$ is HE skill then
 4:         $\boldsymbol{x}_\mathrm{g} \leftarrow$ get_trafo$(O, p) + \boldsymbol{P}_r^\mathrm{out}(N_{\mathrm{n},r}) - \boldsymbol{P}_r^\mathrm{out}(1)$
 5:     else
 6:         $\boldsymbol{x}_\mathrm{g} \leftarrow$ get_trafo$(O, r)$
 7:     end if
 8:     $\boldsymbol{x}_\mathrm{s} \leftarrow$ get_robot_pos()
 9:     execute_skill$(r, \boldsymbol{x}_\mathrm{s}, \boldsymbol{x}_\mathrm{g})$
10:     if skill $r$ is HE skill then
11:         $\boldsymbol{x}_\mathrm{c} \leftarrow$ get_contact_pos()
12:         $\boldsymbol{t}_p^c \leftarrow \boldsymbol{x}_\mathrm{c} -$ get_trafo$(O, p)$
13:         update_trafo$(r, p, {}_p\boldsymbol{t}^c)$
14:     end if
15: end for
```

6.4 Experiments

6.4.1 Hardware and Setup

A DLR LWR4 [6] robot was used in all experiments, mounted on a linear axis and equipped with a two-finger Robotiq-85 gripper. A FTS manufactured by JR3 was mounted between robot tip and gripper. The measured external wrench at the FTS was compensated by the tool mass. Therefore, it resembled only the contact force and torque between tool and environment. The robot was operated in Cartesian impedance control with 800 N/m linear and 80 Nm/rad angular stiffness. Skill learning and sequencing were implemented in *Python* and low-level control in C++ communicating with *Matlab/Simulink* models.

6.4.2 Experimental Task

The experimental task was to pick boxes from an uncertain location and place them in a fixed, demonstrated location. Fig. 6.5 shows the experimental environment, which contains a 2-dimensional arrangement of boxes with four possible box locations. The number of boxes can vary from one to four, and the boxes can be stapled in different configurations. The experimenter gave two demonstrations applied on a subset of two box configurations. This is sufficient to allow the system to exploit the variance among two dimensions. Moreover, it is the minimum number of demonstrations to let the algorithm work while keeping the teaching effort low. In the first demo, boxes were set at locations p11, p21, p22. In the second demo, only one box was located at p11. Figure 6.6 shows that in each demonstration, one box was touched in the horizontal and vertical direction as an exploration strategy to identify its position. The demonstrator started the first exploration motion horizontally at the bottom row because it is guaranteed to find a box when one is available on the table. Starting the exploration behavior in the

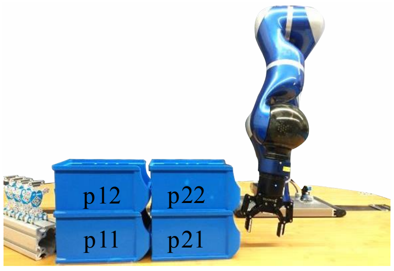


Figure 6.5: Arrangement of boxes at locations p11, p12, p21, p22 and approaching robot executing first skill HE0.

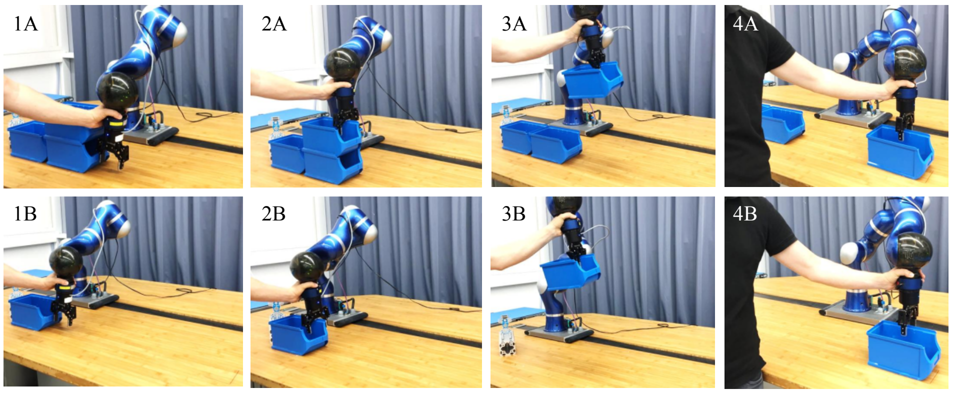


Figure 6.6: Demonstration of the adaptive task. In demonstration A, the demonstrator touched the bottom row of boxes horizontally (1A) and the upper box of the first staple vertically (2A). The box was picked (3A) and placed (4A) in a fixed position on the table. After rearranging boxes for demonstration B, the demonstrator touched the residual box in horizontal (1B) and vertical (2B) directions, picked it (3B), and placed it (4B) at the same position on the table.

vertical direction to explore a column of boxes would not succeed whenever this column contains no boxes.

6.4.3 Results

The two task demonstrations were commonly segmented into contact states and free movements and thereafter segmented by their gripper states. The segmentation result with according segment labels is shown in Fig. 6.7. These segment labels were then mapped to a sequence of skills: (NC, IC) $\rightarrow$HE0; (NC, IC) $\rightarrow$HE1; (GC) $\rightarrow$GC, (GO) $\rightarrow$GO, (NC) $\rightarrow$MO. The number after each skill label denotes the sequential order during task execution.

Skill relations were learned based on the relation distance matrix $\boldsymbol{V}$ (Fig. 6.8 and are represented in the tree structure in Fig. 6.9. The position data $\boldsymbol{P}_r$ and learned motion of both HE skills (HE0, HE1) are shown in Fig. 6.10. The shown motion and according key-points of HE1 are already transformed by Algorithm 5. The system can

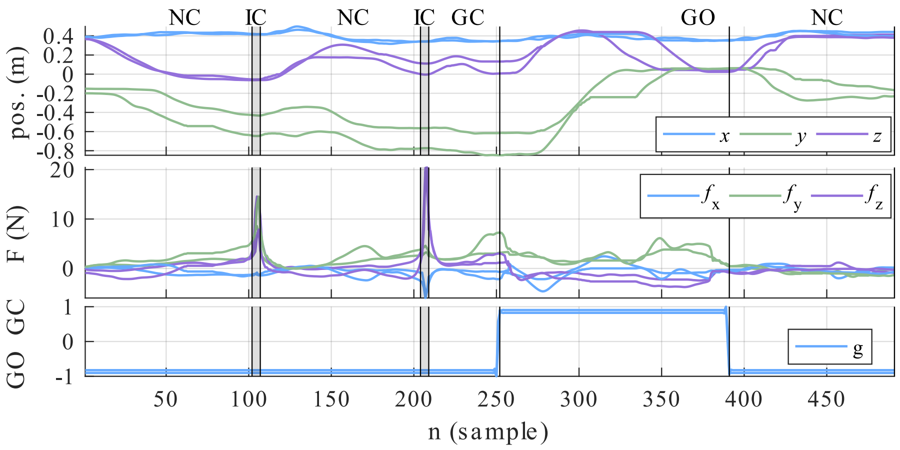


Figure 6.7: Segmented task with labeled segments.

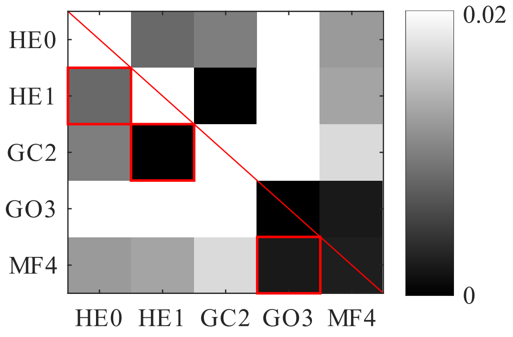


Figure 6.8: Relation distance matrix $\boldsymbol{V}$ with extracted relations, where the red boxes mark which skill (in each row) is relative to a skill in the column.

generalize to unseen box arrangements during execution. Figure 6.11 shows the motion and contact forces for a new box configuration {p11, p12, p21}. The pick position at $t = 21\,\mathrm{s}$ was adapted based on the detected state of the environment from the contact-based explorations at $t = 4.8\,\mathrm{s}$ and $t = 12\,\mathrm{s}$. The fixed place position at $t = 32\,\mathrm{s}$ was reached with a fixed goal as demonstrated at sample 385 in Fig. 6.7. This experiment proved that no prior knowledge about the initial object configuration was needed and no knowledge about the involved object's geometry needed to be modeled. However, this required that the same objects were consistently used in demonstration and execution.

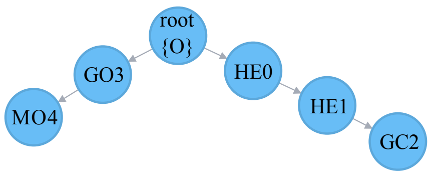
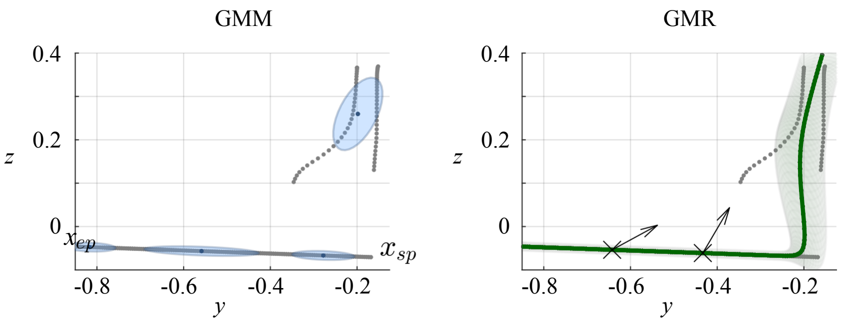
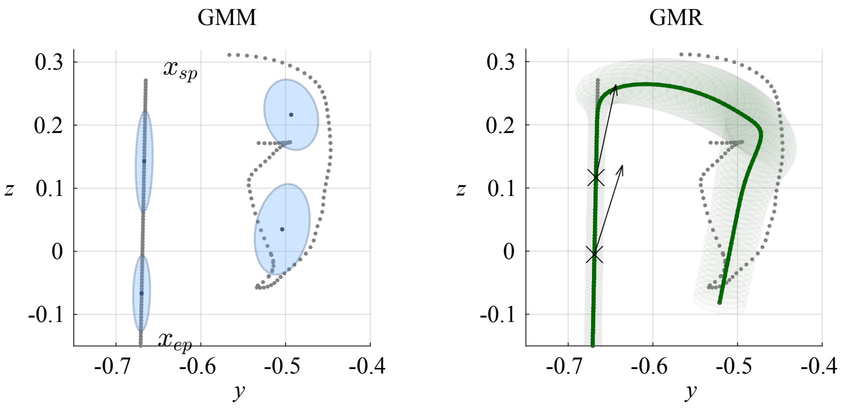


Figure 6.9: Relative skill tree from the box experiment. The goal point transformation between two skills is a directed relationship from parent to child skill.



(a) Skill HE0 exploring in approximately horizontal direction of the workspace.



(b) Skill HE1, which is executed relative to HE0, exploring in approximately vertical direction.

Figure 6.10: Haptic exploration skills HE0 and HE1 exploring in two different directions of the workspace. Shown are position samples $\boldsymbol{R}_{\mathrm{p},r}$ (grey dots), GMM covariance ellipses (blue), key-points ($\times$), and learned GMR motion (green). Black arrows show magnitude and direction of $\hat{\boldsymbol{F}}_{\mathrm{IC}}^{[i]}$.

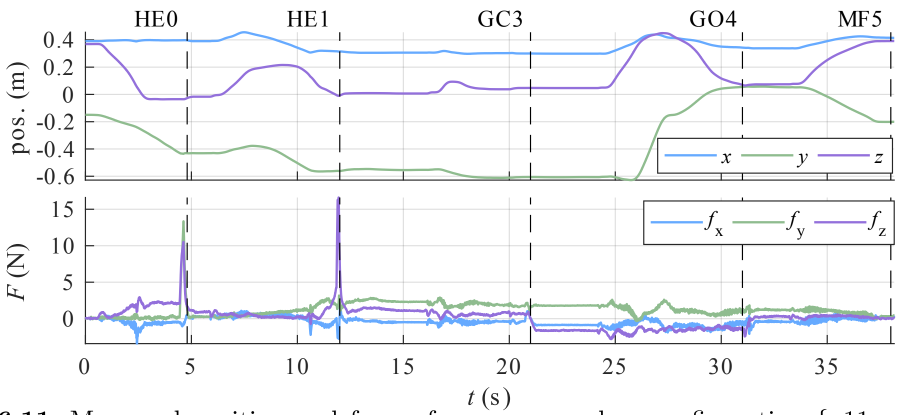


Figure 6.11: Measured position and forces for an unseen box configuration {p11, p12, p21}. Dashed lines show where each skill terminated. Detected contacts by the HE skills stopped the robot movement and adapted the pick position at GC3. The place position at GO4 was reached with a fixed goal.

6.5 Discussion and Conclusion

6.5.1 Discussion

The learned adaptive task allows to explore a workspace with so-called haptic exploration skills via linear exploration paths according to the demonstrator's desired exploration strategy. Theoretically, each object might be manipulated after exploring its location if it is touchable and graspable at the same points as the corresponding object used in the demonstration. The exploration strategy is also tailored to the requirements of the demonstrator and, therefore, considers only relevant parts of the workspace during execution. This strategy differs from approaches that explore a whole workspace [96,97], which is rarely needed in industrial applications. However, the presented approach does not provide advanced exploration behaviors if an object is not contacted at all in the current exploration skill. If the object's location is outside of the modeled exploration region, it would not be found by the HE skill.

The exploration does not consider object orientations and cannot estimate them. Estimating the object orientation might come with the additional burden of modeling the geometry of each involved object. The presented approach does not require any object to be geometrically modeled. It is also independent of the number of involved objects. Therefore, the approach is easy to implement in new environments. Approaches that assign motions to specific reference frames usually first model objects in the world and then associate reference frames with them. For instance, [80] assigns motions to reference frames of objects. During execution, these frames can be tracked, for example, by means of visual perception. This requirement differs from the presented approach, which does not depend on any object tracking approach.

6.5.2 Conclusion

The introduced framework consists of a demonstration system, a segmentation method, and a skill learning approach to execute adaptive tasks. The framework relies on so-called haptic exploration skills that use linear exploration motions to identify objects locations during task execution. It was shown that learning of an adaptive task structure is possible within only two demonstrations. The segmentation approach uses averaged demonstration data with the aim of increasing robustness. The skill learning method considers environmental constraints such that the exploration path starts at a point outside of the demonstrated contact points to avoid unexpected collisions. The haptic exploration behaviors were validated with two exploration dimensions and can adapt skills online in a systematically changing environment. The proposed framework does not require visual perception and prior knowledge about the involved objects, and purely relies on the robot's sense of touch.

The experiments also showed that the strategy of the haptic exploration skill substituted parts of the human demonstrations. In detail, the demonstrated approach path from the user was substituted by the extended exploration path in the execution. This use case shows that parts of the human demonstration can be optimized by the strategy embedded in the skill according to the task objectives.

The haptic exploration skill capabilities of the framework are limited to exploring object positions only, without estimating an object's orientation. This works well if the environment is partially structured and objects are, for instance, arranged in a stack or a staple. A contact based full pose estimation including object orientation could be addressed in future work. However, a possible use-case should first justify if camera-free object pose estimation is needed or if the system capabilities can possibly be extended with a camera. Future work could also assess the mental load of providing multiple demonstrations with consistent actions that vary only spatially with respect to the objects that shall be explored in the task. Although the demonstration effort is kept to a minimum, it is hardly known how well approaches that require multiple demonstrations would be received from end-users, for instance, in industrial working environments.

7 Task Decision Programming

Conditional tasks include a decision on how the robot should react to an observation. This problem requires the selection of an appropriate action during execution by a so-called task decision. Such events can either be foreseen by the user or detected by the system in case of unexpected task faults. Then, recovery behaviors must be selected during execution to account for the error. Intuitive programming of these use cases via PbD is termed task decision programming.
This chapter includes content from the following publications [193, 203, 204]:

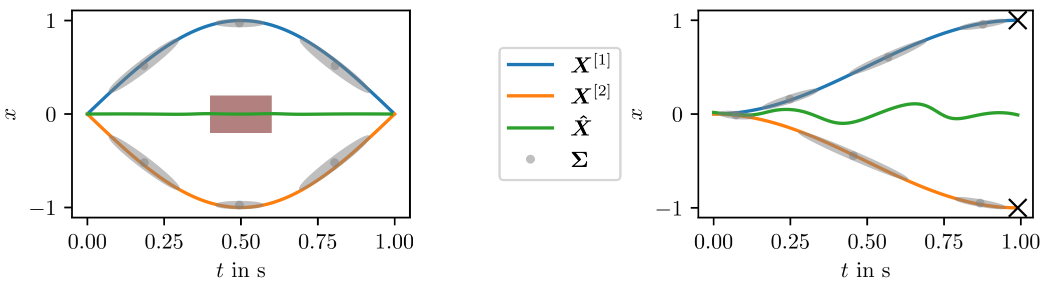
- T. Eiband, M. Saveriano, and D. Lee, "Intuitive Programming of Conditional Tasks by Demonstration of Multiple Solutions," *IEEE Robotics and Automation Letters*, vol. 4, Art. no. 4, Oct. 2019.

- C. Willibald, T. Eiband, and D. Lee, "Collaborative Programming of Conditional Robot Tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5402–5409.

- T. Eiband, C. Willibald, I. Tannert, B. Weber, and D. Lee, "Collaborative Programming of Robotic Task Decisions and Recovery Behaviors," *Autonomous Robots*, pp. 1–19, 2022.

The author of this thesis conducted the conceptual development, contributed with the anomaly detection method from [203], conducted additional experiments with respect to [204], and planned a user study to compare the developed approaches. M. Saveriano adviced in the method development. C. Willibald developed the task graph construction concept and implemented it in the experimental system. I. Tannert and B. Weber proposed the statistic design of the user study. I. Tannert conducted the user study, assisted by C. Willibald. D. Lee advised in developing research methodologies and analyzing the results and revised the article.

This chapter is structured as follows. Section 7.1 introduces the fundamentals for the problem of task decision programming. Section 7.2 proposes the concept of *multiple solutions* to program conditional tasks and recovery behaviors. This concept allows the system to switch to an alternative behavior anytime during execution. Section 7.3 adapts this concept to construct task graphs with the method of *collaborative programming*. These task graphs allow online decision-making at specific points during execution and can be incrementally extended or refined. Section 7.4 presents a user study that compares the proposed approaches. The findings are concluded in Sec. 7.5.



(a) Avoiding an obstacle (marked as red box). **(b)** Approaching different goal points (marked as ×).

Figure 7.1: Two examples of encoding two demos each ($\boldsymbol{X}^{[1]}$, $\boldsymbol{X}^{[2]}$) in a single GMM with Gaussian components represented as covariance ellipses $\boldsymbol{\Sigma}$. Both examples do not showcase their initial objectives in the reproduced trajectory $\hat{\boldsymbol{X}}$ obtained by GMR.

7.1 Fundamentals

From the user's perspective, there are examples of conditional tasks, such as sorting objects by specific object properties. An exemplary task is sorting objects by their weight by picking each object and putting the object in tray A if it is light and in tray B if it is heavy. Another example is performing a task in the workspace, and after detecting that specific conditions are not met, e.g., parts are missing to complete the subtask, the robot continues with another subtask. Examples of recovery behaviors are sorting out defective parts based on a measurable property and continuing the nominal task with the next available healthy part. Another example is to mate assembly parts, and if the mating is unsuccessful, the robot tries another behavior given the current condition.

Although there might be a different understanding of what a conditional task or a recovery behavior is from the user's perspective, the introduced methods of *multiple solutions* (Sec. 7.2) and *collaborative programming* (Sec. 7.3) handle these use cases equally. From the robot's perspective, a task should follow an expected behavior. If an observed condition deviates from the expectation, the robot can react to this condition by selecting and executing an appropriate behavior, which is also called *solution*. Therefore, the following sections do not differentiate between the terms *solution* and *recovery behavior*.

7.1.1 Introductory Example

Several works in the field of LfD make use of multiple demonstrations (Def. 7) for the same task. This strategy can be meaningful for learning algorithms to increase robustness, exploit task variations, or enable task generalization, such as in task parameterization approaches. Often, multiple demonstrations are merged to learn a single model. Depending on the application, this might cause problems, as shown in the two examples in Fig. 7.1. In both examples, a GMM is fitted with six Gaussian components on two demonstrations ($\boldsymbol{X}^{[1]}$, $\boldsymbol{X}^{[2]}$) together with a time vector $\boldsymbol{t}$. GMR is used to reproduce a trajectory $\hat{\boldsymbol{X}}$ by conditioning the model's distribution on the time t. Figure 7.1a shows an obstacle avoidance example, where the learned trajectory cancels out the intended variations of both demos. Similarly, in Fig. 7.1b, two demonstrations that end in different goal positions lead to a learned trajectory that ends at a position in between the demonstrated goal positions.

These issues indicate that a single machine-learning model cannot seamlessly encode the objectives from multiple demonstrations. Although task parameterization models [205, 206] account for different coordinate frames, these frames have to be known or be observable by the system, which is not always given. Further, they do not generalize well if, as for conditional tasks, the required action is too different from what has been demonstrated. Additionally, it may be hard for a novice user to select task parameters manually or to implement a higher-level abstraction layer that accounts for decisions or recovery behaviors. Therefore, it is suggested that a system learns individual models for diverse task conditions and automatically selects an appropriate model that matches the currently observed situation.

7.1.2 Requirements

This chapter proposes that a task decision and recovery behavior programming framework needs the following properties:

(i) An anomaly detection mechanism,

(ii) an extendable knowledge representation allowing to learn from the user and environment,

(iii) correctability of the robotic actions to increase robustness, and

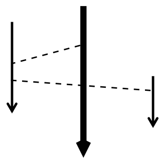
(iv) an adaptive system to react during task execution. (Sec. 7.3.5).

With reference to the above, (i) anomaly detection is required to determine if the system is behaving as expected. (ii) A knowledge representation is needed to store existing and future information in a structured format that both humans and robots can interpret. (iii) Correctability of existing actions is required, such that the system can either learn from experiences or human feedback. (iv) An adaptive system can react to changes that occur during task execution and that cannot be foreseen at the starting time of the task.

7.1.3 Task Representations

Reactive behaviors are required for both task decisions and fault recovery. Therefore, two fundamental task representation types are introduced in Fig. 7.2 and used in the following sections. It is argued that fault recovery and conditional tasks are closely related because they require (a) monitoring of the execution, (b) branching from the nominal execution flow, and (c) multiple actions for each decision and recovery behavior.

Solution Pool This task representation was introduced in a previous work [203] and represents a set of multiple actions, so-called solutions, that are already known to the


(a) Solution pool (SP), with a nominal solution (thick arrow) and possible transitions (dashed lines) to alternative solutions.

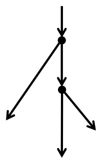

(b) Task graph where links represent actions and nodes represent decision states.

Figure 7.2: A task representation that incorporates recovery behaviors can be defined as solution pool, where multiple solution actions exist in parallel (a) or as task graph, which arranges the actions as links and decision states as nodes (b).

system (Fig. 7.2a). In the solution pool (SP), no branching states are specified. Instead, transitioning between solutions is possible at any time during execution.

Task Graph The presented framework uses a structured task graph (Fig. 7.2b) that employs specified decision states, which are the graph's nodes. The links represent the robotic actions that either lead to the next decision state or a designated task termination. Later in this chapter, it is explained how this representation can be generated incrementally in an interactive scheme involving the user and robot.

7.1.4 Anomaly Detection Mechanisms

In the presence of possible task faults, the end-user wants the robot to handle such situations autonomously. In reality, it might not be always clear to the robot what is exactly a fault state. However, a human might have capabilities the robot does not have to identify such states. Therefore, both manual and autonomous detection mechanisms are considered in this chapter.

Manual Anomaly Detection This mechanism relies on the humans to use their perception capabilities to identify states where the robot is in an unexpected or faulty state or where it should make a decision about its following action [58,134]. This information can be obtained from a user who observes the task execution and provides manual feedback, e.g., via a button or GUI.

Autonomous Anomaly Detection This mechanism automatically detects unexpected or faulty states in the task execution. It removes the burden on the user to observe the task execution and react accordingly. It enables the detection of abnormal states in the absence of the user and of newly occurred situations that could not be foreseen at programming time. Other approaches focus on identifying low confidence task regions to improve the robot's spatial generalization capabilities [207]. Instead, the focus lies on identifying anomalies that can occur in the position and force domain. The anomaly

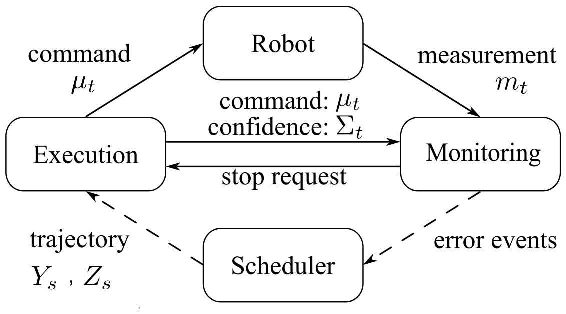


Figure 7.3: System components for real-time execution and monitoring. Solid connections are real-time-capable up to 1 kHz, dashed connections are slow asynchronous connections.

detection scheme was proposed in a previous works [203] and is based on a probabilistic action encoding and statistical outlier detection using the Mahalanobis distance.

7.1.5 Demonstration System

The user starts the task definition process by providing multiple demonstrations for varying task conditions, which the robot shall account for. The demonstration system presented in [42] is used to transfer knowledge to the robot that is equipped with an FTS mounted between the robot and gripper. The system provides three foot pedals that trigger the start and stop of teaching, open and close the gripper, and switch to the next demonstration. A sample at time t of demonstration i is given by $\boldsymbol{X}^{[i](t)}$ (3.3). The wrench is filtered by a 1st-order Butterworth low-pass filter with a 1 Hz cutoff frequency. The samples of demonstration $i \in \{1, \ldots, I\}$ with sample length N_i are stored in a matrix $\boldsymbol{X}^{[i]} = [\boldsymbol{X}^{[i](1)}; \ldots; \boldsymbol{X}^{[i](N_i)}]$. Steady states with nearly no change in position and orientation are removed.

7.1.6 System Architecture

Fig. 7.3 shows an overview of the system architecture. The task of each module is explained subsequently.

Scheduler A high-level decision module without real-time capability that handles events that occur during task execution. It selects an action when the system reaches a decision state or when the *monitoring* detects an anomaly during the execution. Then, it passes the trajectory and covariance time series of the selected action to the *execution*. Depending on the method and situation, it makes a decision autonomously or actively queries input from the user.

Execution This module commands a Cartesian Impedance controller with overlaid wrench term. The joint torque is controlled as

$$\boldsymbol{\tau}_{\mathrm{cmd}} = \boldsymbol{J}^T(\boldsymbol{K}(\boldsymbol{x}_{\mathrm{d}} - \boldsymbol{x}) + \boldsymbol{w}_{\mathrm{d}} - \boldsymbol{D}\dot{\boldsymbol{x}}) + \boldsymbol{g}(\boldsymbol{q}), \tag{7.1}$$

where $\boldsymbol{J}$ is the Jacobian, $\boldsymbol{K}$ the Cartesian stiffness matrix, $\boldsymbol{x}_\mathrm{d}$ and $\boldsymbol{x}$ the desired and measured position, $\boldsymbol{w}_\mathrm{d}$ the desired wrench, $\boldsymbol{D}$ a positive definite damping matrix and $\boldsymbol{g}$ represents the robot's gravity term, depending on the joint position $\boldsymbol{q}$. The overlaid wrench $\boldsymbol{w}_\mathrm{d}$ accounts for the demonstrated dynamics and external wrenches observed in the demonstrations. The gripper finger distance g is commanded by a simple feed-forward controller with a predefined maximum velocity and grasp force. The grasp status h is only streamed to the robot for monitoring purposes.

Monitoring This module observes both the nominal state, which is the command sent to the robot, and the measured state in real-time during execution. Anomalies are detected by a one-class classification approach [208] via a probability distribution of the expected state and an anomaly distance metric. Whenever an anomaly is detected during execution, an event with relevant process data is sent to the *scheduler*. The commanded state $\boldsymbol{Y}^{(t)}$ and measured states $\boldsymbol{M}^{(t)}$ at time step t are defined as:

$$\boldsymbol{Y}^{(t)} = [\boldsymbol{P}^{(t)}, \boldsymbol{O}^{(t)}, \boldsymbol{W}^{(t)}, \boldsymbol{G}^{(t)}] \in \mathbb{R}^{15}$$

$$\boldsymbol{M}^{(t)} = [\boldsymbol{M}_\mathrm{p}^{(t)}, \boldsymbol{M}_\mathrm{o}^{(t)}, \boldsymbol{M}_\mathrm{w}^{(t)}, \boldsymbol{M}_\mathrm{g}^{(t)}] \in \mathbb{R}^{15}$$

The deviation between nominal execution and measured state is computed at each time step t by the Mahalanobis distance

$$D_\mathrm{M} = \sqrt{(\boldsymbol{X}^{(t)} - \boldsymbol{M}^{(t)})(\boldsymbol{Z}^{(t)})^{-1}(\boldsymbol{X}^{(t)} - \boldsymbol{M}^{(t)})^T}\,, \tag{7.2}$$

with $\boldsymbol{Z}^{(t)}$ being a matrix that describes the covariance of the data dimensions in the commanded trajectory at time t. The distance D_M is compared with a threshold ϵ at each time step. Consequently, a large enough deviation from the commanded state signalizes the failure of the nominal solution. If the threshold is exceeded for a few consecutive time steps, an anomaly event is triggered, which stops the *execution* module in real-time. This event and the measured erroneous state $\boldsymbol{x}_\mathrm{e}$ at time step t are sent to the *scheduler*. All modalities contribute equally to the anomaly detection as the error that they are causing is scaled by the covariance matrix. Specifically, these errors can be introduced by

i) deviation in position or orientation;
ii) abnormal wrench due to unexpected interaction forces or object weights
iii) deviation in gripper finger distance due to unexpected object geometry; or
iv) mismatch of grasp status due to misplaced objects in the environment.

7.2 Multiple Solutions

This section includes content of [203]. The framework introduced in the following allows a user to demonstrate conditional tasks, including recovery behaviors for expected situations. The user can demonstrate several solutions corresponding to different executive conditions that construct the robot's behavior. Each solution accounts for different task conditions that may arise during execution. The demonstrations are clustered, and each cluster is assigned to a specific solution, which is then encoded in a probabilistic model. At runtime, the robot monitors the pose, external wrench, and grasp status of the current solution. Deviations from the expected state are classified as anomalies. This triggers the execution of an alternative solution appropriately selected from the solution pool (SP). The programming paradigm of this framework is called SBP since the teaching and execution phases can follow sequentially after each other. Batch programming refers to the user's option to provide a batch of demonstrations that account for different solutions.

In summary, the presented framework:

i) learns a variety of task solutions without labeled data or a symbolic task representation by clustering human demonstrations,

ii) switches online between solutions by anomaly detection in the motion and force domain, and

iii) recovers from an error by choosing the most likely state within the set of alternative solutions.

7.2.1 Overview

The execution of a conditional task is affected by the executive context, meaning that a conditional task has several possible outcomes depending on the conditions faced during execution. Therefore, a task is not necessarily a fixed temporal sequence of skills or actions. Instead, it can be a variable execution strategy that considers environmental changes. For instance, sorting objects by their weight, where light and heavy objects should be placed on two different destinations, would require two different solutions as shown in Fig. 7.4. In addition, the sorting example requires physical interaction with the objects to estimate their weight. This object property cannot be observed visually prior to task execution. The correct execution of a conditional task requires continuous monitoring of the execution state and the ability to detect anomalies, i.e., deviations between the expected state of the current (nominal) solution and the measured state. Such anomalies can trigger the execution of an alternative solution that accomplishes the task. A suitable solution is found by identifying the best alternative solution in the SP (Fig. 7.4, right). In the teaching and execution phases, the robot's proprioceptive sensing is exploited without incorporating a vision system. This constraint requires a partially structured environment since object frames cannot be tracked online. Each task is bootstrapped from the demonstrations without requesting further knowledge about task goals. In line with that, neither a world model nor any object model is required in the learning and execution phases.

One or multiple human demonstrations build each solution required for task completion. Fig. 7.5 shows an example of multiple demonstrations encoded in two solutions. Since a solution can be learned from multiple demonstrations, a clustering algorithm

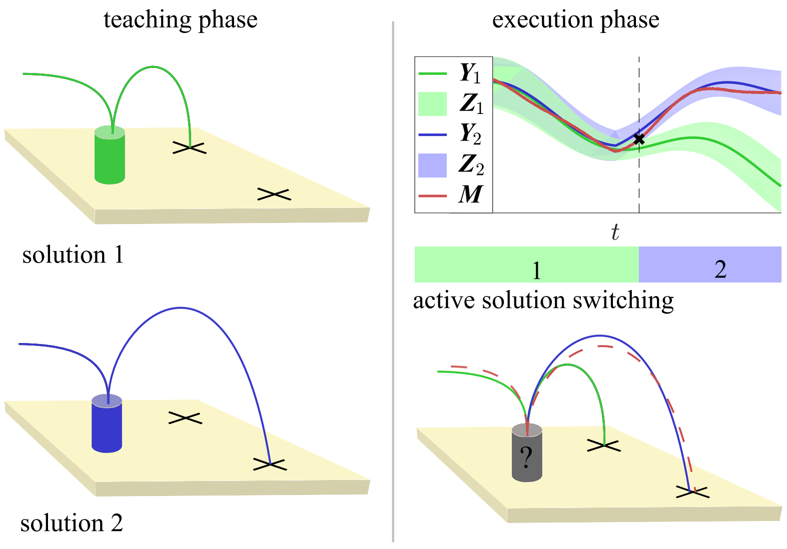


Figure 7.4: Teaching multiple solutions for different conditions in a task, e.g., sorting by object weight. The robot executes a nominal solution (solution 1) and monitors measured state $\boldsymbol{M}$ and commanded state $\boldsymbol{Y}_1$. The scheduler selects an alternative solution (solution 2) with commanded state $\boldsymbol{Y}_2$, when the current solution's confidence bound $\boldsymbol{Z}_1$ is violated.

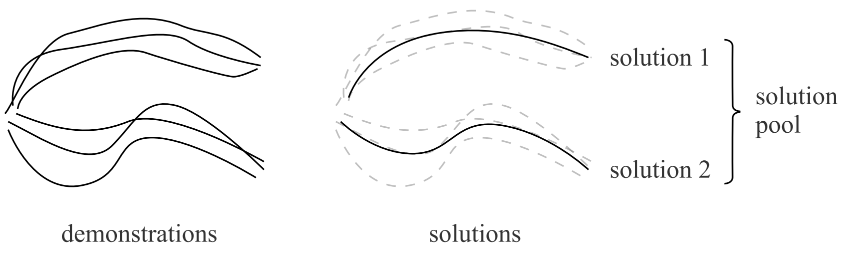


Figure 7.5: Example of six demonstrations encoded in two solutions, which are added to a SP.

can assign multiple demonstrations to a specific solution in an unsupervised fashion, i.e., without explicitly labeling each demonstration. Multiple solutions are then included in a SP. Here, it does not matter whether an execution that deviates from the target execution is considered a recovery behavior or an alternative solution. Similarly, the event that leads to the adjustment of the execution can be considered a failure or simply an expected environmental condition that deviates from the nominal execution.

During execution, alternative solutions can be executed by switching from the initial nominal solution to the state in the alternative solution where the error between the environmental condition and the expected state is minimized. This strategy can be compared to a behavior tree where the precondition of an inactive branch is continuously observed until it is activated when the condition becomes true.

All observable state variables of the robotic system can be taken into account for anomaly detection. The presented framework follows a vision-free approach focusing only the proprioceptive measurements of the robot, which are 1) robot pose, obtained by joint position sensors; 2) wrench, obtained by an external FTS; and 3) gripper finger distance in combination with a discrete grasp status (-1: no object in gripper, 0: gripper moving, 1: object in gripper) provided by the gripper interface.

7.2.2 Clustering

Several unlabeled demonstrations can be clustered to find a set of solutions (Fig. 7.5). First, all demonstrations are vertically stacked in $\boldsymbol{X} = [\boldsymbol{X}^{[1]}; \dots ; \boldsymbol{X}^{[I]}]$. In a preprocessing stage, all demonstrations are standardized dimension-wise by a z-transform. Let

$$z_{n,d} = \frac{x_{n,d} - \bar{x}_{*,d}}{\sigma_{x_{*,d}}},$$

with $x_{n,d}$ being the element of $\boldsymbol{X}$ in row n and column d, $\bar{x}_{*,d}$ being the mean and $\sigma_{x_{*,d}}$ being the standard deviation over column d of $\boldsymbol{X}$. Computing $z_{n,d}$ for each sample n and dimension d results in the standardized demonstrations $\{\bar{\boldsymbol{X}}^{[1]}, \dots, \bar{\boldsymbol{X}}^{[I]}\}$. This step is required to handle all dimensions with the same importance in the DTW step to contribute equally to the warping error. Otherwise, signals with relatively high amplitudes, such as forces and torques, would significantly impact the warping error compared to position values. Next, a pairwise distance matrix between all possible pairs of demonstrations is computed by the DTW distance. This distance is denoted as $\mathrm{DTW}(\boldsymbol{A}, \boldsymbol{B})$, for some multi-dimensional time series $\boldsymbol{A}$ and $\boldsymbol{B}$. The distance matrix over all demonstrations is given by

$$\boldsymbol{D}_{\mathrm{DTW}} = \begin{bmatrix} \mathrm{DTW}(\bar{\boldsymbol{X}}^{[1]}, \bar{\boldsymbol{X}}^{[1]}), \cdots, \mathrm{DTW}(\bar{\boldsymbol{X}}^{[1]}, \bar{\boldsymbol{X}}^{[I]}) \\ \ddots \\ \mathrm{DTW}(\bar{\boldsymbol{X}}^{[I]}, \bar{\boldsymbol{X}}^{[1]}), \cdots, \mathrm{DTW}(\bar{\boldsymbol{X}}^{[I]}, \bar{\boldsymbol{X}}^{[I]}) \end{bmatrix}.$$

The distance matrix $\boldsymbol{D}_{\mathrm{DTW}}$ serves as input to single linkage hierarchical clustering [209]. The system requests the user to specify the desired number of solutions S, used as cluster quantity, where each cluster corresponds to an individual solution. A number of S clusters is obtained by flattening the hierarchical cluster structure, where a minimum threshold is computed on the cophenetic distance between two observations in the same cluster such that no more than S flat clusters are created.

Once the clusters are identified, DTW is applied again by computing a warping path between each demonstration and its cluster's medoid. The cluster medoid is found by the minimum sum of squared distances to all other demos within the same cluster, given as

$$\underset{k\in\{1,\ldots,C_s\}}{\operatorname{argmin}} \sum_{l=1}^{I_s} (d_{\mathrm{DTW}\,k,l})^2 \,,$$

with I_s being the number of demonstrations in cluster s, and $d_{\mathrm{DTW}\,k,l}$ referring to the DTW distance between the k-th and l-th demonstration within the current cluster. The resulting warping path is used to realign all demons of a cluster with their medoid. The warped demos of solution s are resampled to share the same length of N_s within one cluster and are vertically stacked into the matrix $\bar{\boldsymbol{C}}_s = [\tilde{\boldsymbol{X}}_s^{[1]}; \ldots ; \tilde{\boldsymbol{X}}_s^{[I_s]}]$.

7.2.3 Trajectory Learning

Each cluster corresponding to solution s consists of one or multiple demonstrations and is converted into a generalized trajectory. The data has been standardized (Sec. 7.2.2), whereas the original scaling of the data is used in the trajectory learning stage. Therefore, the inverse z-transform is applied by column-wise multiplication with the standard deviation and addition of the mean, resulting in a non-standardized cluster $\boldsymbol{C}_s$. The number of data dimensions is $D = 15$.

A time-based GMM is learned for each solution s to generalize multiple demos. The input matrix

$$\boldsymbol{G}_s = \begin{bmatrix} \boldsymbol{C}_s & \begin{bmatrix} \boldsymbol{u} \\ \vdots \\ \boldsymbol{u} \end{bmatrix} \end{bmatrix} \in \mathbb{R}^{N_s I_s \times D+1} \tag{7.3}$$

serves as input to learn a joint probabilistic model, given a time vector $\boldsymbol{u} = [1, \ldots , N_s]^T$ that is added to each demo in C_s.

See Sec. 3.3 about learning the GMM using Expectation Maximization that fits a number of N_G Gaussian components on the data. The model complexity is defined proportionally to the length of the time series, i.e., $N_G = \frac{N_s}{f_s}$ with f_s as the sample rate of the time series. GMR (Sec. 3.3) provides a generalized trajectory $\boldsymbol{Y}_s \in \mathbb{R}^{N_s \times D}$ and a time series of covariance matrices $\boldsymbol{Z}_s \in \mathbb{R}^{N_s \times D \times D}$ by conditioning the model on the time vector $\boldsymbol{u}$. The trajectory of each solution s is stored in $\boldsymbol{Y}_s = [\boldsymbol{Y}^{(1)}; \ldots ; \boldsymbol{Y}^{(N_s)}]$ and the according covariance time series in $\boldsymbol{Z}_s = [\boldsymbol{Z}^{(1)}; \ldots ; \boldsymbol{Z}^{(N_s)}]$. A symmetric covariance matrix at time step t is represented as

$$\boldsymbol{Z}^{(t)} = \begin{bmatrix} \boldsymbol{Z}^{(t)}_{\mathrm{p,p}} & \boldsymbol{Z}^{(t)}_{\mathrm{p,o}} & \boldsymbol{Z}^{(t)}_{\mathrm{p,w}} & \boldsymbol{Z}^{(t)}_{\mathrm{p,g}} & \boldsymbol{Z}^{(t)}_{\mathrm{p,h}} \\ \dots & \boldsymbol{Z}^{(t)}_{\mathrm{o,o}} & \boldsymbol{Z}^{(t)}_{\mathrm{o,w}} & \boldsymbol{Z}^{(t)}_{\mathrm{o,g}} & \boldsymbol{Z}^{(t)}_{\mathrm{o,h}} \\ \dots & \dots & \boldsymbol{Z}^{(t)}_{\mathrm{w,w}} & \boldsymbol{Z}^{(t)}_{\mathrm{w,g}} & \boldsymbol{Z}^{(t)}_{\mathrm{w,h}} \\ \dots & \dots & \dots & \boldsymbol{Z}^{(t)}_{\mathrm{g,g}} & \boldsymbol{Z}^{(t)}_{\mathrm{g,h}} \\ \dots & \dots & \dots & \dots & \boldsymbol{Z}^{(t)}_{\mathrm{h,h}} \end{bmatrix}. \tag{7.4}$$

The subscripts of $\boldsymbol{Z}$ refer to each modality, with p as position, o as orientation, w as wrench, g as gripper finger distance, and h as grasp status. The quaternions of the orientation trajectory are normalized at this point. Since GMR and DTW act in Euclidean vector space, they do not preserve the quaternion properties that are only valid in $SO3$. The deviation between original quaternions and regressed and normalized quaternions is small enough to justify the usage of such a compact orientation encoding alongside the other modalities. The trajectory learning stage is applied on each solution s, and the trajectories of $\boldsymbol{Y}_s$ and $\boldsymbol{Z}_s$ are added to the SP.

7.2.4 Online Solution Selection

The scheduler manages a SP, from which it selects an appropriate solution in case this is triggered by the monitoring module during task execution. Its behavior is depicted in Algorithm 7. Two types of events are expected, which are either an anomaly or task completion. There are a few suggested strategies to select the initial nominal solution for a task from the SP:

i) The scheduler selects the solution whose cluster contains the first demonstration given by the human.
ii) The user chooses the solution to start with.
iii) The scheduler selects the solution with the minimum number of samples to favor short execution times and simple behaviors as employed in the experimental evaluation of this chapter.

Once an anomaly has been detected by the *monitoring* module, the *scheduler* selects an appropriate alternative solution that lies closest to the abnormal state. The trajectory and covariance time series of the selected solution is then passed to the *execution*. Suppose only two solutions are in the SP, and one is currently executed. In that case, only one solution remains to be executed, and the problem of selecting an alternative becomes trivial. If there are more than two solutions in the SP, the *scheduler* shall select the best strategy to cope with the current abnormal situation.

Let the *monitoring* module identify an abnormal state $\boldsymbol{X}^{(e)}$ at time step $t = e$ given a nominal solution η and let $\boldsymbol{Y}_s^{(t)}$ be the sample in the trajectory of solution s at time step t and $\boldsymbol{Z}_s^{(t)}$ the associated covariance matrix. Then, the minimum squared Mahalanobis distance over each sample in solution s is given by

$$C_s = \min_{t \in [1, N_s]} \{(\boldsymbol{Y}_s^{(t)} - \boldsymbol{X}^{(e)})(\boldsymbol{Z}_s^{(t)})^{-1}(\boldsymbol{Y}_s^{(t)} - \boldsymbol{X}^{(e)})^T\}. \tag{7.5}$$

Algorithm 7 Scheduler

```
Require: solution pool: SP; anomaly threshold: $\epsilon$
 1: Initialization :
 2: $t \leftarrow 1$                                          ▷ set trajectory starting index
 3: $s \leftarrow$ get_nominal_solution(SP)                    ▷ described in this section
 4: while not_empty(SP) do
 5:     goto_start_point($s, t$)
 6:     execute_solution($s$)          ▷ implemented in execution module (Sec. 7.1.6)
 7:     remove solution $s$ from pool SP
 8:     event $\leftarrow$ wait_for_event($\epsilon$)   ▷ implemented in monitoring module (Sec. 7.1.6)   ▷
    anomaly or finished task
 9:     if event is anomaly then
10:         if not_empty(SP) then
11:             given: measured erroneous state: $\boldsymbol{x}_\mathrm{e}$
12:             $s, t \leftarrow$ find_alternative_solution($\boldsymbol{x}_\mathrm{e}, s$)
13:                                                            ▷ described in Sec. 7.2.4
14:         else
15:             return stop_on_error                       ▷ no more solutions available
16:         end if
17:     else
18:         return finished                                     ▷ successful completion
19:     end if
20: end while
```

and the solution with the closest state to the erroneous state is extracted as follows

$$s^* = \underset{s \in \mathrm{SP} \setminus \eta}{\operatorname{argmin}} \{C_s\} \ . \tag{7.6}$$

It excludes the currently executed nominal solution η. The identified alternative solution trajectory is stated as $\boldsymbol{Y}_{s^*}$.

There are two additional requirements for the alternative solution. First, it shall be executed right after the error is detected with the smallest possible time delay. Second, it shall start at a point to resolve the error without executing it from the beginning. Therefore, the time step in which the alternative solution starts is identified similarly to (7.5) with

$$t^* = \underset{t \in [1, N_s^*]}{\operatorname{argmin}} \{(\boldsymbol{Y}_{s^*}^{(t)} - \boldsymbol{X}^{(e)})(\boldsymbol{Z}_s^{(t)})^{-1}(\boldsymbol{Y}_{s^*}^{(t)} - \boldsymbol{X}^{(e)})^T\}. \tag{7.7}$$

This step leads to the construction of a trimmed solution trajectory that the *scheduler* switches to at runtime, defined as $\boldsymbol{Y}_{s^*} = [\boldsymbol{Y}^{(t^*)}; \boldsymbol{Y}^{(t^*+1)}; \ldots; \boldsymbol{Y}^{(N_{s^*})}]$.

7.2.5 Experiments

The following experiments were performed on a DLR Light Weight Robot (LWR IV) [6] mounted on a linear axis, equipped with a 2-finger Robotiq 85 gripper.

Evaluation of Monitoring System A baseline experiment was conducted to evaluate that a system without monitoring capabilities cannot detect anomalies from an expected behavior and that the proposed monitoring system can do so. The anomaly threshold was chosen to be $\epsilon = 6$ in the following experiments. The task was to pick a peg object and insert it into a hole, as shown in Fig. 7.6. Only small external forces were expected between the peg and hole for the nominal solution. The user provided four demonstrations of picking and inserting the peg in the hole. From that, a single cluster was identified (7.2.2) and a single solution was learned 7.2.3.

The results show how errors from different modalities contributed to the overall error D_{M}. Therefore, positional errors are defined by

$$D_{\mathrm{p}} = \sqrt{(\boldsymbol{P}^{(t)} - \boldsymbol{Y}_{\mathrm{p}}^{(t)})(\boldsymbol{Z}_{\mathrm{p,p}}^{(t)})^{-1}(\boldsymbol{P}^{(t)} - \boldsymbol{Y}_{\mathrm{p}}^{(t)})^T}$$

and force errors are defined by

$$D_{\mathrm{f}} = \sqrt{(\boldsymbol{F}^{(t)} - \boldsymbol{Y}_{\mathrm{f}}^{(t)})(\boldsymbol{Z}_{\mathrm{f,f}}^{(t)})^{-1}(\boldsymbol{F}^{(t)} - \boldsymbol{Y}_{\mathrm{f}}^{(t)})^T} \ ,$$

where $\boldsymbol{F}^{(t)}$ is the measured force, $\boldsymbol{Y}_{\mathrm{f}}^{(t)}$ the commanded force, and $\boldsymbol{Z}_{f,f}^{(t)}$ the sub-matrix of the wrench covariance $\boldsymbol{Z}_{w,w}^{(t)}$. The gripper finger distance and grasp status errors are defined by D_{g} and D_{h} respectively. In the *monitoring* module, the entire state space is used to compute D_{M} as denoted in (7.2).

In the following, three execution runs were conducted. In the first run, the robot executed the nominal solution without obstacle in the hole (Fig. 7.6 (c)) The execution

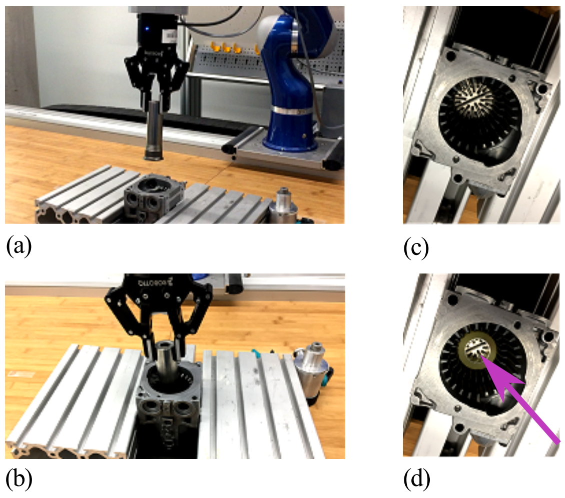


Figure 7.6: Robot is approaching the hole (a) and inserts the peg with expected forces during the nominal solution (b). On the right, the empty hole (c) and the hole with the obstacle marked with a purple arrow (d) are shown.

led to success regardless of monitoring since there is no abnormal configuration in the setup (Fig. 7.7). In the second run, an obstacle was inserted in the hole (Fig. 7.6 (d), purple arrow) to construct an abnormal configuration of the setup. The task was then run by a system without monitoring capability (Fig. 7.8), which resulted in the robot executing the whole commanded motion without detecting the failed insertion, potentially damaging the robot or the object. In the third run, the task was rerun with the inserted obstacle, the same as in the second run, but while the monitoring system was activated. Consequently, the *monitoring* detected the anomaly and stopped the robot to prevent further damage and signalized that something went wrong during execution (Fig. 7.9). At this state, no recovery behaviors were available in the SP. Hence, the error could not be resolved.

Conditional Task: Weight-based Sorting This task of sorting objects by weight can be solved by the nominal solution or by switching to an alternative solution that resolves the abnormal state. Figure 7.10 shows the experimental setup. Full milk cartons shall be placed in the left box, and empty milk cartons shall be dropped in the right box. For this purpose, the user gave three demonstrations each for the full and empty carton setup. The user specified that the number of desired solutions is two since the robot shall either place the cartons in the left or in the right box. The clustering stage assigned three demonstrations to the solutions labeled as *full_carton* and *empty_carton*.

In the execution phase, the nominal solution was *full_carton*, which is successfully executed if a full carton is present. Figure 7.11 shows the force in z-axis and the monitored errors. In the next run, an empty carton was present, and the nominal solution *full_carton* caused an anomaly, as can be seen from the force measurements and monitored error in Fig. 7.12. Here, a switch occurs from *full_carton* to *empty_carton*. It

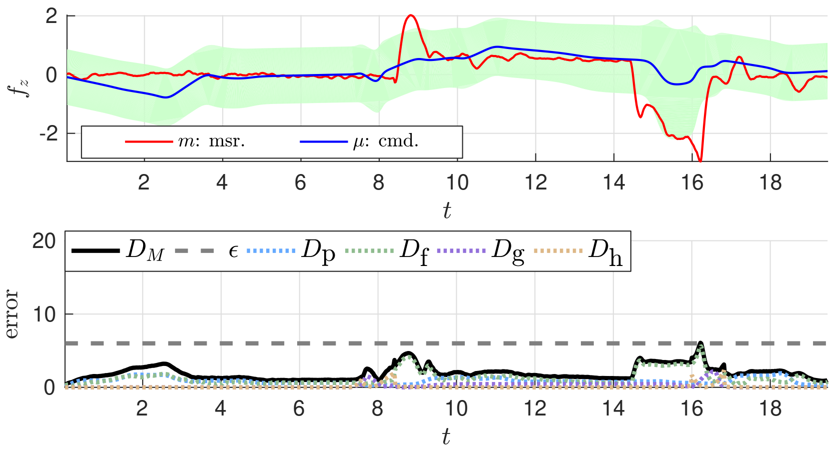


Figure 7.7: Peg-in-hole experiment: First run with nominal environment, where running with or without monitoring does not affect the execution. Top: Measured (m) and commanded (μ) force in z-axis and ± 2 standard deviations around commanded state (green). Bottom: Monitored error in different domains with anomaly threshold set to $\epsilon = 6$.

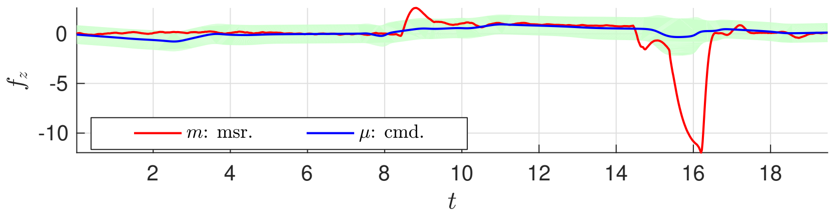


Figure 7.8: Second run with obstacle but without monitoring, leading to an undetected error at around 16 s. Plot shows measured (m) and commanded (μ) force in z-axis and ± 2 standard deviations around commanded state (green).

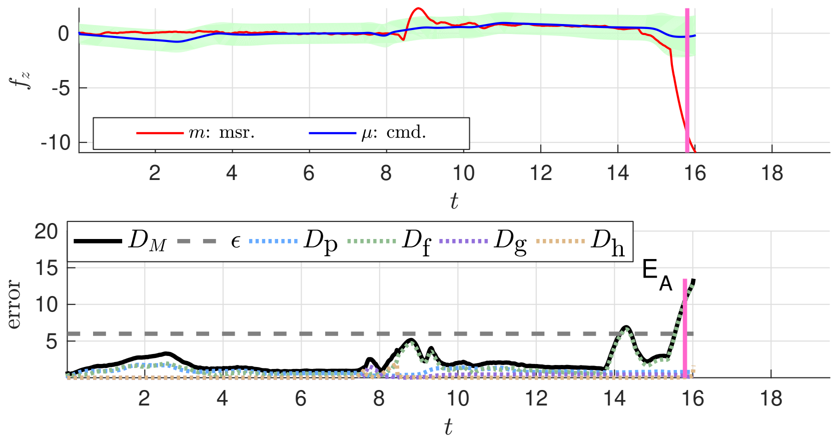


Figure 7.9: Third run with obstacle and active monitoring, stopping the robot at the abnormal state (event $\mathrm{E_A}$ marked by vertical purple bar). Top: Measured (m) and commanded (μ) force in z-axis and ± 2 standard deviations around commanded state (green). Bottom: Monitored errors in different domains.

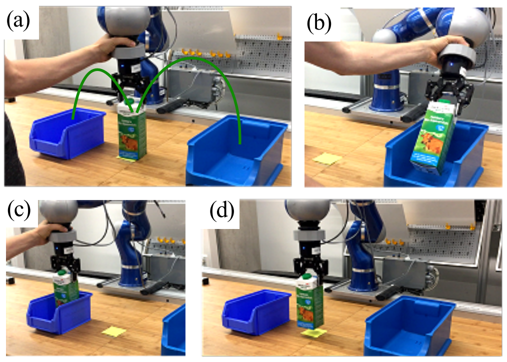


Figure 7.10: (a) Human demonstration at carton pick location with motion paths in green; (b) empty milk carton disposed at the right box; (c) full milk carton placed in the left box; (d) robot execution while lifting the carton after picking it.

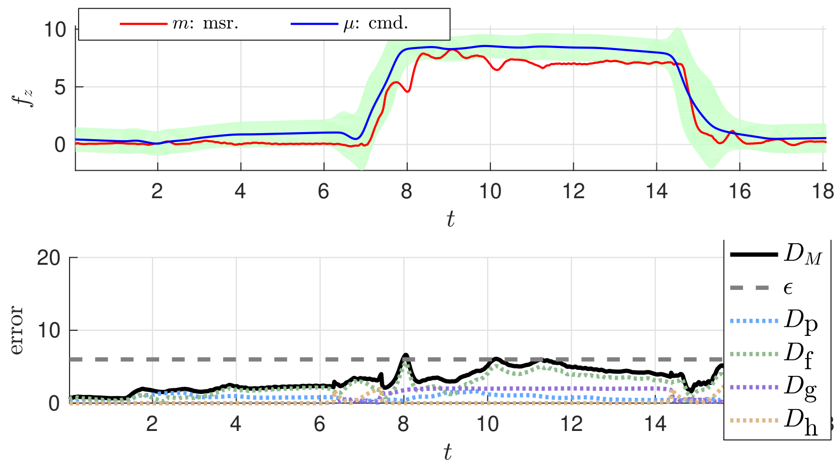


Figure 7.11: Top: Measured (m) and commanded (μ) force in z-axis for execution with solution *full_carton* and ± 2 standard deviations around commanded state (green). Bottom: Monitored error in different domains. No anomaly was detected in the nominal solution.

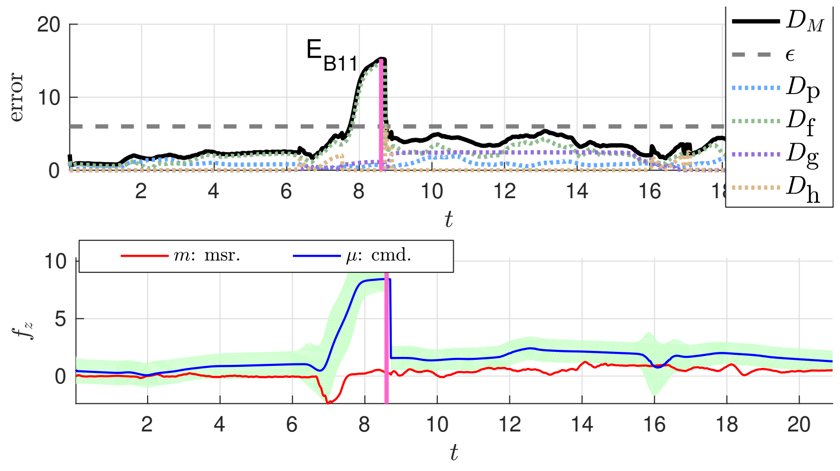


Figure 7.12: Top: Measured (m) and commanded (μ) force in z-axis for execution with solution *full_carton* but environment with empty carton, while monitoring was active. In green, ± 2 standard deviations around the commanded state. Bottom: Anomaly during execution at event $\mathrm{E}_{\mathrm{B}11}$ caused the switch from *full_carton* to *empty_carton*. Vertical purple bars mark the switching event.

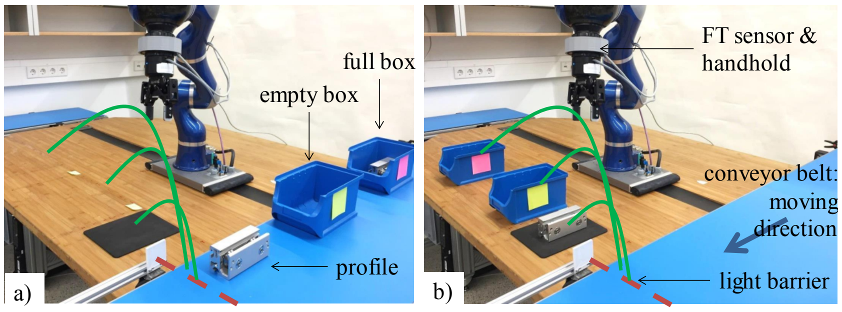


Figure 7.13: (a) Possible initial setup; (b) target configuration. The conveyor belt moved in the direction of the blue arrow until an object interfered with the light barrier (red dashed line).

is emphasized that the visual appearance of the full and empty carton does not allow humans to detect their state by vision. Hence, only the robot's force measurements allowed the system to make a decision.

Switching with Multiple Alternatives This experiment shows that switching to not only one but multiple alternative solutions is possible, which may refer to an intended task goal or a recovery behavior. The experiment was carried out in a partially structured environment where objects with the same properties (e.g., geometry and weight) were located at the same position during execution. Figure 7.13 shows the experimental setup. A conveyor belt (in blue, to the right side of the robot) was used to deliver new objects in a random sequence. It stopped whenever an object entered the light barrier and restarted after the object had been removed. The teaching phase consisted of nine demonstrations, showing three different behaviors to the robot. The demonstrations were given in a random order, specified by the sequence of objects arriving at the conveyor belt. In three demonstrations according to solution *empty_box*, the user showed the desired behavior to sort empty boxes (mass $m = 0.15\,\text{kg}$, marked with a yellow square) to a desired goal pose. Similarly, three more demonstrations were provided for solution *full_box*, where full boxes (mass $m = 0.85\,\text{kg}$, marked with a pink square) were placed at a different goal pose, according to Fig. 7.13. Three more demonstrations were dedicated to handling the profile object (mass $m = 0.70\,\text{kg}$) with solution *profile*. The clustering method finally mapped nine demonstrations to three solutions ($S = 3$).

In the following, it was evaluated that the robot was able to detect anomalies during execution and to act without requiring any prior knowledge or a symbolic task representation. The anomaly threshold was set to $\epsilon = 20$ based on preliminary experiments. The task was to manipulate the arriving objects (full box, empty box, profile) onto the target locations as shown in Fig. 7.13. When the robot executed solution *empty_box* and faced an empty box at the pick location, the task could be solved without anomalies. Next, the case was analyzed where the nominal solution was *empty_box* but the robot faced a full box at the pick location. Figure 7.14 shows the monitored error and the detected anomaly during lifting of the box, mainly caused by the deviations in the force domain.

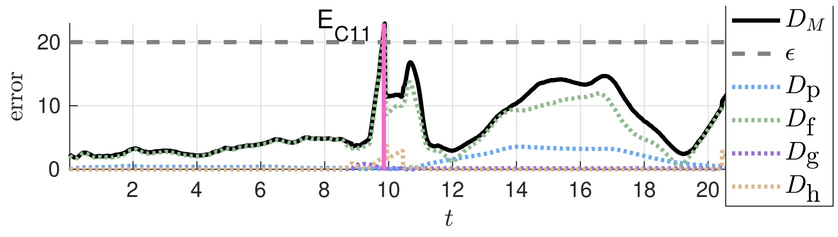


Figure 7.14: Error during execution with solution *empty_box* with a switch at event E_{C11} to solution *full_box*. A vertical purple bar marks the switch in both plots.

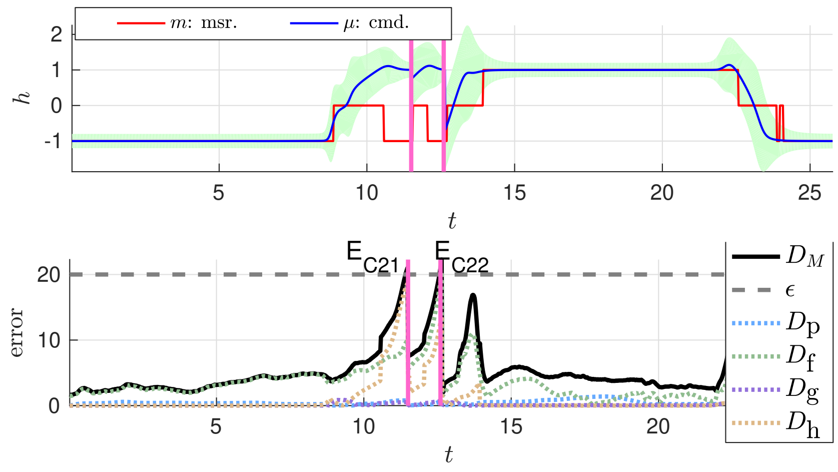


Figure 7.15: Top: Measured (m) and commanded (μ) grasp status h, when the robot tried to pick an unavailable object. In green, ±2 standard deviations around the commanded state. Bottom: Error during execution with solution *empty_box* with a switch at event E_{C21} to solution *full_box* and a 2^{nd} switch at event E_{C22} to solution *profile*. Vertical purple bars mark the switch in both plots.

This triggered a switching event E_{C11}, where the *scheduler* switches from *empty_box* to *full_box*.

In the next run, the robot starts again with the nominal solution *empty_box* but faces the profile at the pick location. Figure 7.15(top) shows the grasp status, indicating if objects were as intended. Figure 7.15(bottom) shows the Mahalanobis distance and how different modalities contributed to the overall error, especially influenced here by the grasp status (D_h) and force (D_f). The first anomaly when the object picking failed triggered a switching event E_{C21} from *empty_box* to *full_box*. It was identified as the next closest solution mainly because of the proximity of the pick location. The next picking attempt triggered the switching event E_{C22} and the start of the solution *profile*. Finally, an alternative solution was executed that manipulated the profile object successfully to the target position.

7.2.6 Conclusion

The proposed framework uses an anomaly detection strategy to trigger an alternative solution that best fits the current observed erroneous state. The alternative is parameterized such that it starts near the error sample in the nominal solution. Multiple alternatives can be provided via demonstrations, where the most appropriate one is selected by minimizing the error to the failed state. The framework employs PbD to allow end-users to teach conditional tasks and error recovery strategies without requiring any other prior knowledge or semantic information about the task.

The experiments show that the online solution switching works in different scenarios. However, there is no guarantee that the transition to the alternative will succeed. This limitation may depend on multiple factors, such as the initially selected solution, the demonstrator's performance, or how the objects in the environment are arranged. Finding or learning a general anomaly threshold on the state space, which is invariant of the task goals and does not require any parameterization is a challenge that should be considered in future work. This framework exploits the proprioceptive sensing capabilities of the robot, which might be fused with additional sensors in the future, such as visual perception, to allow an even more adaptive and reactive framework in unstructured environments.

7.3 Collaborative Programming with Task Graphs

Section 7.2 explained how a batch of demonstrations provided before task execution can be exploited to program adaptive robot behaviors for conditional tasks or recovery behaviors. It proposed a sequential programming paradigm that consisted of two phases. In the first phase, the user demonstrated all possible scenarios. In the second phase, the robot used the accumulated knowledge to spawn new behaviors during execution time.

This section introduces a new programming paradigm called collaborative programming, which sees the robot as a cooperative counterpart to the human, contributing with efforts to create a robot program. This paradigm constitutes a flexible and interactive teaching strategy, where the joint efforts of human and robot can add knowledge at any time.

7.3.1 Overview

The manufacturing domain moves from repetitive robot tasks to more adaptive, collaborative, and intelligent robotic applications. The two main goals of this section are to increase the adaptability and robustness of robot behaviors. In this context, collaboration between human and robot is the enabler to reach these goals utilizing a tree-like task structure, which is called task graph.

Adaptability is needed whenever the robot needs to change its behavior on the fly if the task requires it. It is seen as an inherent property of a robot's behavior. Therefore, two possible solutions are given. On the one hand, new environmental conditions could be foreseen by the programmer and implemented on demand. On the other hand, the system

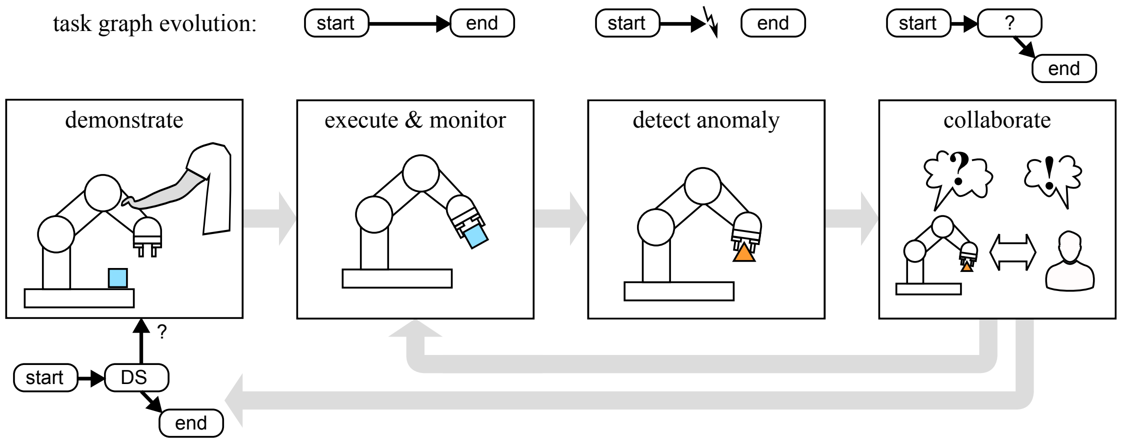


Figure 7.16: Collaborative PbD framework for programming of task decisions and recovery behaviors that uses decision states (DS) within a task graph. From left to right, a human demonstrates a task, the robot executes and monitors the current action, the robot detects a possible anomaly, human and robotic agent collaborate and decide how the new information shall be embedded into the task graph. The evolution of the task graph is shown on top.

could extend any existing robot behavior on the fly. Similarly, robustness is required to achieve the task goals with environmental variations or unforeseen environmental states. Per se, a system can only be seen as robust concerning a certain environment in which it has been tested. As soon as the environment changes, the robot's behavior might no longer be able to cope with all eventualities and require modifications to guarantee robustness. With that in mind, it becomes infeasible for the programmer to think about all eventualities and, therefore, requires an incremental strategy to add missing knowledge to the system. Incremental teaching, as proposed in this section, has the following requirements on the human, task, and system:

i) Availability of a human teacher in case the system requests input

ii) A structured, repetitive task with possibly unexpected conditions

iii) Physical access to the system allowing kinesthetic teaching

One way to achieve robustness can be through recovery behaviors, where the system knows how to resolve erroneous states. Another way is to increase the system's adaptability to the environment with task decisions that are made based on the environmental condition, enabling the robot to act in different ways. Ideally, end-users can intuitively transfer the required knowledge for robot programs that include task decisions or recovery behaviors. For instance, a robot could cope with a failed grasp by a re-grasping action that is automatically triggered during runtime. Another example is to let a robot make a decision based on a specific object property, for instance, sorting objects by their weight. Such problems that depend on observed conditions are also referred to as conditional tasks.

This section proposes a PbD task graph learning framework that allows the intuitive transfer of task knowledge that includes task decisions and recovery behaviors using a bidirectional communication channel between human and robot (see concept in Fig. 7.16).

7.3.2 Graph-based Incremental Programming

The proposed approach combines task graph programming with an autonomous fault detection scheme. The robot acts as an active learner that requests new user demonstrations in unknown regions of the input space. This enables the robot to decide ad-hoc when further information is required to extend the task graph with decision states and possible recovery behaviors.

7.3.3 Probabilistic Action Encoding

The user is requested to only demonstrate a new behavior once to add a new action. This demonstration is combined with a robotic repetition of the same. This process accounts for two problems. First, it incorporates some variance of the environment since the objects that are manipulated in both cases are set back to their initial state with possible uncertainty. Second, it accounts for the differences in dynamics and control behavior between human demonstration and robotic repetition. For instance, the impedance controller's stiffness parameters influence the robot's tracking performance and disturbance-induced control deviations.

The obtained two trajectory samples from human and robot are then used to encode the demonstrated action and determine the regions of variance around the nominal trajectory. Fig. 7.17a shows an example of these variance regions. Here, low variance regions lead to a more sensitive anomaly detection and vice versa. In parts with high variability, higher deviations are accepted during the execution, which increases the overall robustness. The whole system only makes use of the robot's proprioceptive sensing capabilities, which are the wrench measured with a FTS at the robot wrist, the Cartesian pose, the distance between the gripper fingers, and the grasp status indicating if an object is grasped or not, evaluated by the grasp force. Visual perception is not part of the presented approach. Instead, force measurements are exploited, which help to make a system independent of object visibility or lighting conditions.

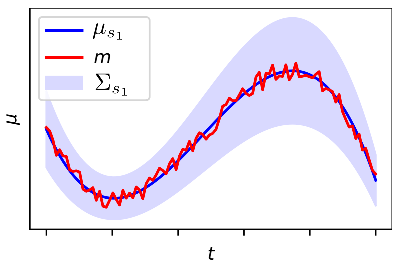
A data sample at time t is given as $\boldsymbol{X}^{(t)}$ as specified in (3.3) with $D = 15$ data dimensions. It contains the robot pose, wrench, and gripper states. The data is recorded at a frequency of 1 kH and is down-sampled to 50 Hz in order to reduce the computational effort in the data processing. The recorded data from the user demonstration

$$\boldsymbol{X}_{\text{Udem}} = \left[\boldsymbol{X}_{\text{U}}^{(1)}; \ldots; \boldsymbol{X}_{\text{U}}^{(N_{\text{U}})}\right] \in \mathbb{R}^{N_{\text{U}} \times 15}$$

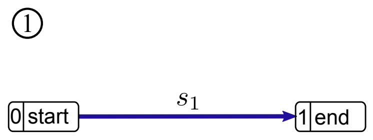
and robot repetition

$$\boldsymbol{X}_{\text{Rrep}} = [\boldsymbol{X}_{\text{R}}^{(1)}; \ldots; \boldsymbol{X}_{\text{R}}^{(N_{\text{R}})}] \in \mathbb{R}^{N_{\text{R}} \times 15}$$

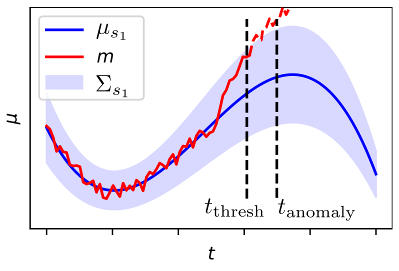
with respective sample length N_{U} and N_{R} are collected for each new demonstration. Similar as in Sec. 7.2.2, the data is standardized dimension-wise with the z-transformation



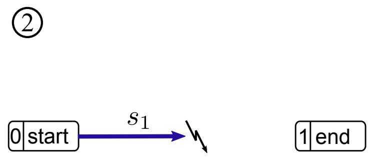
(a) Normal execution.



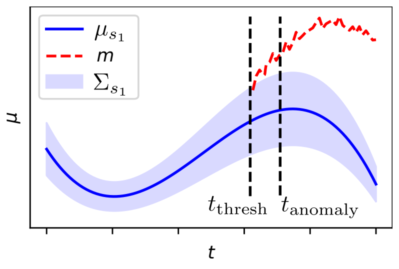
(b) Initial graph after first action encoding.



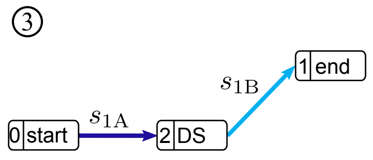
(c) Anomaly detection.



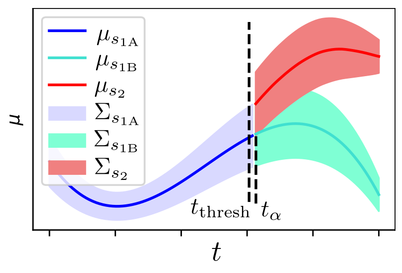
(d) Robot stops execution at abnormal state.



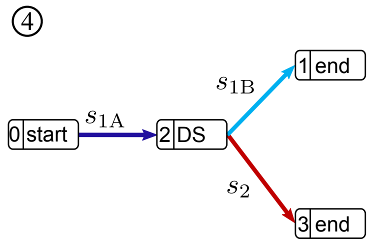
(e) Demonstration of new action.



(f) Insertion of a decision state (DS) by splitting the current action.



(g) Splitting up action s_1 at t_α.



(h) Insertion of an alternative action starting at the decision state.

Figure 7.17: Stages of creating a task graph by monitoring the execution and by reacting to anomalies.

by subtracting the mean and dividing by the standard deviation. This step assures that each dimension contributes equally to the DTW error in the following step. DTW is then used to align the two sensor sequences on a joint time vector with equalized length N. After warping the data, the standardization is undone by applying the inverse z-transformation dimension-wise.

In the next step, Expectation Maximization (EM) approximates a multivariate, time-based Gaussian Mixture model (GMM) for the input matrix

$$\boldsymbol{G}_s = \begin{bmatrix} \boldsymbol{X}_\mathrm{U} & \boldsymbol{n} \\ \boldsymbol{X}_\mathrm{R} & \boldsymbol{n} \end{bmatrix} \in \mathbb{R}^{(2N)\times(D+1)} \tag{7.8}$$

with a time vector $\boldsymbol{n} = [1, \dots, N]$.

The variables $\boldsymbol{X}_\mathrm{U}$ and $\boldsymbol{X}_\mathrm{R}$ serve as placeholders for demonstrated and repeated time series, respectively, where further use-cases are considered in Sec. 7.3.4. The model complexity is chosen such that the number of model components k is proportional to the length N of the demonstrated time series data. The EM algorithm is initialized by k-means clustering with a number of k clusters. Applying EM results then in the model $\mathcal{M} = GMM(\boldsymbol{G}_s)$. GMR is applied to reproduce a generalized trajectory

$$\boldsymbol{Y}_s = [\boldsymbol{Y}^{(1)}; \dots; \boldsymbol{Y}^{(N)}] \in \mathbb{R}^{N\times D}$$

with an associated sequence of covariance matrices

$$\boldsymbol{Z}_s = [\boldsymbol{Z}_s^{(1)}; \dots; \boldsymbol{Z}_s^{(N)}] \in \mathbb{R}^{N\times D\times D}.$$

A Cartesian impedance controller can now track the generalized trajectory. The desired behavior is monitored by computing an error between measurements and generalized trajectory while considering the sequence of covariance matrices as a confidence boundary.

7.3.4 Online Anomaly Detection

The system design for online anomaly detection is shown in Fig. 7.3. Its main purpose is to monitor the execution and detect new situations unknown to the system. A number of sensor modalities are introduced that allow to distinguish the source of error. These modalities are

1. the robot pose $(\boldsymbol{p}, \boldsymbol{o})$,
2. the wrench $(\boldsymbol{f}, \boldsymbol{\varrho})$, and
3. the gripper finger distance g and grasp status h.

The symbol m is a placeholder for the above modalities. The system constantly compares the commanded and measured variables for each of these modalities to detect abnormal states. This mechanism detects new situations and determines a possible error source, e.g., an abnormal state caused by unexpected external forces. In each time step t of the execution, the deviation between the measurement $\boldsymbol{M}_m^{(t)}$ and commanded state $\boldsymbol{Y}_m^{(t)}$ of a modality m is quantified using the Mahalanobis distance as defined in eq. (7.2). The modality specific Mahalanobis distance is denoted as $D_m^{(t)}$.

By defining a custom anomaly threshold $\epsilon_m i$ for each modality m of an action s, this metric leads to a higher error sensitivity in time steps where the execution needs to be precise, indicated by small values of the reduced covariance matrix $\boldsymbol{Z}_m^{(t)}$. During the execution, all modalities are monitored simultaneously. If any $D_m^{(t)}$ exceeds its action and modality-specific anomaly threshold ϵ_m for e consecutive time steps, an anomaly is detected.

The approach does not rely on manual error threshold tuning but is automatically parameterized by the training data. The anomaly threshold ϵ_m is computed for each modality m of an action, based on the recorded trials of the user demonstration U and robot repetition R. Let $\boldsymbol{M}_{m,d}^{(t)}$ be the sample of time series $d \in \{\mathrm{U}, \mathrm{R}\}$ belonging to modality m at time step t. It corresponds to the mean sample $\boldsymbol{Y}_m^{(t)}$ of the GMR trajectory. After encoding a new demonstration in a GMM and computing the GMR, the maximum Mahalanobis distance is extracted by

$$\tilde{D}_{m,d} = \max_{t \in [1,N]} \sqrt{(\boldsymbol{M}_{m,d}^{(t)} - \boldsymbol{Y}_m^{(t)})(\boldsymbol{Z}_m^{(t)})^{-1}(\boldsymbol{M}_{m,d}^{(t)} - \boldsymbol{Y}_m^{(t)})^T} \,. \tag{7.9}$$

Then, the maximum distance over all trials is defined as

$$\epsilon_m = \max_{d \in \{\mathrm{U},\mathrm{R}\}} \tilde{D}_{m,d} \tag{7.10}$$

and used as a modality-specific error threshold.

Collaborative and Incremental Graph Construction A task graph structures the available robotic actions and possible decision states on an abstract level (as shown in Fig. 7.17h). Such a graph is incrementally built by accumulating task knowledge from several user demonstrations. The graph's nodes represent system states that can be of type *start*, *end*, and *decision state (DS)*.

In order to construct a new task, a user enters a demonstration phase and provides an initial task demonstration. The system extracts a robotic action from this demonstration as explained in Sec. 7.3.3 (Fig. 7.17a). Next, a start and end state is added to the beginning and end of this action. The result is shown in Fig 7.17b, which already allows the execution of that simple task.

If an anomaly is detected during execution, as explained in Sec. 7.3.4, the robot stops at the unseen state (Fig. 7.17c and Fig. 7.17d). Next, the system requests the user to choose from the following options:

1. The detected situation shall be handled by a new dedicated action in future executions (*Graph Extension*)
2. It shall be incorporated as a refinement for the current action (*Action Refinement*).

Graph Extension If the user selects to add a new action to resolve the current situation, the system switches to a demonstration phase and waits for the user's input. The robot configuration is still in an abnormal state and can now be changed by the user via kinesthetic teaching. It is assumed that an anomaly has been detected beforehand at time step t_{anomaly}. In the following, a user demonstration $\boldsymbol{X}_{\text{Udem}}$ is recorded. This data is appended to the time series $\boldsymbol{M}$ recorded during the interval $[t_\alpha; t_{\text{anomaly}}]$, resulting in $\tilde{\boldsymbol{X}}_{\text{Udem}} = [\boldsymbol{M}, \boldsymbol{X}_{\text{Udem}}]$. After the demonstration, the system requests the user to restore the environment to the state before the demonstration. Here, the manipulated object locations are set back to their initial state. Next, the robot moves to the configuration at time step t_α and repeats the extended user demonstration $\tilde{\boldsymbol{X}}_{\text{Udem}}$. Two resulting time series from the user and robot are then probabilistically encoded and saved as action s_2.

Finally, a new decision state is inserted into the graph, splitting up action s_1 into two actions before and after the anomaly, depicted $s_{1\text{A}}$ and $s_{1\text{B}}$ respectively (see Fig. 7.17g and Fig. 7.17h). The actions $s_{1\text{B}}$ and s_2 are then appended to the newly inserted decision state. In detail, action s_1 is split at time step

$$t_\alpha = t_{\text{thresh}} + \alpha e \ , \tag{7.11}$$

where t_{thresh} is the time step in which the error metric $D_m^{(t)}$ first exceeds the anomaly threshold ϵ_m. The parameter e is the number of consecutive time steps for which $D_m^{(t)} > \epsilon_m$ until an anomaly is triggered. The scaling factor α $(0 < \alpha < 1)$ places the decision state in between time step t_{thresh} and t_{anomaly}.

An early and smooth transition from action $s_{1\text{A}}$ to its successor, without pursuing a possibly erroneous strategy for too long, requires a minimum α. This means that the decision state should be close to time step t_{thresh}. However, a robust decision requires a sufficiently long sequence of unique sensor readings associated with a particular action, shifting the decision state towards t_{anomaly} and thus $\alpha \to 1$. Moreover, the decision for the subsequent action must be made before t_{anomaly} is reached when the action $s_{1\text{A}}$ is executed. Otherwise, the anomaly detection would incorrectly identify a new situation for the scenario handled by s_2 (see Fig. 7.17h). Preliminary experiments have shown that the number of error samples $e = 30$ (corresponding to $30/50\,\text{Hz} = 0.6\,\text{s}$) and the scaling factor $\alpha = 1/3$ provide a good trade-off between robustness and delay in decision making.

Action Refinement If the user chooses to refine the action s where the anomaly is detected, its encoded trajectory $\boldsymbol{Y}_s$ and associated sequence of covariance matrices $\boldsymbol{Z}_s$ is adjusted by new data. This refinement option enables an existing action to handle various conditions so that the robot learns which features are essential and which regions of the state space do not require strict monitoring and error handling. For example, a sorting task for geometrically different objects that ignores object weight can be achieved by refining the actions that handle the different geometries. In this case, the refinement results in actions where monitoring no longer depends on object weights, thus avoiding false positive detection of force anomalies in future task executions. One such example is evaluated later in the experiments section. A trial that provides the time series acting in the new environment is performed in one of the following ways:

1. by a user demonstration in the so-called *user-refine* mode
2. by the robot in the so-called *auto-refine* mode.

In the *user-refine* mode, a manual demonstration allows adjusting the whole trajectory of the correction, which starts directly where the anomaly has been detected. In comparison, in *auto-refine* mode, the robot can autonomously continue execution after the anomaly is detected until the end of the action. Since it is already known that a new situation shall be included in the action encoding, anomaly detection is disabled for the rest of the execution. For both possible modes, the recorded time series $\boldsymbol{X}_{\text{Rref}}$ is appended to the time series $\boldsymbol{M}$ of the action before the anomaly, resulting in a stacked matrix $\tilde{\boldsymbol{X}}_{\text{Rref}} = [\boldsymbol{M}, \boldsymbol{X}_{\text{Rref}}]$. This data, along with the initial user demonstration $\boldsymbol{X}_{\text{Udem}}$ and robot repetition $\boldsymbol{X}_{\text{Rrep}}$ of this action, is used for new probabilistic coding as described in section 7.3.3. Finally, the task graph is updated to incorporate the new action model.

7.3.5 Task Execution

An efficient switching mechanism between robot execution and human teaching governs the production phase. Switching occurs ad-hoc, just if required by environmental changes. Otherwise, the system stays in the task execution phase to operate under nominal conditions. After an initial task demonstration, an execution phase can immediately follow to start production. Adding knowledge to the task graph at any time is seamlessly possible. The user can intentionally add knowledge for known situations from the beginning, or the system requests a user interaction at any time, e.g., after unforeseen errors occur during production. The task graph allows the robot to reproduce any demonstrated task and to adapt to environmental conditions by exploiting known decision states. It enables a fundamental extension compared to simple sequential task execution, namely selecting the appropriate action based on current sensor readings. Conditional tasks allow, for example, sorting by object properties or selecting recovery actions for faulty states.

The task graph structures the available actions at a high level. The actions are encoded probabilistically at a low level, enabling real-time monitoring by exploiting the model's uncertainty. Decision states are automatically inserted at critical state transitions of the task, which simplifies the decision process for a particular state but also eliminates perceptual aliasing and thus the risk of making a wrong decision for an action. A decision state also helps to make decisions as early as possible and avoids unnecessary delays or robot movements. Since the decision is enforced in the decision state, the system can react faster than approaches that use only sequential knowledge representations.

The presented approach also identifies the sensor modality that has a major impact on causing the anomaly. This is achieved by only considering the relevant sensor values to select the subsequent action in a decision state. An example is used in the following to explain the action selection in a decision state, as shown in Fig. 7.17h. The robot starts with the first action $s_{1\mathrm{A}}$. In case no anomaly is detected during the execution, the robot reaches the first decision state (DS). From here, the subsequent action $\hat{s}$ is determined by

$$\hat{s} = \underset{s}{\operatorname{argmin}} \left(\| \boldsymbol{m}_{\mathrm{DS},m} - \boldsymbol{Y}^{(0)}_{s,m} \| \right) , \tag{7.12}$$

where $\boldsymbol{m}_{\mathrm{DS},m}$ is the measurement in the modality m at the decision state and $\boldsymbol{Y}^{(0)}_{s,m}$ is the first sample in the modality m of an encoded action s. In this example, the next executed action $\hat{s}$ is selected from the set $\{s_{1\mathrm{B}}, s_2\}$. It contains all actions that are attached to the decision state. In contrast to the anomaly detection method, an Euclidean distance metric is used here. The reason is that the Mahalanobis distance favors actions with high uncertainty, expressed by large values in the covariance matrix that lead to small errors in the first time step. With the proposed scheme, the robot always chooses an action that minimizes the error in the current environmental state and monitors that action to detect possible future anomalies.

7.3.6 Experiments

The experiments evaluate a scenario where a user transferred a sorting task to a robot by incrementally adding knowledge. The system queried only three user demonstrations while providing instructions via a GUI. When the robot detected an anomaly during task execution, the user could either demonstrate a new action that accounts for this situation or refine the current action by incorporating the new conditions into it. This experiment demonstrates both the action refinement and the task graph extension capabilities of the presented framework, which allows the robot to ignore irrelevant and learn relevant features of a task.

Experimental Setup As shown in Fig. 7.18, a DLR LWR IV was mounted on a linear axis and equipped with a Robotiq 85 two-finger gripper as well as a FTS measuring the forces and torques acting on the end-effector. The robot was impedance controlled with a control rate of 1 kHz and parameterized with constant linear stiffness $k_{\mathrm{trans}} = 1200$ N/m, angular stiffness $k_{\mathrm{rot}} = 100$ Nm/rad, and damping coefficients $d_{\mathrm{m}} = 0.3$ Ns/m respectively. User buttons and a tablet displaying a GUI allowed the user to operate the robot and receive visual feedback. The buttons were used to open and close the gripper and to start and stop the demonstration recording. Kinesthetic teaching was used to provide the demonstration data. The GUI guided the user through the teaching process and requested input whenever needed. A conveyor belt placed perpendicular to the table transported boxes with supplies for an assembly task to a determined place in the working space.

The goal of the task was to program the robot to distinguish different boxes based only on their geometry to place the supplies at a specific spot in the part storage. This experiment focuses on sorting by the object's geometry while sorting by an object's weight was validated in Sec. 7.2.5. Here, the weight of the boxes was intentionally ignored when deciding on the final position of a box. More specifically, it is assumed that boxes of equal dimensions contain the same type of pieces but do not necessarily

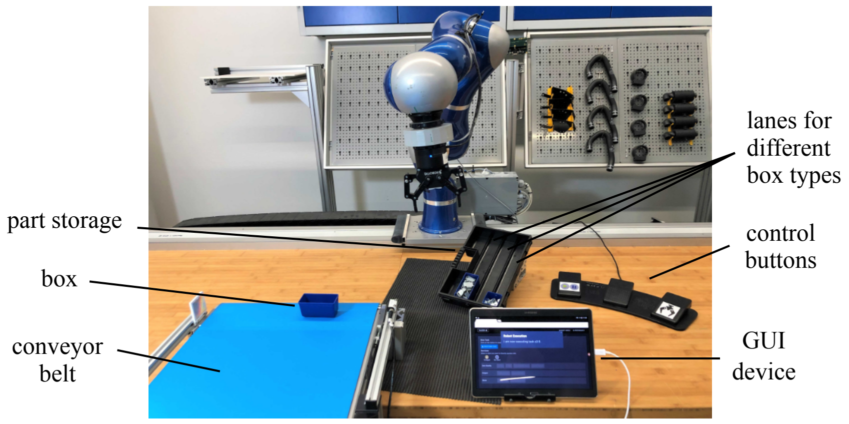


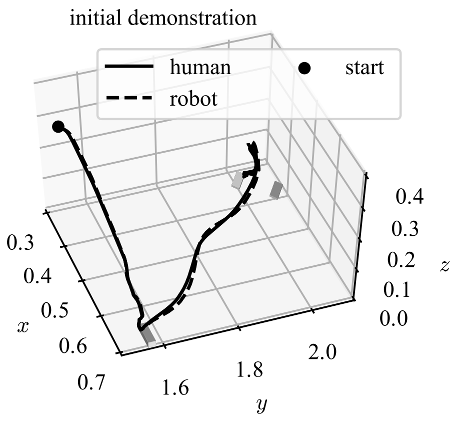
Figure 7.18: Experimental setup where a conveyor belt delivers different boxes to a pick location, from where they can be sorted with respect to different properties.

contain the same amount of pieces, hence differing in their weight. The experiment is structured in the following phases, considering three different conditions in the task's environment.

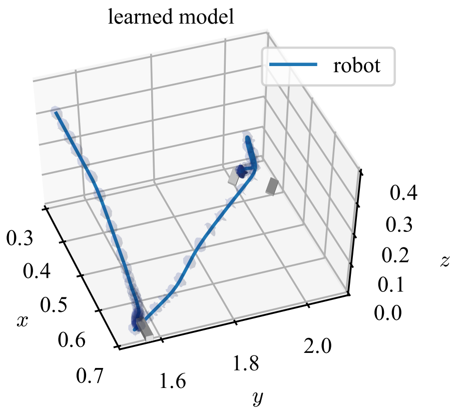
Nominal Condition First, the user provided an initial demonstration by guiding the robot to pick up a box with supplies from the start position on the conveyor belt and to place it on its designated spot in the part storage (Fig. 7.19d). After the demonstration, the box was set back to the start position on the conveyor belt so that the robot could repeat the demonstrated sequence. The resulting two time series of human and robot are shown in Fig. 7.19a. They were used to learn a model of the action represented in Fig. 7.19a with the Gaussian mixture components shaded in gray. Then, the robot executed the learned model in the same environment, where both commanded and measured states behave similarly (Fig. 7.19c).

Differing Weight Condition The robot manipulated a box of the same type but with a different weight, which caused an anomaly caused by an unexpected force f_z in z-direction (Fig. 7.20a). A weight deviation of a box was not considered to be an important feature of the task. Therefore, the user decided to refine the current action with the *auto-refine* mode (Sec. 7.3.4) in order to incorporate the new condition into the action's model. The refinement condition is shown in Fig. 7.20d. In *auto-refine* mode, the robot continued the learned motion and placed the box in its designated spot in the part storage.

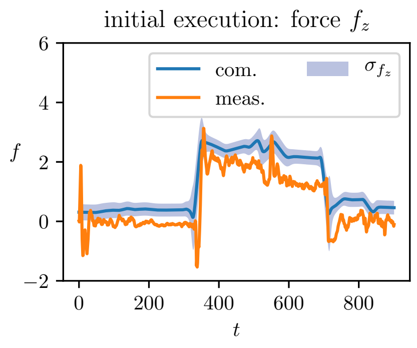
Differing Shape Condition Here, the robot was confronted with a box of a different shape during grasping, resulting in an anomaly detected by the gripper finger distance (Fig. 7.21a). The user decided to demonstrate a new action, where the box with differing shape was placed on another spot in the part storage (Fig. 7.21d). The user demonstra-



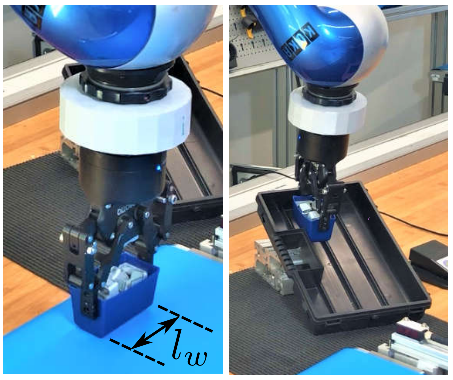
(a) Human demonstration and robot repetition for the initial task.



(b) Encoded action s_1 from the data shown in a).



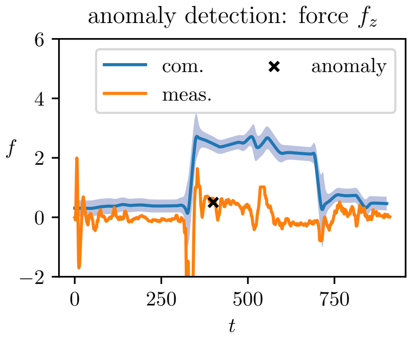
(c) Initial execution of s_1 with commanded (com.) and measured (meas.) time series.



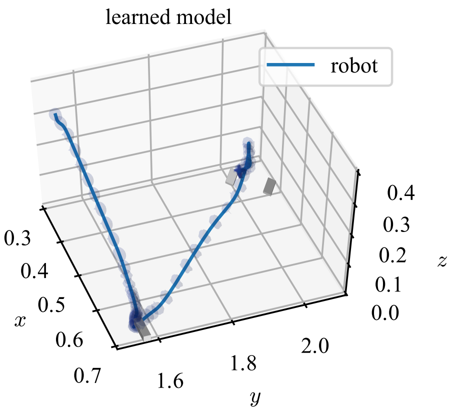
(d) Left: picking the wide box with edge length l_w, right: placing the wide box in the tray.

tion was repeated by the robot with the same box type and encoded in the model of action s_2, shown in Fig. 7.21b.

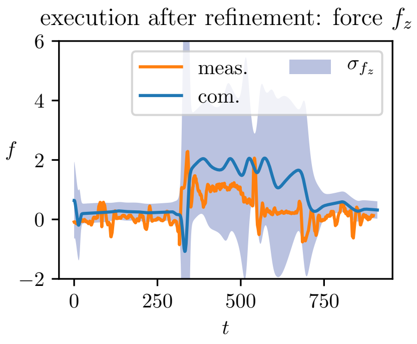
Results The approach has been evaluated to show robustness in the case of differing external forces (box weight) and adaptability in the case of different object shapes (box geometry). It learned that the critical feature to consider in the online decision was the object shape and that the object weight is irrelevant to the task. As shown in Fig. 7.20c, the task structure handles the lighter box the same as the heavier box by considering the value of f_z with a larger variance σ_{f_z} and increased force anomaly threshold ϵ_f, derived by (7.9) and (7.10). As introduced in Sec. 7.3.4, this leads to a less sensitive force anomaly detection in future executions of this action and allows to manipulate boxes with a variety of weights without triggering a false positive anomaly detection. At the same time, the robot can still learn additional actions for unforeseen situations by requesting user demonstrations.



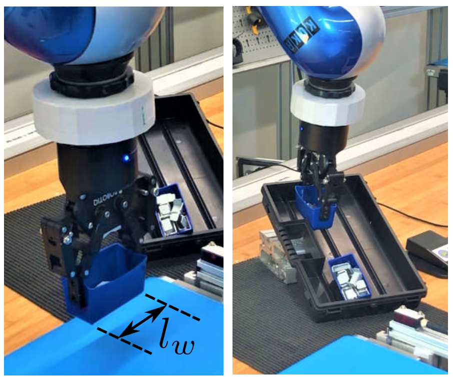
(a) Execution with anomaly detection in the force domain for action s_1 while picking a lighter box as before. The measured (meas.) time series is used to update the model of this action.



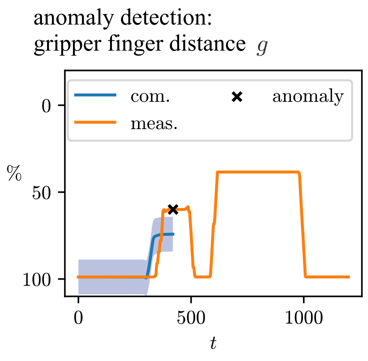
(b) Refined model of action s_1 considering the measured data shown in e).



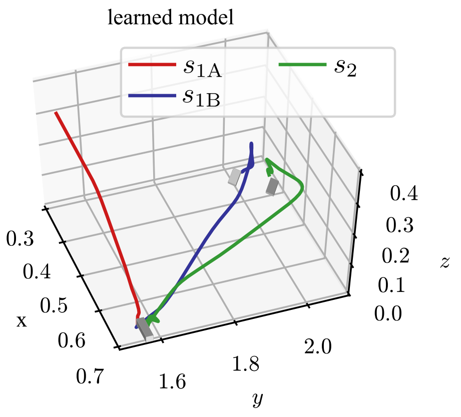
(c) Execution of s_1 with an increased error margin related to σ_{f_z}.



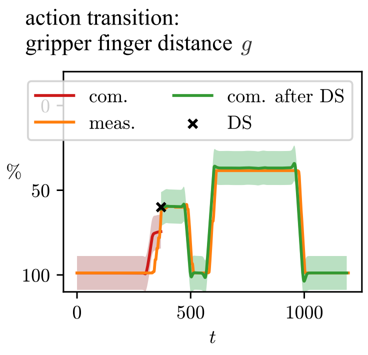
(d) Left: picking the lighter wide box with edge length l_w, right: placing the lighter wide box at the same location.



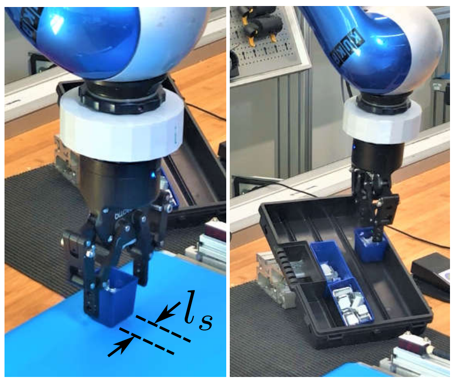
(a) Execution with anomaly detection in the gripper finger distance.



(b) Learned model after splitting action s_1 into s_{1A} and s_{1B} and added action s_2.



(c) Execution exploiting the decision state to switch from action s_{1A} to the new action s_2 that resolves the anomaly.



(d) Left: picking the small box with edge length l_s, right: placing the small box on a new location specified by the user.

Figure 7.21: Experimental results of the box sorting task with action refinement and graph extension.

Fig. 7.21a and 7.21d shows the detected anomaly of the gripper finger distance when grasping a smaller box. This event allowed the user to demonstrate a new action that places the box on another goal location. When grasping the box, a new decision state was inserted into the task graph (Fig. 7.21b). Here, the robot decided on the subsequently executed action based on the measured gripper finger distance (see Fig. 7.21c).

7.4 User Study for Approach Comparison

This section evaluates and compares the intuitiveness of CIP, SBP, and UIP by means of a user study. UIP is introduced in the related work (Sec. 2.4.4). Furthermore, the task structures generated with these frameworks are compared by their ability to reach the task goals.

7.4.1 Materials and Study Design

Sample Within this study, 21 participants (19 male and 2 female) were recruited from the German Aerospace Center (Age = 25.24 ± 7.03 years, ranging from 21 to 56). All participants have a background in different technical fields but not necessarily in robotics.

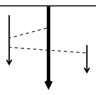
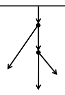
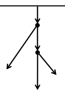
Setup Sec. 7.1.5 describes the used demonstration system. Throughout the study, all tasks involved the same object, which is an aluminum block (dimensions: 6.8 cm x 4 cm x 2 cm) that was placed in different locations (see Fig.7.22).

Procedure Each participant was informed about the study's aim and procedure. In the introduction phase, the robot's sensing capabilities were explained, highlighting that no vision-based monitoring of the environment was used. After a maximum of five minutes to familiarize with the robot and control buttons, the experimental tasks were explained. Each participant watched a short instruction video that explained each method (CIP, SBP, UIP) individually. Then, the participant programmed both tasks for all three methods. The order of teaching each task with each method was permuted among all subjects by a Latin square design [192].

After completing the programming with one of the methods, the NASA-TLX [210] and the *Questionnaire for Measuring the Subjective Consequences of Intuitive Use* (QUESI) [211] were filled out by the participant. At the end of the experiment, an overall evaluation of the methods took place, where the participant rated intuitiveness and efficiency on a 7-point Likert-type scale followed by a semi-structured interview.

Data Analysis Successful task completions were nominally scaled and analyzed using Cochran's Q test and McNemar post-hoc tests in case significant differences between methods were found. For questionnaire items, a repeated measures ANOVA was calculated. In case of violation of sphericity (Mauchly's sphericity test), Huynh-Feldt ($> .75$) or Greenhouse-Geisser ($< .75$) corrections were made. Post-hoc tests with Bonferroni correction were performed to identify which methods differed significantly.

Table 7.1: Overview of Compared Frameworks

Methods / Properties	proposed: Sequential Batch Programming (SBP)	proposed: Collaborative Incremental Programming (CIP)	baseline: User-triggered Incremental Programming (UIP)
task representation			
teaching-interaction	unidirectional	bidirectional	unidirectional
incrementally extendable	✘	✔	✔
online decision making	✔	✔	✔
collaborative programming	✘	✔	✘

7.4.2 Compared Methods

Table 7.1 provides an overview of the PbD approaches that were compared in the user study, which all used the same sensory input without visual perception. The approaches are summarized in the following:

SBP (Sec. 7.2) uses separated teaching and execution phases. First, a teacher sequentially demonstrates all task solutions that the robot shall account for and stores these solutions independently in a SP. If an anomaly occurs during task execution, the system switches to the state within an alternative solution that minimizes the error between the current measurement and all alternative solution states.

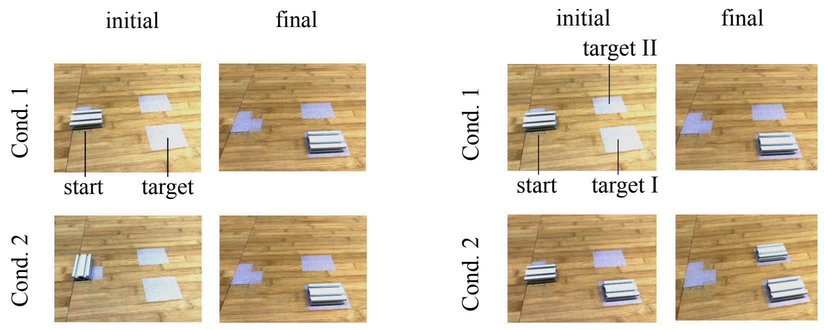
CIP Sec. 7.3 combines anomaly detection with collaborative programming to account for new task conditions. Collaborative programming relies on intertwined teaching and execution phases. Compared to SBP, the decision state is explicitly programmed by collaboration between the user and robotic agent.

UIP (as introduced in Sec. 2.4.4) is inspired by the framework presented in [134], where similar to CIP, a task graph is incrementally constructed in a combined teaching and execution phase. The difference is that the teacher has to detect anomalies with UIP during the execution of the task and needs to trigger the exact time step when the robot shall make a task decision.

7.4.3 Hypothesis

This study had the aim to verify the following hypotheses:

- $\mathcal{H}\mathbf{1}$ (based on objective metrics): Using CIP with its collaborative programming concept and autonomous anomaly detection results in a significant increase in successful task completions,
    - compared to SBP (hypothesis $\mathcal{H}\mathbf{1.1}$), and



(a) Task 1 **(b)** Task 2

Figure 7.22: Initial and final setups of each task with each two different environmental conditions (Cond. 1 and Cond. 2).

  - compared to UIP (hypothesis $\boldsymbol{\mathcal{H}}$**1.2**).

- $\boldsymbol{\mathcal{H}}$**2** (based on subjective ratings): A significant increase in programming intuitiveness is achieved by CIP with its collaborative programming scheme,
  - compared to SBP, which uses a training phase to collect all demonstrations in the beginning (hypothesis $\boldsymbol{\mathcal{H}}$**2.1**), and
  - compared to UIP, which requires the user to trigger the insertion of decision states manually (hypothesis $\boldsymbol{\mathcal{H}}$**2.2**).
- $\boldsymbol{\mathcal{H}}$**3** (based on subjective ratings): A significant decrease in workload is achieved by CIP,
  - compared to SBP (hypothesis $\boldsymbol{\mathcal{H}}$**3.1**), and
  - compared to UIP (hypothesis $\boldsymbol{\mathcal{H}}$**3.2**).

7.4.4 Experimental Tasks

The user study evaluated all methods in two experimental tasks, namely task 1: *Reorientation* and task 2: *Contact-based Sorting*. Fig. 7.22 shows their initial and final environmental states. The goal of task 1 was to manipulate an object from a start location to a target location with the following constraints: 1) At the start location, the object might be aligned either with its short or with its long edge with a mark on the table, i.e., it may be rotated by $\pm 90°$ in the table plane; 2) at the target location, the object's long edge must be aligned with a mark on the table. Given constraints 1 and 2 and depending on the environmental state, the robot might be required to reorientate the object before placing it in the target location. Schematic examples of how the task can be performed are given in Fig. 7.23 for SBP and in Fig. 7.24 for CIP and UIP. The goal of task 2 was to fill target locations in a sequential order, starting with target I. The object had to be placed on target II if target I was occupied. The manipulation steps and the generation of the task graph are shown in Fig. 7.27.

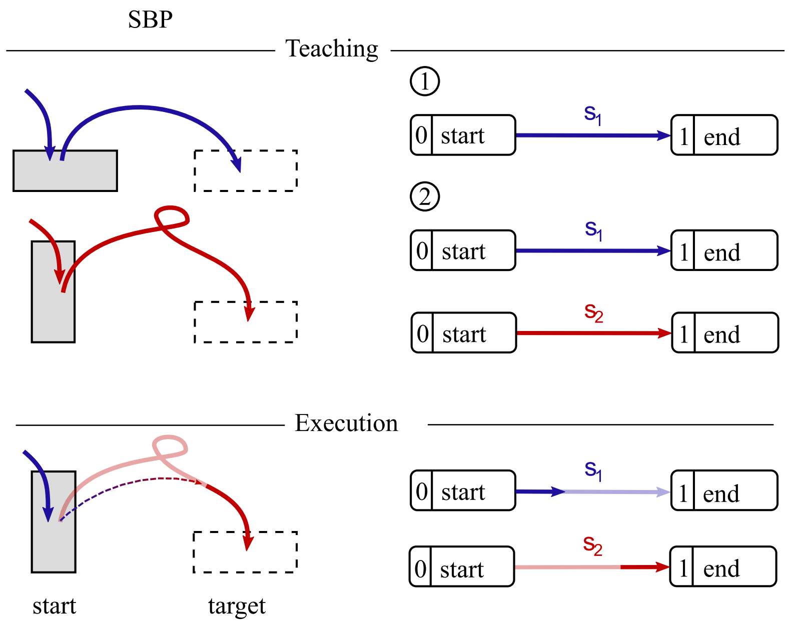


Figure 7.23: Task 1: Reorientation, SBP: In step (1), the user demonstrated a pick and place action s_1. In step (2), the user extended the SP with a second action s_2, in which the object was rotated by 90° before being placed in the target location. During the execution of the nominal solution s_1, the rotated object in the start location caused an anomaly that triggered a transition to the alternative solution. The bottom row illustrates an example of a failed execution, where the robot decided on a wrong entry point of the alternative and skipped the reorientation part of s_2.

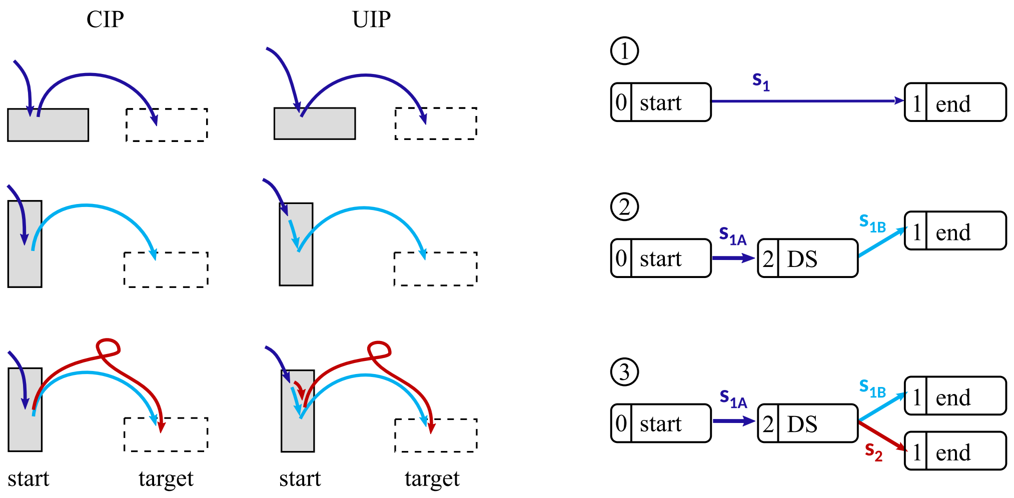
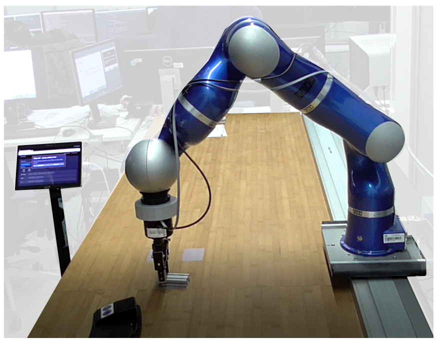


Figure 7.24: Task 1: Reorientation, CIP and UIP: In step (1), the user demonstrated a pick and place action s_1. Step (2) shows the updated graph after the first execution, where an anomaly led to the insertion of a decision state (DS) and split s_1 into s_{1A} and s_{1B}. The DS was created by the anomaly detection algorithm in CIP and by the user manually in UIP. In step (3), the user added a new action s_2 that accounted for the anomaly and properly rotated the object before placing it.


(a) **CIP**: Autonomous anomaly detection **when** grasping object.

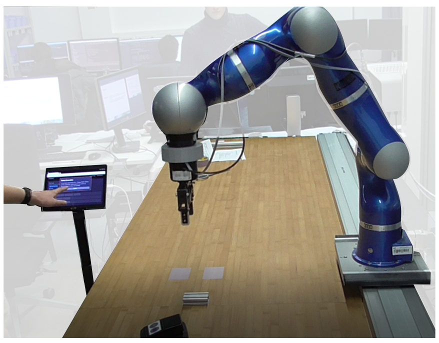

(b) **UIP**: Example for an incorrect user-triggered anomaly **before** grasping object.

Figure 7.25: Correct (a) and wrong (b) robot configuration to provide an alternative action for solving a new situation. Due to the user's influence on the anomaly detection, a configuration in which the robot can't sense the anomaly is more likely with UIP.

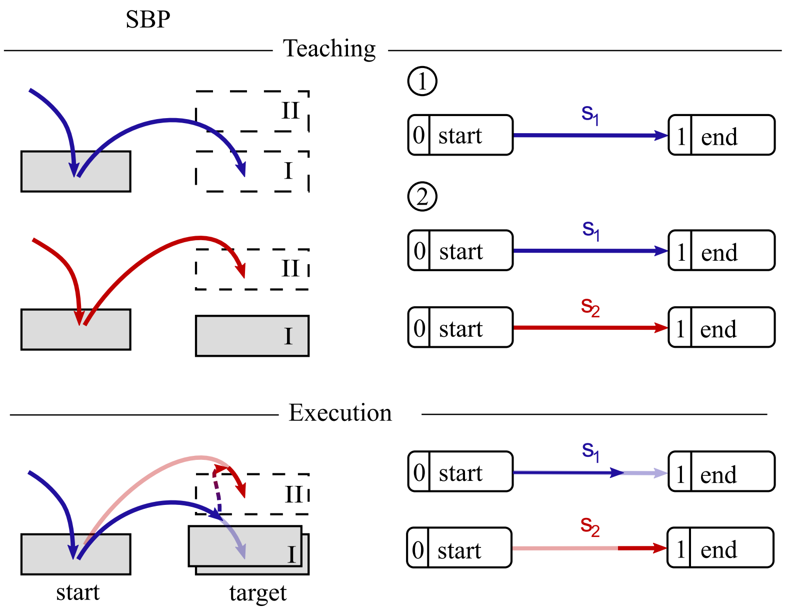


Figure 7.26: Task 2: Contact-based Sorting, SBP: The user successively demonstrated two pick and place actions in step (1) and (2). In demonstration of action s_2, the object is placed in target location II, if target location I is occupied by another object. The bottom row shows the execution of the nominal solution s_1, where an unexpected contact force triggered a transition to s_2 while approaching target location I. The robot interpolated to the entry state of the alternative solution and placed the object in location II.

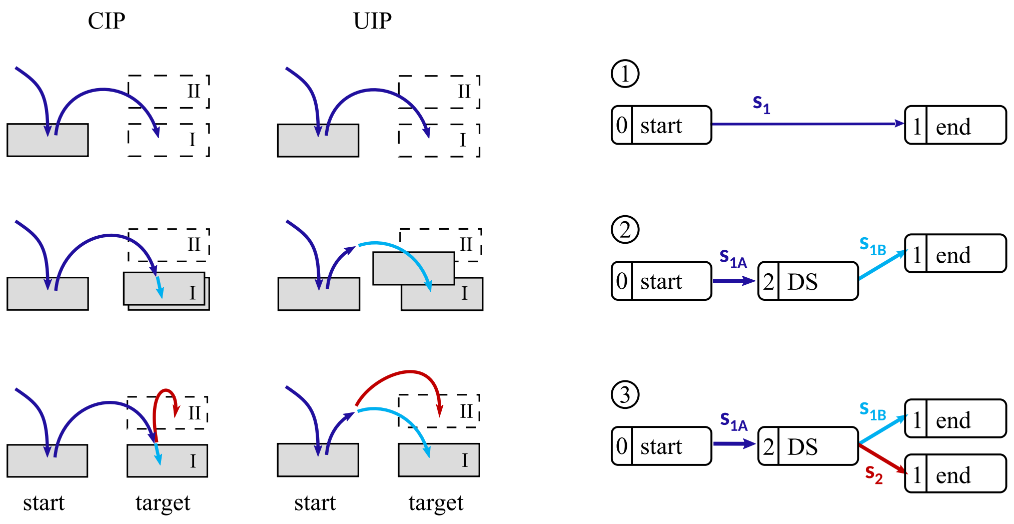


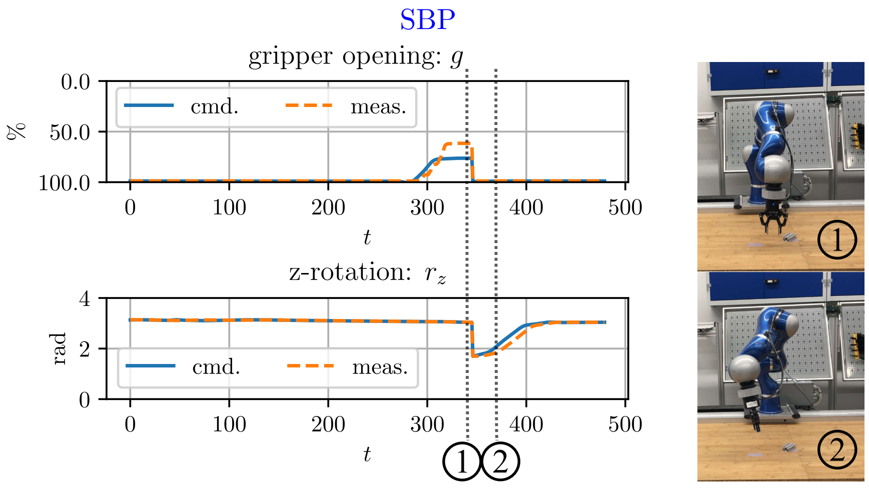
Figure 7.27: Task 2: Contact-based Sorting, CIP and UIP: Step (1) shows the initial demonstration of a pick and place action s_1. Step (2) shows the updated graph after the first execution where an anomaly led to the insertion of a decision state (DS) and splitting of s_1 into s_{1A} and s_{1B}. The DS was created by the anomaly detection algorithm in CIP and by the user manually in UIP. In step (3), the user added a new action s_2 that recovers from the anomaly.

7.4.5 Objective Results

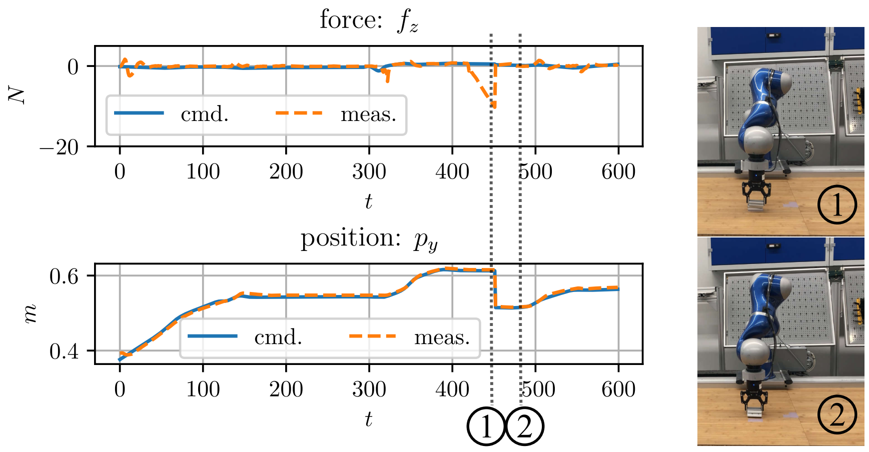
The compared methods were evaluated using objective performance data and subjective user feedback from the post-experimental questionnaires and the interview. Furthermore, the experimental hypotheses are evaluated accordingly. The objective data analysis examines successful completions and the decision state insertion.

Successful Completions The experimenter observed each task execution to determine if it reached its task goals successfully, as described in the experimental task description. This led to the success rate of executions for each method shown in Fig. 7.31. Cochran's Q test indicated significant differences between the conditions for task 1 ($p < .001$) and task 2 ($p < .001$). McNemar post-hoc tests revealed significant differences between SBP and CIP ($p < .05$) and CIP and UIP ($p < .001$) for task 1. For task 2, significant differences could be found for SBP versus UIP ($p < .001$) as well as CIP versus UIP ($p < .001$).

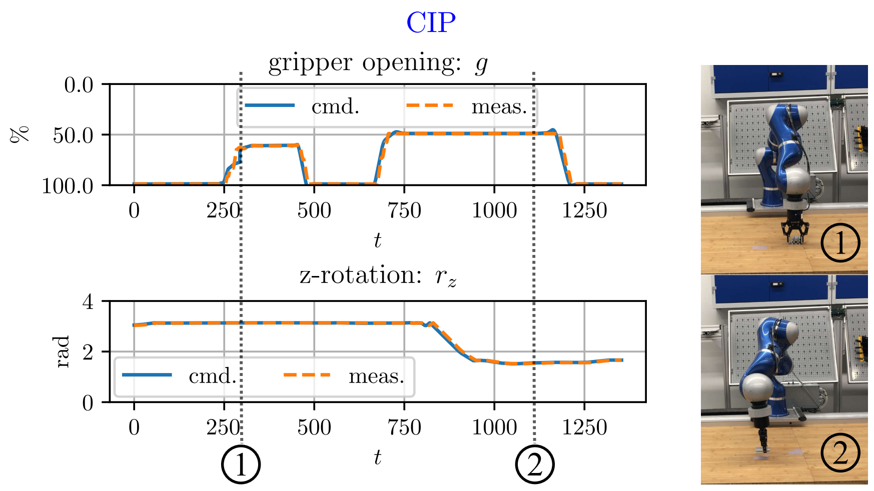
$\mathcal{H}$**1.1** does not hold for task 1 (✘) but is true for task 2 (✔), meaning that there are significantly more successful task completions by using the collaborative programming scheme of CIP compared to the collection of demonstrations in a batch as used in SBP. This effect could be explained by the importance of proper timing in task 1 (Reorientation), where it was critical for SBP to find the precise entry point in the alternative solution, which could lead to failed grasps and an unsuccessful task outcome. In task 2, this timing issue was less critical as the recovery behavior did not grasp the object again, but just executed an action with a different trajectory while the gripper remained closed.



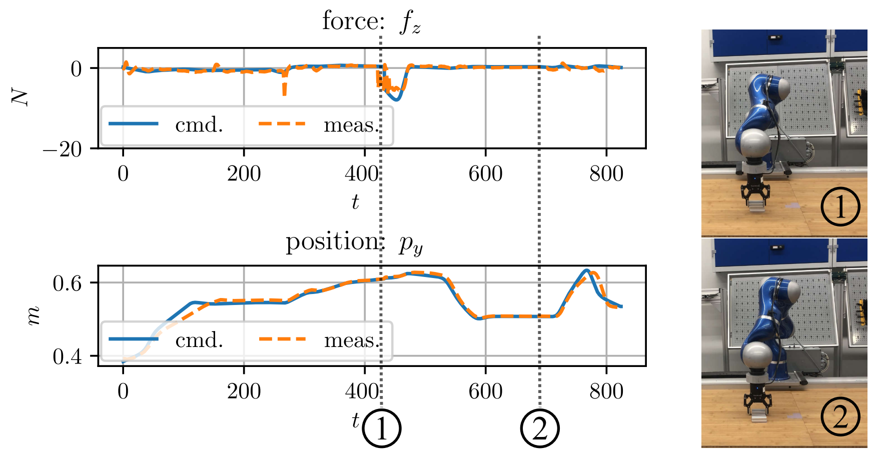
(a) Task 1: Reorientation: At time step (1), the robot detected an unexpected griper opening that triggered the switching to an alternative action. The robot chose a wrong entry time step in the alternative action, thus skipping the reorientation part, which caused the task execution to fail (2).



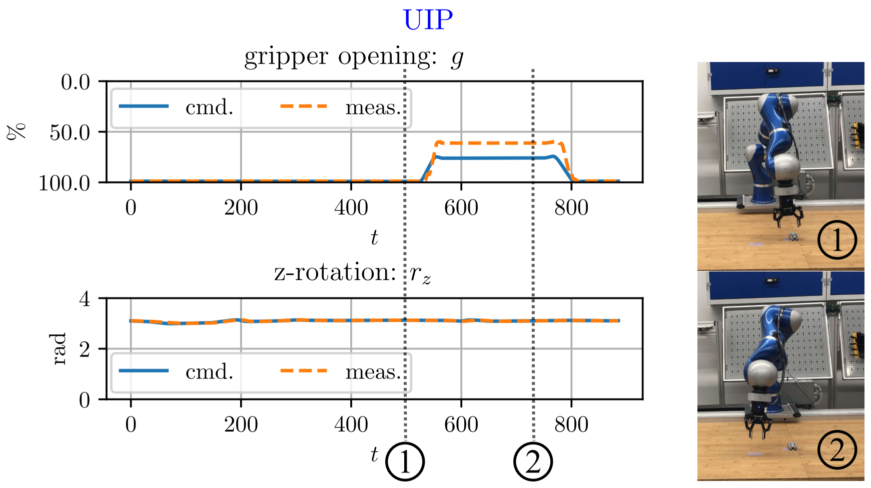
(b) Task 2: Contact-based Sorting: When trying to place the object in the occupied target location I, the robot sensed an unexpected force in the z-direction (1) that triggered switching to an alternative action. The robot transitioned to a correct entry time step in the alternative action, adjusting the end-effector's y-position (2) before it successfully placed the object in target location II.



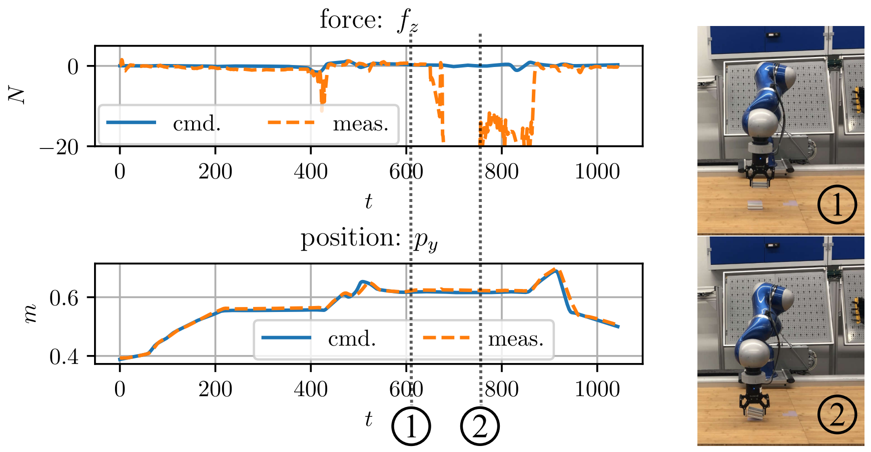
(a) Task 1: Reorientation: When grasping the object, the robot decided for the subsequent action s_2 in the decision state (1) based on the measured gripper finger distance. Using action s_2, the robot rotated the object before successfully placing it in the goal location (2).



(b) Task 2: Contact-based Sorting: When trying to place the object in the occupied target location I, the robot sensed a contact force in z-direction in the decision state (1) and decided for the subsequent action s_2. With action s_2, the robot adjusted the end-effector's y-position (2) before it successfully placed the object in target location II.



(a) Task 1: Reorientation: In the decision state (1), the next action was chosen before grasping the object, at a time step in which the robot did not interact with the environment and thus could not sense a difference between action s_{1B} and s_2. The robot selected the unsuited action s_{1B} in this situation, which led to an unsuccessful task execution (2).



(b) Task 2: Contact-based Sorting: In the decision state (1), the next action was chosen before the object in target location I could be detected by a contact force in the z-direction. The robot selected the unsuited action s_{1B} for this situation and tried to place the object in the occupied target location (2), thus causing an unsuccessful task execution.

Figure 7.30: Exemplary executions from the user study experiments.

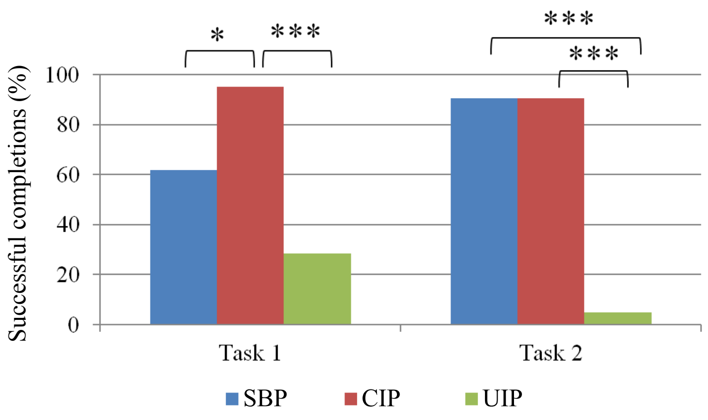


Figure 7.31: Successful completion of the different tasks for all three methods in percent.*, p < .05; **, p < .01; ***, p < .001.

$\mathcal{H}$1.2 holds for both tasks (✔) with significantly more successful task completions by using the autonomous anomaly detection of CIP in favor of a manual anomaly detection in UIP. Due to this discrepancy in the success rates, it was examined where exactly the decision states were inserted in these approaches.

Decision State Insertion This analysis only concerns CIP and UIP, which deal with decision states. The time step where the anomaly is detected by either human (in UIP) or robot (as in CIP) defines where the decision state is inserted into the task graph. This insertion is a critical operation as it determines when and where to switch to the appropriate action from the task graph during execution. The "when" is considered less critical since a task can be fulfilled at a wide range of speeds, and the pace of actions might differ throughout the task execution. Therefore, the "where" is considered of higher importance since the end-effector position has 1) the main sensing capabilities caused by interactions with the environment, and 2) has a major influence on causing forces and altering the environment. The end-effector position at the decision state is called decision state position.

A ground truth had to be defined for each task to analyze the distance between the decision state position and an optimal end-effector position. First, all decision state positions among all users were extracted for each task, considering only successful task executions of both CIP and UIP. Next, the mean positions were computed for each task, which are seen as near-optimal solutions. Finally, the distance $d_{EE,C}$ between each decision state position of the overall amount of trials and the ground truth was computed, shown as green marks in Fig. 7.32. With CIP (Fig. 7.32(a) and (c)), the automatically identified decision states are densely distributed around the ground truth while with UIP (Fig. 7.32(b) and (c)), the manually triggered decision states are distributed with a larger spread compared to the ground truth. These errors may lead to decision states that are not physically grounded because the targeted sensor signal is not present in that state. An exemplary scenario is when the user manually triggers a decision state that

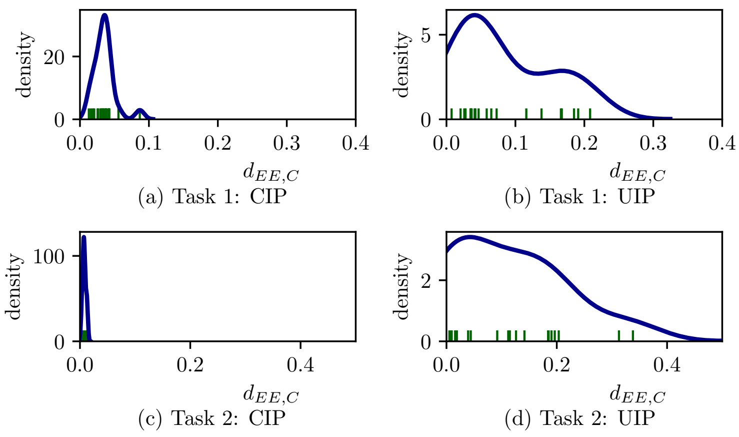


Figure 7.32: Each plot shows the probability density (blue curve) for the computed distances between decision state position and ground truth. These distances are marked by the samples on the x-axis (green ticks). The left column displays the automatically detected decision states by CIP, while the right column shows the manually triggered decision states by UIP. Automatically detected states (left column) lie closer to the ground truth.

should decide about the weight of an object before the robot actually grasps it, which makes it impossible to sense such property.

7.4.6 Subjective Results

QUESI Ratings Fig. 7.33 reports the intuitiveness of the compared methods. Users rated the intuitive use of SBP best, followed by CIP, except for "perceived achievement of goals," where CIP reached the highest score. UIP was rated worst for all scales. A repeated measures ANOVA showed that statistically significant differences occurred for the subscales "Subjective Mental Workload" (F(1.37, 27.39) = 5.36; p < .05), "Perceived Effort of Learning" (F(1.38, 27.67) = 5.39; p < .05) and "Familiarity" (F(2, 40) = 4.09; p < .05). Post-hoc comparisons showed that SBP scored higher for those items than UIP ("Subjective Mental Workload": p < .001; "Perceived Effort of Learning": p < .05; "Familiarity": p < .05) (see Fig. 7.33).

Workload Fig. 7.34 reports the NASA-TLX scores. A significant ANOVA main effect has been found (F(2, 40) = 4.30; p < .05). Post-hoc comparisons identified a significant lower workload for SBP (M = 4.48; SD = 2.21) compared to UIP (M = 5.82; SD = 2.79; p < .05). No significant difference was evident comparing CIP (M = 4.81; SD = 2.29) to any other method. $\boldsymbol{\mathcal{H}}$**3.1** suggests that the programming workload is reduced by CIP in comparison with SBP and $\boldsymbol{\mathcal{H}}$**3.2** suggests the same effect for the comparison of CIP with UIP. Both hypotheses were rejected. Instead, only a significant difference between SBP and UIP was found. That SBP shows the smallest workload rating could

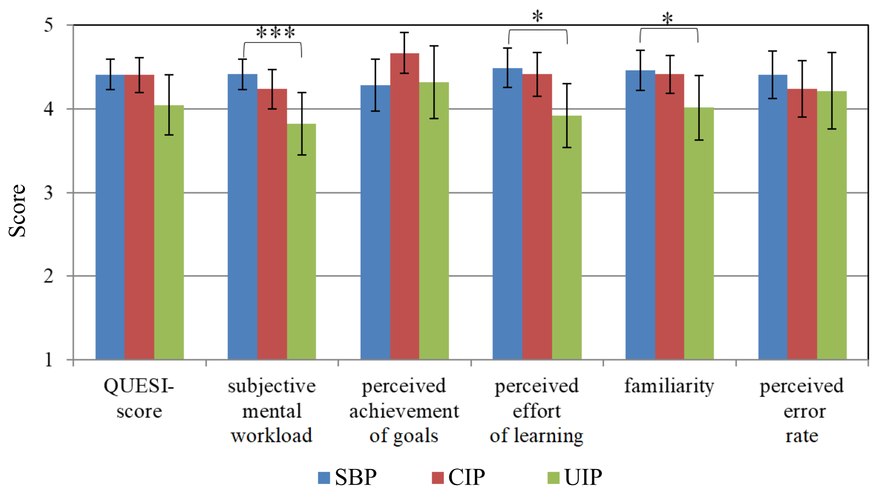


Figure 7.33: Scores for QUESI (error bars indicate 95% confidence intervals).*, p < .05; **, p < .01; ***, p < .001.

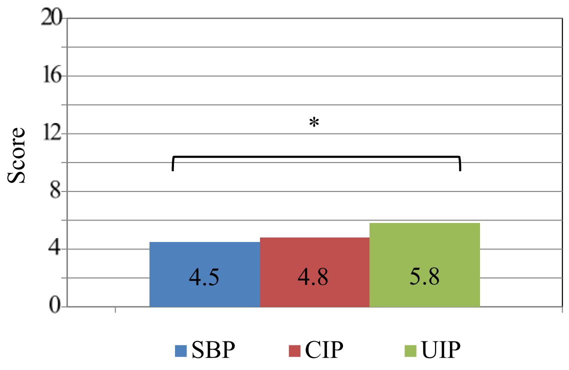


Figure 7.34: NASA-TLX workload

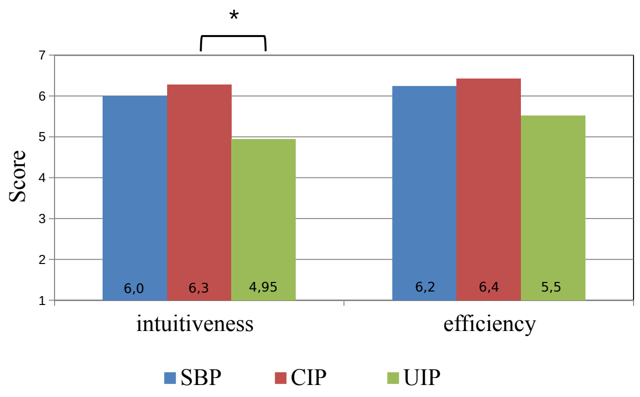


Figure 7.35: Scores for overall evaluation. p < .05; **, p < .01; ***, p < .001.

be explained by a minimum of required human-robot interactions since all knowledge is transferred sequentially in the teaching phase before the robot executes the task.

Overall Evaluation Fig. 7.35 shows the user ratings for intuitiveness and efficiency that are explained in the following. *Intuitiveness of the method (with the query shown to the user: "The method was easy to use and intuitive").* CIP (M = 6.29; SD = 1.35) and SBP (M = 6.00; SD = 1.10) were more intuitive than UIP (M = 4.95; SD = 1.69). This is supported by a significant ANOVA main effect (F(2, 40) = 4.89; p < .05), where CIP and UIP significantly differ (p < .05). Conventional level of significance for the difference between SBP and UIP was not reached (p = .053). $\mathcal{H}$**2.1** that suggests a higher intuitiveness of CIP compared to SBP in programming a task is supported by the overall QUESI ratings but without statistically significant effect (Fig. 7.33 very left). In contrast, $\mathcal{H}$**2.1** holds for the comparison of CIP with UIP (✔) and shows a significantly higher intuitiveness in programming a task.

Efficiency of the method (with the query shown to the user: "I could solve the given tasks efficiently with the method"). Subjects rated CIP (M = 6.43; SD = 0.98) as most efficient, followed by SBP (M = 6.24; SD = 1.09). UIP (M = 5.52; SD = .47) was slightly less efficient. However, this is not supported by a significant ANOVA effect.

7.4.7 Discussion

Objective Data The results from the number of successful completions (Fig. 7.31) show that programs created with SBP and CIP were able to solve both experimental tasks with relatively high success rates. In contrast, UIP failed in the majority of cases in both tasks. Due to the different abilities of the user and the robot to perceive the environment (e.g., vision), UIP cannot guarantee that the robot will be able to measure abnormal values when the user identifies a new situation and demonstrates an alternative behavior. As shown in Fig. 7.25b, many subjects did not wait before triggering a new decision state until the robot touched the environment and sensed the transition condition for the second sub-task. When programming the reorientation task, 13 participants demonstrated a new action before the robot closed the gripper to grasp the turned object. For the contact-based sorting task, even 16 subjects did not wait until the robot could detect an object in the target location. With CIP, however, a deviation in sensor values is a requirement for detecting new situations. Thereby, a measurable difference between the programmed transition conditions for every action of a *decision state* can be guaranteed. Consequently, it leads to a successful transition to the appropriate successor action when reproducing the situation because the measured sensor values reflect a programmed condition for action transitioning.

With SBP, a transition between actions is triggered when an anomaly is detected during the task execution. The time step of an action with the closest sensor values to the anomaly state is chosen as an entry point to continue the task. Since all time steps of all actions are potential candidates for the entry point, the approach is prone to perceptual aliasing, which causes transitions to wrong actions or entry points. Furthermore, the interpolation to the entry point does not guarantee a collision-free trajectory. In CIP, a transition between actions only happens in decision states. Therefore, it limits the number of possible successor actions to the intended ones and avoids perceptual aliasing by avoiding arbitrary transitions. Consequently, this guarantees a successful transition between actions when facing known situations.

Subjective Data Analysis of the questionnaires and the responses in the interviews led to the conclusion that SBP is an easily usable and intuitive framework for programming apriori known tasks and conditions. Compared to SBP, CIP has the advantage that overlapping parts of actions can be reused between different scenarios, and complex tasks can be incrementally generated. For tasks with several different decisions and actions, it is difficult for the user to predict all possible scenarios and demonstrate the corresponding behavior before execution. Hence, it can be argued that for more complex tasks, the advantages of CIP can be fully exploited since the user does not have to anticipate new situations but can demonstrate new actions on demand. Furthermore, the combined teaching and execution mode of CIP allows the users to instantly verify the result of their demonstrations.

Analyzing the NASA-TLX sub-categories reveals that CIP reduces the user's mental workload when programming a task, compared to UIP. CIP reached a score of 6.2 in this sub-category, compared to 8.5 for UIP. These outcomes match with the results from the guided interviews, where 19 of 21 participants mentioned as advantages of CIP that the robot autonomously detects new situations and that the user does not have to pay constant attention. The negative aspects for UIP, related to an increased need for attention and mental demand, were mentioned 17 times. The overall evaluation of the methods confirmed the increased intuitiveness of CIP over UIP. As mentioned 11 times in the interviews, deciding for the right moment to stop the task execution of UIP to add a new action is not intuitive for the user. This decision would require a deeper understanding of the principle behind the method. Fig. 7.25 shows an example where the robot automatically stopped the task execution when it sensed an anomaly. This

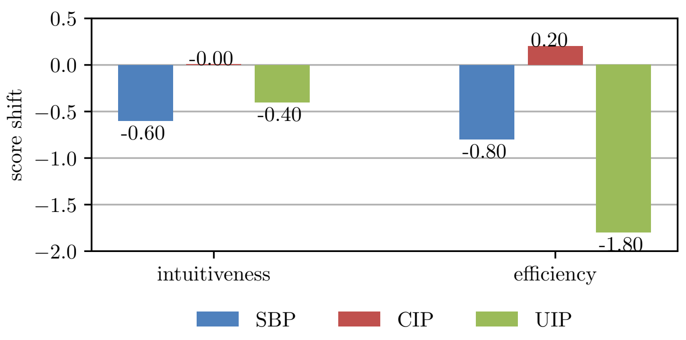


Figure 7.36: Shift of *intuitiveness* and *efficiency* scores before and after users have seen the robot's execution. A negative value means that users have downgraded their ratings on average compared to their first ratings.

automated behavior outperformed the human in most cases, who stopped the robot too early before it could sense the anomaly in the task environment.

7.4.8 Post-Experiment User Ratings

The robot's execution success was evaluated in the absence of the 21 users. Here, five of the user study subjects were consulted again after finishing their participation in the study. They were asked to rate again the intuitiveness and efficiency of each programming method. The ratings of these five users were also collected before they saw the execution. Hence, a comparison was made, analyzing the shift in intuitiveness and efficiency scores from before and after users have seen the robot's execution (Fig. 7.36). For both SBP and UIP, the intuitiveness and efficiency dropped noticeably, while for CIP, the intuitiveness remained the same (no change), and the efficiency increased by 0.2 points on the Likert-type scale. These results support the assumption that due to the bidirectional information flow between human and robot in CIP, the participants better understood how the robot changed its task knowledge compared to the other methods.

This study concludes that CIP is more transparent to the user regarding what the system has learned and what the robot is expected to do in the task execution. Related to that, an objective metric to evaluate the teacher's efficiency in robot learning is proposed in [188], given a specific feedback channel, e.g., by observing the robot's execution performance. From the objective success rate, it is concluded that CIP performed best given the experimental tasks. However, in PbD, many subjective circumstances caused by the human in the loop influenced the quality of interaction between human and robot. Hence, analyzing the effect of different task representations used as feedback channels in terms of teaching efficiency could aid developers in creating better user interfaces.

7.5 Conclusion

The problem of task decision programming concerns two scenarios. First, the programming of conditional tasks, for instance to allow sorting of objects by their object properties. Second, the programming of recovery behaviors, for instance to re-grasp and object in case of a task fault. Therefore, this chapter proposes two different methods, which are termed *multiple solutions* and *collaborative programming.* Both methods share an anomaly detection mechanism that allows the robot to switch to alternative actions that still solve the task. Further, both methods allow a user to program task decisions purely by demonstrations without the need to provide additional knowledge. In both cases, the complete behavior is transferred by kinesthetic teaching using the robot's proprioceptive capabilities as input modalities.

The main differences of the methods are the following. *Multiple solutions* work in two stages. First, the user provides a batch of demonstrations that accounts for possible task variants. Second, the robot executes the task and is able to switch to alternative solutions at arbitrary states once it detects an anomaly. While this might increase the flexibility to switch whenever desired, it could also lead to invalid switching conditions where the robot might execute unexpected paths. Therefore, the allowed region of switching could be manually limited.

Collaborative programming works with intertwined teaching and execution stages. The user starts with an initial set of demonstrations. Once the robot executes the task and detects and anomaly, the user can incrementally add knowledge to the system to account for further task variants. The automatic anomaly detection reduces the user's workload by just querying necessary information and guarantees a functioning task model, since transitions to other actions are only allowed within a decision state. This enables the user to scale the complexity of a task over time without demonstrating the whole task repeatedly.

A user study compared both methods also with a baseline method and examines their advantages and disadvantages in two different programming tasks. It also analyzes how users performed in reaching the task goals with each method. The study suggests that both methods are intuitively usable by non-experts that have only a minor experience in robotics. While *multiple solutions* provide more flexibility due to the possibility of switching at arbitrary states, it requires the user to foresee possible task variants. Instead, *collaborative programming* actively queries the user to add knowledge in case of unseen task variants. However, this methods constrains the system to only switch at so called decision states.

The conducted experiments stressed on the discrepancies in human and robot perception, since humans use vision but not every robotic system does so. Future approaches should incorporate a vision system as an additional sensor source given the fact that some anomalies can be observed visually before object interaction and some cannot, such as interaction forces. Beside that, the options for program flow control elements could be extended. So far, only branching of the execution was enabled by so-called decision states. Desirable would be also to merge execution flows at later stages or to allow creation of loop closures. Adding such flow control elements could be achieved manually or automatically by the system. Exploitation of extended flow control elements could be especially helpful to extend the options for recovery behaviors.

8 Unified Framework for Skill-based and Demonstration-based Programming

This framework allows end-users to intuitively setup new robotic tasks via PbD, employing skill recognition, automatic skill parameterization and task execution. PbD is combined with offline, skill-based programming via a GUI to propose a unified task definition process.
The chapter includes the content of the following publication [212]:

- T. Eiband, F. Lay, K. Nottensteiner, and D. Lee, "Unifying Skill-Based Programming and Programming by Demonstration through Ontologies," in *Procedia Computer Science*, 2023, p. accepted.

The author of this thesis integrated the task definition approaches into a single framework and developed the skill recognition, parameterization and execution pipeline. F. Lay implemented the GUI for manual task definition by offline, skill-based programming. K. Nottensteiner advised in the framework design and revised the article. D. Lee advised in developing research methodologies and analyzing the results and revised the article.

The chapter is outlined as follows. Section 8.1 describes the overall framework, Sec. 8.2 states experimental uses cases, and Sec.8.3 concludes the findings.

8.1 Framework

Modern robotic production sites require fast reconfiguration towards new products to deal with uncertain market conditions or problems in the supply chain. A key enabler for fast reconfiguration of a robotic workcell is that it must be intuitively programmable by the end-user. There are two main programming paradigms considered in this chapter. The first method is termed OSP, which is the method of arranging skills as building blocks of a robot program in a GUI. This method can be also used without blocking a robotic system that is currently performing a production task. The other programming method is known as PbD, which enables a fast and intuitive definition of new robot behaviors whenever the robotic system is directly accessibly and not currently involved in a production task.

Furthermore, this framework is connected to an ontology that represents the conceptual knowledge and the world model, based on the Factory of the Future (FoF) ontology developed for the automation and manufacturing domain [14]. The highlights of the proposed framework are

i) intuitive programming by combining (PbD) with OSP;

ii) automatic skill parameterization by combining demonstrated features with semantic knowledge represented by the ontology; and

iii) integration of skill recognition, automatic parameterization, and execution in a single framework.

Despite the desire for a high level of autonomy that would minimize manual programming efforts, achieving such autonomy is often challenging due to uncertainties and unforeseen edge cases that arise in new tasks. [105]. Instead of achieving a high level of autonomy, the presented framework in this chapter tries to reuse exiting knowledge about the concepts in the world in combination with information provided by the end-user either in the form of manual programming or via programming by demonstration. It unifies the concepts of PbD and OSP with the help of an ontological knowledge base, such that the end-user only needs to take care of missing task knowledge and where the system exploits existing knowledge wherever possible. It is aimed that an end-user would use OSP for well-known, easy to describe parts of a tasks and PbD for parts that are more intuitive to be demonstrated and difficult to model.

The method of OSP allows the user to manually add skills in the task definition process with the help of the so-called Human Factory Interface (HFI) [14, 213]. This GUI visualizes an editor for task definition and offers a library of robot skills, that each can be manually parameterized according to the task needs. The PbD method in this framework lets the end-user intuitively set up new robotic tasks by demonstration, using skill recognition, automatic skill parameterization and task execution as a sequence of parameterized skills. The process of automatic skill parameterization involves extracting parameter values from data instead of manually specifying them. This approach requires only a single demonstration to achieve the following two objectives: 1) defining the skill type, and 2) automatically parameterizing it. This is ultimately achieved by connecting this framework to an ontological knowledge base, which represents conceptual knowledge of the system as well as resembles a world model, which represents geometric information about the environment. The underlying ontology was developed for the automation and manufacturing domain and is called FoF, where a detailed description can be found in [14].

Figure 8.1 shows an overview about the unified programming framework. It allows to define a new task either by PbD(blue enclosed area) with the demonstration data $\boldsymbol{X}$ or graphically by OSP (green enclosed area) with the manually selected actions $\boldsymbol{A}$. The output of both programming methods is fed to the execution stage, which runs the actual skill instances on the robotic hardware. These skill instances command the robot via the associated robot hardware interface, manipulate the physical world, and update the representations of the objects in the knowledge base. The motion planner module can be queried from each skill instance and provides collision free paths in the environment of the workcell. It is interfaced with both the execution stage and a central knowledge base, depicted as ontology. The knowledge base serves geometric data about the environment, and conceptual data, such as the relationship between knowledge entities and their properties.

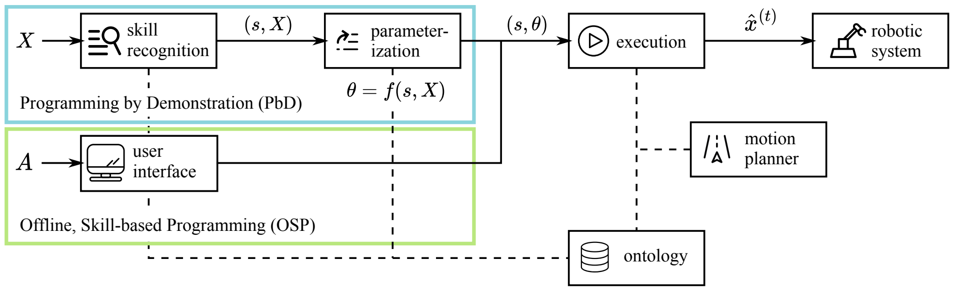


Figure 8.1: Unified programming framework. Solid lines show the process flow, dashed lines show the data exchange with the ontological knowledge base (ontology).

The connection to the ontological knowledge base has two major advantages when combined to the PbD method. First, the skill recognition process can make use of a set of known skills and can query semantic data during the recognition phase if a skill is valid in the current context. An example is that a place skill is only a valid candidate if an object is currently present in the gripper. Second, it also simplifies the skill parameterization process because knowledge about the environment exists apart from the human demonstration data. For instance, the skill parameterization process can exploit information about the arrangement of objects in a workcell, shaping the motion that a specific skill should perform in this situation.

8.1.1 Skill Recognition and Parameterization

The blue-bordered area in Fig. 8.1 refers to the programming mode of PbD. The skill recognition method is based on an earlier work [154], which mainly considers skills with physical contact, such as *touch*, *press*, or *insert*. Later, it was extended to manipulation skills such as *pick*, *place*, or *move* [184]. The skill recognition stage accepts a time series $\boldsymbol{X}$ from a human demonstration and outputs the tuple $(s, \boldsymbol{X})$ with the predicted skill s that is displayed to the user. Then, the parameterization stage automatically derives a parameter set $\boldsymbol{\theta}$ using the mapping function $\boldsymbol{\theta} = f(\boldsymbol{X})$ and delivers the tuple $(s, \boldsymbol{\theta})$ to the execution stage.

8.1.2 Offline, Skill-based Programming

The green-bordered area in Fig. 8.1 refers to the programming mode of OSP and uses the HFI as programming interface. The user actions are referred to as $\boldsymbol{A}$. The first user action is to select a predefined skill from a skill library. The second user action is to define the parameter values of the skill. Here, the HFI guides the user through the parameterization process, displaying only parameter values that are valid in the current world state. For instance, the user can only select objects that are available in the current workcell and are not yet bound as a resource elsewhere. Analogous to PbD, the output of the OSP stage is a tuple $(s, \boldsymbol{\theta})$ that is passed to the execution stage.

8.1.3 Task Execution

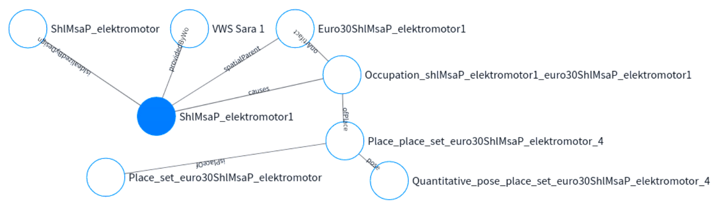
The execution stage performs two key tasks: running a skill and configuring its underlying primitives dynamically at runtime. A skill consists of reusable building blocks called primitives, denoted as $< ... >$, which represent atomic functions of the robot. Some examples of such primitives are $<$ planned_motion $>$, $<$ open_gripper $>$, and $<$ cartesian_path $>$. This work assumes a sequential arrangement of primitives within a skill, although other structures like decision trees or state machines are possible. During the execution phase of a skill, each primitive is configured based on the skill's extracted parameters $\boldsymbol{\theta}$. A mapping function, denoted as $\boldsymbol{\theta}_p = f_p(\boldsymbol{\theta})$, determines the configuration of a primitive p in $[1..P]$. Suppose that there exists a $<$ planned_motion $>$ primitive that represents an approach path for an insertion skill. Analyzing force and torque values and applying a threshold allows to identify the first contact point on the target object. The endpoint of the approach path can then be extracted relative to the object, allowing to plan future motions towards different target object locations.

Skill parameters are specific to a particular task and remain constant for the same task goals, while capturing the overall requirements and objectives of the skill. On the other hand, primitive parameters are context-specific and can change depending on the specific location or properties of each object involved in the task. These parameters are adjusted to the current world state and might not be known before task execution. During execution, a skill orchestrates its underlying primitives and the controllers they implement. This requires each primitive to generate the control input $\hat{\boldsymbol{x}}^{(\boldsymbol{t})}$ at each time step t. The control input guides the actions and behavior of the robot during the execution of the skill, ensuring that the desired task goals are achieved.

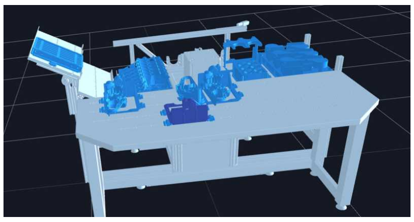
8.1.4 Ontological Knowledge Base

The development of robotic domain ontologies has made significant progress in representing the abilities and skills of robotic agents in the manufacturing domain, as well as concepts related to autonomy. These ontologies serve as structured, human readable notations that enable the modeling and sharing of knowledge in the robotics domain. One contribution in this area is the IEEE 1872-2015 standard, which provides a core ontology that defines the fundamental concepts, relations, and axioms of robotics and automation most general concepts, relations, and axioms of robotics and automation [214,215]. It serves as a foundation for building more specialized ontologies, like the FoF ontology [14]. It extends the already standardized concepts by object manipulation properties relevant to assembly processes. This includes for instance the modeling of pick poses, approach and departure paths, and entities like `Parameter, Skill, Task, Place,` and `StorageDevice`. Each entity is uniquely addressed by an Internationalized Resource Identifier (IRI). This ensures that the information captured in the ontologies can be accessed publicly and enables the sharing of conceptual knowledge among the robotics community and among robotic systems.

Every physical object, referred to as an `Artifact`, is associated with a transformation relative to a spatial parent. For example, an object located in a `StorageDevice` has a


ShlMsaP_elektromotor
VWS Sara 1
Euro30ShlMsaP_elektromotor1
isIdealizedByDesign
providedByWo
spatialParent
onArtifact
Occupation_shlMsaP_elektromotor1_euro30ShlMsaP_elektromotor1
causes
ShlMsaP_elektromotor1
ofPlace
Place_place_set_euro30ShlMsaP_elektromotor_4
isPlaceOf
Place_set_euro30ShlMsaP_elektromotor
pose
Quantitative_pose_place_set_euro30ShlMsaP_elektromotor_4

(a)



(b)

Figure 8.2: Graphical representations of ontological knowledge. (a) shows entities (circles) and their properties (lines) that are modeled in the ontology. The example shows a physical object marked as blue dot (`Artifact`) of a specific type ($\rightarrow$*idealizedByDesign* $\rightarrow$`Design`) and how it is geometrically ($\rightarrow$*spatialParent* $\rightarrow$`Storage`) and semantically ($\rightarrow$*causes* $\rightarrow$`Occupation` $\rightarrow$*ofPlace* $\rightarrow$`Place`) referenced in the ontology. (b) Visualization of the variable workstation.

transformation chain that leads over a specific `Place` to the frame of the storage. This concept allows to compute transformation chains between any entities with a geometrical context. A `Design` represents the specific type of an object and exists as a single instance. It has assigned the property of `GraspSet`, which specifies at least one grasp pose, as well as an approach and departure path, all expressed relative to the object. This allows to define this information only once and to be shared among all `Artifacts` of that `Design`.

All objects of the same `Design` that reside in the same `StorageDevice` can seamlessly be queried, which is later exploited in Sec. 8.2.1 to find a pattern in the arrangement of objects. A further example is the placement of objects on a predefined `Place`, which can make use of modeled approach paths and create a so-called `Occupation` on the respective `Place`. Since the relative transformation between `Design` and `Place` is known, autonomous placement of objects becomes possible. This allows also to reuse previously learned behaviors from PbD by transforming them to new target frames of a new `Place`, as exploited in Sec. 8.2.2. Figure 8.2a shows an excerpt of the modeled knowledge. Figure 8.2b shows the geometric world representation of a variable workstation.

8.2 Experimental Use Cases

All experimental scenarios were implemented and tested on a DLR SARA lightweight robot [216], mounted on a variable workstation of the flexible production network, as shown in [14, 213].

8.2.1 Object Identification by Touch

In a shared workspace, either humans or robots can manipulate objects, resulting in changes to the physical world. The robot utilizes its skills to update not only the physical state but also the modeled world state, which is represented in the knowledge base. On the other hand, humans may only modify the physical state of objects without necessarily affecting their modeled world state, as it is often difficult for a system to track all the changes introduced by human actions.

There are several reasons why there can be a mismatch between the physical state and the modeled state of the world. These include:

1. Shared assembly processes, where both users and robots manipulate parts, where especially human actions are hard to track by the system
2. Manual removal of objects due to quality checks or sorting out defective parts.
3. Incompletely filled storage arrangements, as depicted in Figure 8.3c, may be expected to be completely filled by the system, but are not in reality.

To address these issues, a *touch* skill is employed in this use-case. This skill allows the system to haptically explore the environment until it locates an object. It is specifically used to identify the next object in an object storage that does not match the modeled state due to the aforementioned reasons.

Implementation The implementation of the *touch* skill is based on the concept introduced in [42]. The objective is to identify a linear exploration path that locates an object within the workspace. In the original work [42], the skill acquired all necessary information through multiple user demonstrations. In contrast, this implementation also exploits the information stored in the ontology to infer the exploration path. Consequently, the system can fully parameterize the skill based on just a single demonstration, resulting in reduced teaching effort. Additionally, the utilization of the ontological knowledge base enables collision-free movements and improves the inference about the exploration path because the expected object locations are already modeled in the ontology.

In more detail, the internal parameterization of the skill constructs multiple primitives based on the initial demonstration (Fig. 8.3b). The first primitive, denoted as $<$ cartesian_path $>$, approaches the designated exploration region. The second primitive, denoted as $<$ cartesian_move $>$, locates the next object by a linear exploration motion. This linear motion is determined by querying all potential storage locations from a storage container. Next, a singular value decomposition (SVD) is performed on the storage positions to identify the best-fitting line through these positions. Notably, the eigenvector as shown in Fig. 8.3b might be anti-parallel to the intended exploration direction, depending on the outcome of the SVD. The identified line is then translated to intersect the contact point observed in the demonstration, resulting in the actual exploration path. A representation of the parameterization can be found in the experimental results (Fig. 8.3b).

The skill execution process arranges the previously generated primitives in sequential order, and the transition from one primitive to the next is determined by predefined conditions. While executing the exploration path, a crucial transition condition is the detection of the contact between tool and object. Upon detecting this contact, the system transitions to the subsequent primitive, which is designed to depart from the contact point.

Experiment The procedure is as follows:

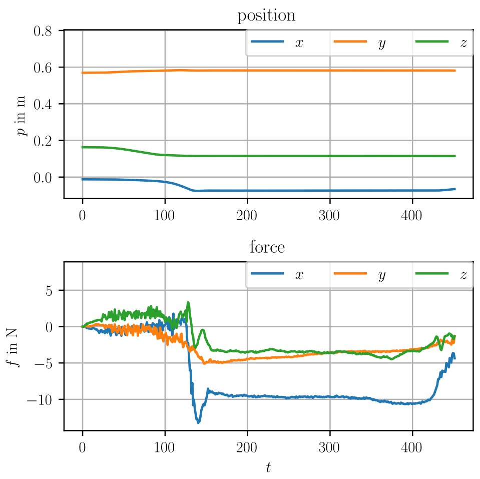
1. User creates a new empty task via the HFI (the whole task is shown in Fig. 8.3d).
2. User manually adds an *equip_device* skill with parameter: `tool: robotiq-140`, attaching it as tool.
3. User provides demonstration of touching an object in the workspace (Fig. 8.3a and 8.3c).
4. Robot automatically recognizes a *touch* skill and adds it to the skill sequence
5. User manually adds a *pick* skill with parameter: `object: housing`, defining the type of object to be picked.
6. User starts the fully defined task.
7. Robot executes task with identification of new object location, and picks the next object of same type (Fig. 8.4).

The experiment shows that the user can demonstrate one part of the task that is automatically recognized as *touch* skill by the system. The user then proceeds to define the task further by manually adding the *pick* skill. As the *pick* skill is already implemented and can function independently, it merely needs to be adjusted with parameters without requiring a user demonstration. Hence, the user demonstrates only those parts of the task where the knowledge base lacks information or where the user prefers the programming mode of PbD rather than OSP.

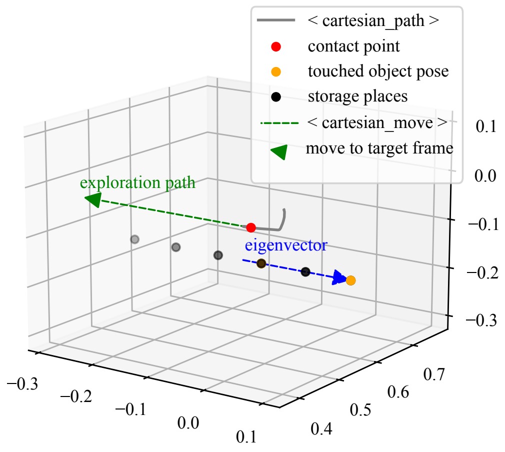
8.2.2 Object Insertion

Implementation The *insert* skill is used to assemble an object by inserting it into the fit of a target object, as it is known for instance from peg in hole tasks. This implementation uses a time-based motion and force trajectory that is extracted from the user demonstration. By accessing the knowledge base, the skill can query the coordinate frame for new target objects, which enables it to adapt and apply the learned strategy with respect to the target object's coordinate frames. Hence

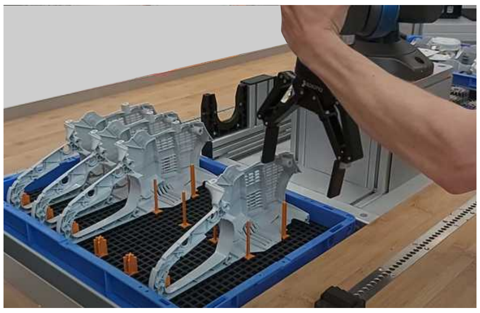
The skill parameterization constructs the following four primitives from the demonstration data (Fig. 8.5b). The initial primitive, denoted as "$<$ planned_motion $>$", is constructed by extracting the first physical contact point from the demonstration until the object makes contact with the environment. This contact point serves as the target frame for motion planning. The second primitive, denoted as "$<$ motion_force_path $>$", is shaped based on the demonstration and replicates the demonstrated trajectory and wrench. It uses a Cartesian impedance controller with a wrench overlay and predefined stiffness values. The third primitive, denoted as "$<$ open_gripper $>$", operates the gripper once the insertion is finished. The fourth primitive, denoted as "$<$ cartesian_move $>$", departs linearly from the insertion location in opposite direction of the insertion direction. During the skill execution, these primitives are sequenced in sequential order, while transitioning from one to the next without additional constraints.



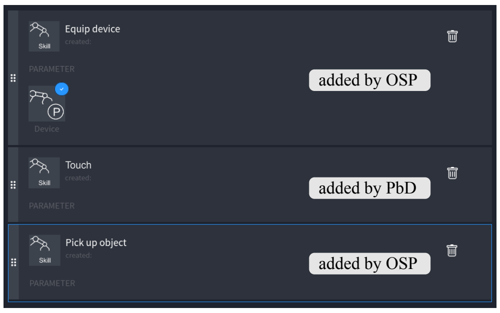
(a) Demonstration data.



(b) Derived parameters and primitives denoted as < ... >.



(c) Demonstrating how to touch an object in a storage.



(d) Task Representation as shown to the user in the HFI

Figure 8.3: Demonstration of touching an object in a storage.

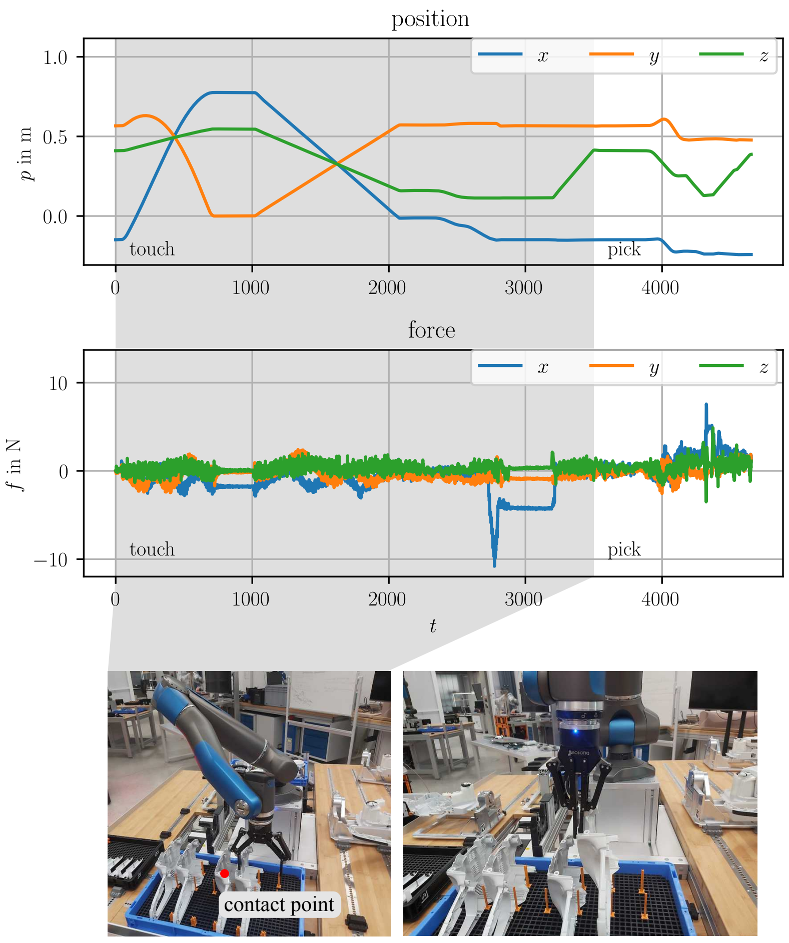


Figure 8.4: Sequenced execution of *touch* and *pick*.

Experiment The procedure is as follows:

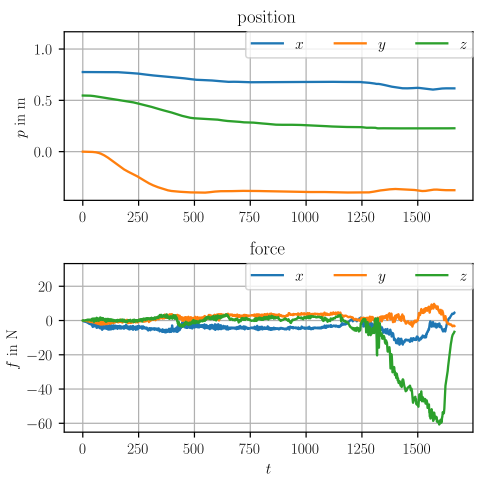
1. User creates a new empty task via the HFI (the whole task is shown in Fig. 8.5d).
2. User manually adds an *equip_device* skill with parameter: `tool: robotiq-140`, attaching it as tool.
3. User manually adds a *pick* skill with parameter: `object: motor`, defining the type of object to be picked.
4. User provides demonstration of inserting a motor into the housing (Fig. 8.5a and 8.5c).
5. Robot automatically recognizes an *insert* skill and adds it to the skill sequence.
6. User starts the fully defined task.
7. Robot executes *pick* skill and picks motor.
8. Robot executes *insert* skill and inserts motor into housing.
9. Robot executes task of inserting the motor into a new target housing by reusing the demonstrated strategy (Fig. 8.6).

In the same manner as in the previous experiment, users have the option to demonstrate specific phases of the task that are recognized as skills. Alternatively, they can manually define parts of the task by adding skills to the HFI. The advantage of this combination is that only the parts of the task are demonstrated where the knowledge base lacks information or when the user favors the programming mode of PbD over OSP due to personal preferences.

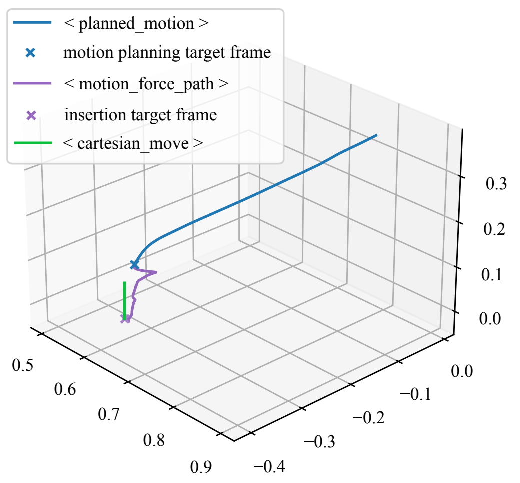
8.3 Conclusion

The presented framework combines a manual, skill-based programming with PbD. The PbD programming mode integrates the full pipeline encompassing user demonstration, skill recognition, skill parameterization, and skill execution. A first advantage of this framework is that the user is relieved from the burden of knowing all individual skills or searching through an extensive collection of skills. Instead, task phases are simply demonstrated and a suitable skill is suggested by the system. This suggests that novice users should familiarize quickly with such as system without requiring prior knowledge about the systems capabilities that is implemented in the collection of skills. Subsequently, a recognized skill can be automatically parameterized using the same demonstration data, without requiring further manual input in comparison to OSP. The programming modes of PbD and OSP can be combined, providing flexibility to the end-user to select a suitable programming method based on personal preferences.

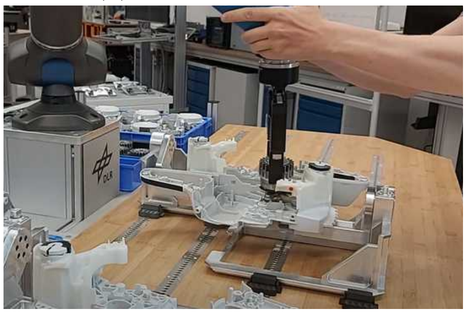
In this chapter, it has been shown that the skill recognition and skill parameterization approach can be enhanced with ontological knowledge, which reduces additional demonstration effort. In comparison to approaches that exploit multiple demonstration trials to cover distributions about object locations as used in [42], the demonstration effort could be reduced to a single trial. This is validated in the first experiment, where the



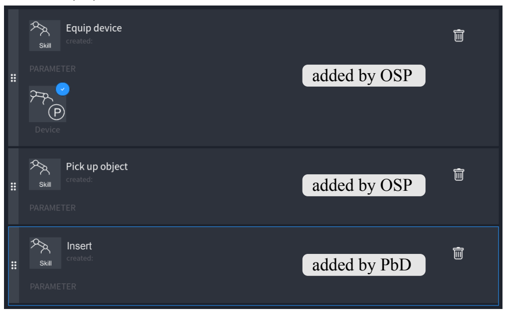
(a) Demonstration data.



(b) Derived parameters and primitives.



(c) Demonstrating how to insert the motor into the housing.



(d) Task Representation as shown to the user in the HFI

Figure 8.5: Demonstration of touching an object in a storage.

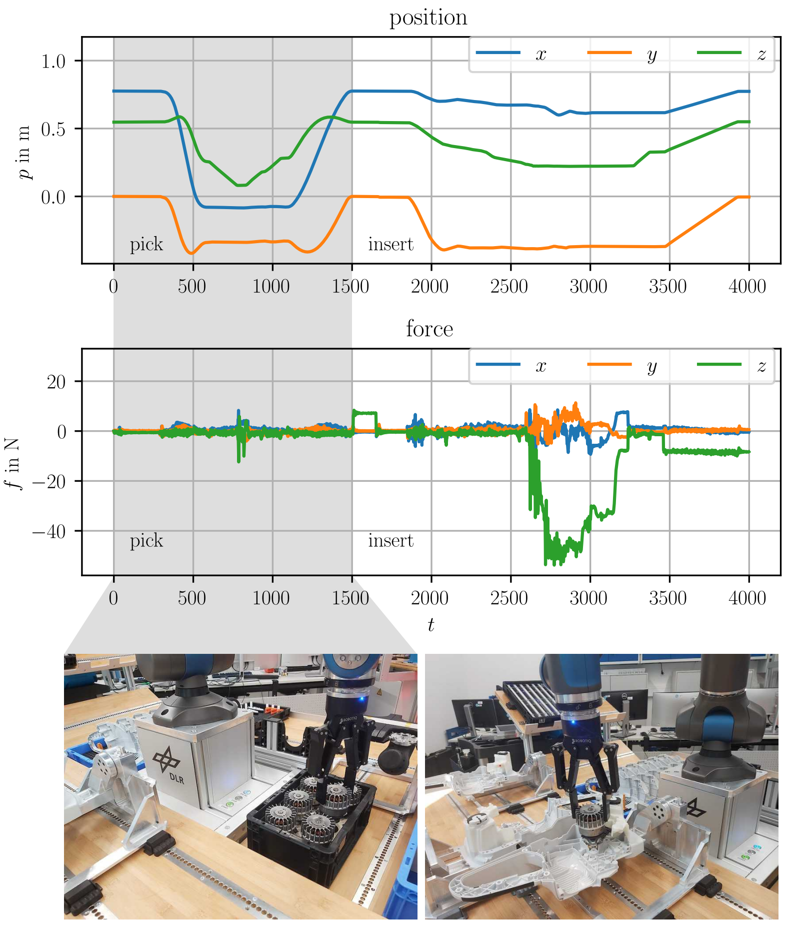


Figure 8.6: Sequenced execution of *pick* and *insert*.

task is represented by a sequence of a *touch* and a *pick* skill. First, a *touch* skill is recognized from demonstration, whose parameters for its exploration path are automatically extracted from both the information contained in the demonstration and ontology. The knowledge that is embedded in the *touch* skill implementation is exploited during execution by halting the robot at the explored contact point and by updating the inferred object position. Second, a *pick* skill is added manually via the user interface, which accounts for the extracted object location of the prior *touch* skill during execution.

This chapter also shows that LfD can be used as input for a skill parameterization technique, which yields optimized, adaptable robot behaviors in the form of previously implemented skills. The second experiment highlights this feature by considering a task that sequences a *pick* skill and an *insert* skill. First, a *pick* skill is manually added to the task via the user interface. Second, a human demonstration leads to the recognition of an *insert* skill, which is parameterized automatically from the demonstration. It uses a planned motion phase to approach the insertion and an impedance controlled, force-superimposed insertion phase during execution.

The two distinct programming modes, PbD and OSP, exhibit fundamental differences in their interaction behavior between human and robot. On the one hand, PbD heavily relies on haptic feedback while the user can directly check the feasibility of the guided motions. However, the user might be negatively affected by the robot dynamics while the arm is manually guided. On the other hand, OSP primarily relies on visual feedback. However, the user cannot perceive the physical correspondence between robot and environment and needs to rely on the representations that are displayed on the screen. Both techniques come with technical advantages and drawbacks, while it is a clear advantage that the user has the superiority to decide about which programming mode shall be employed. This empowers end-users to select the interaction mode that best matches with their personal skills and allows to efficiently solve the respective part of the task.

The main subjective factors that determine the selection of a programming mode are seen in 1) the detail of modeling of the robotic system and its surroundings; 2) the user-specific preference and competence over the teaching modality; and 3) the task-specific requirements that allow each teaching modality. Further studies should analyze these factors in depth to assess the future importance of PbD and LfD beside other programming modes.

9 Conclusion

This thesis focuses on the challenges of robot programming, which is typically limited to experts due to its complexity. To address this, several intuitive programming methods are proposed that work with human demonstrations as input. One path is to first identify required skills, such that they can be automatically recognized from human demonstration data. These skills can than be automatically parameterized and executed without writing a single line of code. Another path allows to program task decisions and recovery behaviors by learning from multiple demonstrations. Experiments and user studies validate the proposed methods, showing that the proposed skills are interpretable by both humans and robots and can be recognized in real-time during kinesthetic teaching. Finally, a unified framework combines automatic skill recognition and parameterization with existing manual programming methods, showing that the proposed method can be well integrated in realistic setups.

9.1 Skill Identification

Identification of suitable skills has been described in Chapter 4 and highlights the importance of pre-defined, human understandable skills, that can be at the same time recognized by a technical system. On the one hand, such skills has been selected from daily live scenarios, for instance touching different objects, applying force onto an object by pressing, or sliding over a surface with constant force. On the other hand, such skills have been employed in robotic scenarios in this thesis. Ideally, each of the identified skills is a reasonable human action that can be easily understood by different users, and at the same time, resembles a meaningful robotic behavior that can be recognized by a robot and be used in a robotic manipulation or assembly task.

The concept of skills allows to embed expert knowledge into each skill, which hides the implementation details from the end-users. This is beneficial, since the end-user does not need to deal with the low level functions of the hardware and can build a generic understanding of what the skill does independent of the hardware. In addition, skills can be developed hardware independent if they make use of hardware abstraction, for instance in the form of an implementation layer that uses primitives, which connect the required functionality of the skill with the hardware itself.

The proposed skills in this work are a comparable small set with respect to all possible human actions. However, these skills were extracted to be used on a robotic system and could already fulfill a variety of tasks in the manufacturing domain. The question of which skills exactly need to be identified and provided to a robot is also of philosophical nature, since it is arguable how a skill has to be labeled to express its capability, which capability it should characterize, and how specific or generic such a behavior shall be.

An example is given with the insert skill, which could either be able to insert all kind of objects and shapes, or there could be a batch of dedicated skills for different constrained insertions, such as round-insert, rectangular-insert, or shallow-insert. Of course, inherent generality and good adaptability is favored by researchers, system designers and end-users. In reality, a good generalization behavior of a skill is often hard to achieve. This challenge often forces system designers to develop dedicated skill implementations that solve only the specific part of a task. Highly specialized skills are, for instance, listed in a survey about capability-based frameworks [13].

No matter on how specific the intended usage of a new skill definition shall be, the identification procedure in Sec. 4.1 shall aid future researchers and system designers to put an eye on how skills shall be labelled legibly such that the system users easily understand what the system is capable of. Consequently, this increases the explainability during the programming and execution phase and increases the capability of the system to explain itself.

9.2 Skill and Skill Sequence Recognition

Once a set of skills has been identified, it can be used in robot programming, termed as skill-based programming. One paradigm of skill-based programming is to manually arrange and parameterize sequences of skills. The paradigm promoted in this work is PbD, where a novice user either demonstrates individual skills to be recognized (Sec. 4.2), or demonstrates the whole task, requiring the recognition of a sequence of skills (Chapter 5). This thesis presents the individual recognition of skills from a set of eight contact skills and the recognition of skill sequences both involving contact skills and manipulation skills such as pick and place. The major aims of skill recognition are to connect LfD with a comprehensive knowledge representation about what the robot has learned, the ability to represent and adapt the robot program by the end-user in the form of understandable building blocks, and the usage of dedicated algorithms during execution time that are embedded in the skill.

In comparison to the individual recognition of skills, it might appear more intuitive to demonstrate the whole task at once, leading to the recognition of skills in a sequence. However, both approaches have their advantages and limitations. Demonstrating a single skill is mentally less demanding and the user can fully focus on one or multiple trials of the same task phase. Additionally, the user has the freedom about what to demonstrate, what to specify manually, or what to reuse from previous demonstrations. As a drawback, the user is required to handle a multitude of individual interaction with the system. This involves for example the triggering of demonstration start and stop signals, checking and optionally modifying the result of each interaction, and providing additional information for each skill. Demonstrating the whole task at once results in a sequence of skills and the construction of a complete task representation. Parameterization of individual skills is simplified since they are demonstrated in conjunction, where the existence of a valid motion path is evident from the demonstration. As a limitation, the mental demand of the users is increased as the whole task has to be demonstrated at once and errors in the demonstration can possibly lead to an unusable skill sequence. Depending on the underlying framework, the user has to demonstrate the whole task again. This is encountered in the framework described in Chapter 5, where incorrect skills can be still corrected afterwards due to the skill-based task representation.

In general, the proposed frameworks focus on the recognition of contact-based skills, which received less attention than unconstrained manipulation skills. Nevertheless, contact skills deserve the same attention since end-users need to understand what the robot actually understood from their demonstration and furthermore, the selection of the appropriate robot behavior does not solely rely on the demonstration but rather on the skill implementation that executes the desired behavior. Due to the supervised learning paradigm in this work, it is straightforward to label a skill in a comprehensive manner since the labels are predefined and proven to be understandable, as it has been shown in the skill identification of Sec. 4.1. In comparison, unsupervised approaches might generalize better to unseen situations and work without predefined skill types but lack the ability to express the results in a comprehensive way that user can understand.

9.3 Contact-based Exploration

Contact based exploration can be used to explore the presence or location of objects in a workspace with the proprioceptive sensors of a robot. Chapter 6 introduces such approach, namely haptic exploration, which makes use of skill sequence recognition, but extends this approach by extracting the skill sequence from multiple demonstrations of the same task. The results showcased that a system can effectively learn an exploration behavior in multiple directions of the workspace, where only one demonstration per exploration direction is needed. This results in a behavior where the robot is able to adapt subsequent skills by means of the explored object locations. An example is that an object can be picked relative to the location, where it has been explored in a previous step.

The tackled problem is to identify the position of an object in a bounded area of the workspace, while ignoring its orientation. This works for task where the environment is partially structured and the robot has to deal with object arrangements such as piles or stacks with one or more dimension of arrangements, for instance an array-like structure storage. The benefits are seen in its robustness, since it can cope with such situations without employing visual perception. This allows to adapt robot behaviors in occluded or rough areas, and estimates the state of objects in the workspace up to the precision of the robot itself, which is likely more precise than a visual estimation. Generally, the proposed technique could be also used to complement other sensing techniques. For instance, a system that is equipped with a pan-tilt camera might be bound in observing one part of the workspace while the robot haptically explores another part of it. One important thing is that the exploration strategy can be tightly tailored to the application needs, since the resulting strategy is closely derived from the human demonstration, which ensures that no unpredictable robot motions will ever occur.

The presented approach deals with multiple demonstrations, namely one demonstration per exploration direction. Although, this method tries to minimize the number of teaching interactions with the system, it still requires the user to demonstrate the task multiple times in a similar manner. On the one hand, it is remarkable that all necessary information can be extracted by a little number of demonstration without the need to provide prior knowledge about the task. On the other hand, it might not be a teaching paradigm for end-users that quickly want to achieve their goals and who fear the increased mental demand. Although there are numerous research works with attempts to learn from multiple demonstrations, it might be advantageous to combine single task demonstrations with an option to add further constraints, such as by means of a GUI.

9.4 Task Decision Programming

Two of the limitations of traditional LfD are a limited robustness to uncertainties when learning from single demonstrations and transfer of decision making behaviors without providing prior knowledge. Chapter 7 proposes strategies to cope with both of these limitations. First, robustness can be increased by learning from multiple demonstrations, and second, decision making can be extracted from multiple demonstrations or by multiple system interactions with the help of an actively learning system. Programming of decision making behaviors also refers to the problem of recovery behavior programming, which copes with anomalies during execution in the same manner as online decision making does, which switches to an alternative execution flow.

In the first approach, presented in Sec. 7.2, the user has to think in advance what is needed for the task decision to be learned, and then comes up with a batch of demonstrations that include all eventualities, represented as multiple solutions in the task representation. The approach in Sec. 7.3 extends this idea and lets the user collaboratively program all required task decisions together with the robot. This is achieved by a continuous interaction scheme in the context of active learning, where the system queries information from the user only when it is not confident to solve it alone. The confidence is derived from multiple demonstrations of the same task phase. Whenever the confidence bound is violated in an anomaly detection scheme, it leads to either a user interaction, if no additional knowledge is present in this area, or it triggers an alternative action that is intended to resolve the faulty state.

A major aim was to reach a highly intuitive teaching scheme for end-users, which was compared to a baseline method in an exhaustive user study involving 21 subjects. The presented framework in Sec. 7.3 received the best scores in the subjective user ratings since it lets the user directly start the programming phase, queries only relevant information when needed, keeps the mental load low due to the autonomous anomaly detection, and is able to present the learned task in an understandable graph-based task representation. Further, it also received the best score in the category of successful execution, since it avoids problems in the process of decision making by using a structured knowledge representation, which is implemented as task graph.

Overall, Chapter 7 has shown that task decisions can be solely programmed by demonstration without the need of adding prior knowledge about the task. The only information that is needed in Sec. 7.2 is the number of expected task solutions that the algorithm shall learn. To recall that it is physically and mentally demanding for the user to provide multiple demonstration, the approach of collaborative programming achieved a good trade-off between its benefits and drawbacks. On the one hand, it can be already used as single-shot learning algorithm if no task decisions need to be programmed. On the other hand, it can be extended by adding further demonstrations at runtime, once an anomaly has been detected that requires human intervention.

9.5 Unified Framework for Skill-based and Demonstration-based Programming

It is important to highlight that the proposed methods in this thesis do not only work in an isolated setup but can also be integrated into a realistic framework in combination with other task definition methods (Chapter 8). A key enabler for this integration is the ontological knowledge base that exchanges the information from multiple input sources using a structured format.

The full integration of the pipeline from user demonstration to skill execution highlights the benefits of automatic skill recognition in a realistic scenario. Here, end-users can simply demonstrate the desired behavior without knowing the entire set of skills. If applicable, recognized skills can be automatically parameterized from the same demonstration data, eliminating the need for additional manual effort. If that is not applicable, the user can always add missing parameterization or whole skills via a GUI

The experiments have shown that a user can either start with PbD or OSP and alternate between these teaching modes such that the whole task can be efficiently programmed and only knowledge that is not yet present in the system is added via PbD. The ontological knowledge that is already modeled in the system complements the skill recognition and parameterization methods. It allows to reduce demonstration effort by providing contextual information that is not contained in the demonstration data itself. Vice versa, demonstration data is used to extract information for skill parameters that is not modeled in the ontology. Another advantage of the framework integration is that a user can also benefit from supportive tools that the framework offers, for instance, automatic workspace analysis and layout planning [213], or automatic commissioning of missing parts.

Factors influencing the choice of the teaching modality (PbD or OSP) include the level of detail in modeling the robotic system and its surroundings, user preferences and competence, and task-specific requirements. Combining PbD with the proposed skill recognition approach empowers users to select between two different teaching modalities, which leverages their personal abilities more effectively.

9.6 Future Research Directions

The identified skills in Chapter 4 shall provide behaviors that solve contact-based tasks. The focus of the identification procedure was to prove their interpretability and discriminability from the human and robot perspective. A number of limitations of the conducted study could be addressed with the following recommendations. First, an analysis could be conducted using the same identification procedure for robot skills of other domains. Although manipulation skills such as "pick" and "place" are frequently used, it might not be obvious for a user to term the placement of an object into another as "place" or as "insert". Second, the considered skills focus all on contact-based task but it is not analyzed to which extent they are complete. Therefore, a more exhaustive study of contact-based tasks, assembly problems, and geometric mating might shed more light on this topic and lead to a more complete contact skill palette. Third, the analysis has been made in English language and could consider other languages as well. Fourth, the same skill identification process could be repeated with pictograms that serve as graphical skill labels, since it is known that humans are able to quickly familiarize with visual cues. Instead of a manually conducted user study, a Large Language Model (LLM) in combination with an image or video captioning network could also extract skill labels from visual observations. However, it might be hard to quantify the legibility of such labels without any user study.

The strength of the proposed skill recognition algorithms can be played out when an intelligent and robust behavior is embedded in each of the recognized skills. The dexterity of the skill implementations is not in the focus of this work, and the motivation lies in connecting the recognized skills with existing approaches that the research community has already developed. If this connection is not made, recent developments are hardly usable by end-users since they require too much expertise to be deployed in the desired applications. Although there were tremendous efforts in developing dexterous robot behaviors, they are seldom deployed in production systems. One reason could be the researcher's lack of awareness of how their approaches can be used in systems, that are actually programmed by end-users with little experience in robotics. Especially the parameterization of existing approaches from human demonstrations could be a promising direction, where the user provides the task demonstration, and the approach extracts automatically the necessary information to parameterize itself. This also follows the trend to not only derive motor primitives but extracting as much as contextual and high-level knowledge as possible from the demonstration [217].

This work introduces different teaching modalities and demonstration data sources but clearly puts the main emphasis on kinesthetic teaching. This is a largely adopted technique as it simplifies many problems that other data sources such as visual perception have. For instance does it overcome the correspondence problem by using the same kinematic embodiment for teaching and execution. Further, it gives immediate and tangible feedback about the applied forces and the kinematic constraints. Nevertheless, it is less intuitive than using the own hands for demonstrating a new task [26] because there is no additional inertia that the human must handle. It is also known that PbD has its limitations when it comes to abstract constraints that can hardly be transferred via demonstrations, such as controller parameters.

Considering LLMs as programming input channel, they could easily process text or speech-based prompts to generate a sequence of skills that would solve a task goal. However, the generated skill sequence would still lack the required parameters to solve the task, for instance, pick and place locations, desired forces, velocities, and control parameters, meaning that the symbol grounding problem remains to be solved. Only if the implementation of each required skill would contain enough intelligence to extract all this information autonomously from the world, human demonstrations could be substituted by other input channels on the long term.

Generally, the user should be offered a combination of teaching modalities that open different channels of interaction to enhance the knowledge of the system, while taking into account the user's personal preferences and strengths. This goes in line with the aims of the 5th industrial revolution, where beside strong digitalization and automation efforts, the human still plays a central role in future production environments.

Bibliography

[1] G. Clark, J. G. D. Clark, et al. *World prehistory: in new perspective*. Cambridge University Press, 1977.

[2] S. J. Shettleworth. *Cognition, evolution, and behavior*. Oxford university press, 2009.

[3] M. Skubic and R. A. Volz. Learning force sensory patterns and skills from human demonstration. In *Proceedings of International Conference on Robotics and Automation*, pages 284–290. IEEE, 1997.

[4] B. Brunner, K. Arbter, G. Hirzinger, and R. Koeppe. Programming robots via learning by showing in a virtual environment. *Virtual Reality World*, 95:63–72, 1995.

[5] C. G. Atkeson and S. Schaal. Robot learning from demonstration. In *ICML*, volume 97, pages 12–20, 1997.

[6] A. Albu-Schäffer, S. Haddadin, C. Ott, A. Stemmer, T. Wimböck, and G. Hirzinger. The dlr lightweight robot: design and control concepts for robots in human environments. *Industrial Robot: an international journal*, 34(5):376–385, 2007.

[7] Y. Mollard, T. Munzer, A. Baisero, M. Toussaint, and M. Lopes. Robot programming from demonstration, feedback and transfer. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1825–1831. IEEE, 2015.

[8] J. A. Adams. Historical review and appraisal of research on the learning, retention, and transfer of human motor skills. *Psychological bulletin*, 101(1):41, 1987.

[9] R. N. Singer. Motor skills and learning strategies. In *Learning strategies*, pages 79–106. Elsevier, 1978.

[10] R. M. Hulteen, P. J. Morgan, L. M. Barnett, D. F. Stodden, and D. R. Lubans. Development of foundational movement skills: A conceptual model for physical activity across the lifespan. *Sports Medicine*, 48(7):1533–1540, 2018-07-01. URL: `https://doi.org/10.1007/s40279-018-0892-6`, `doi:10.1007/s40279-018-0892-6`.

[11] P. Zech, E. Renaudo, S. Haller, X. Zhang, and J. Piater. Action representations in robotics: A taxonomy and systematic classification. *The International Journal of Robotics Research*, 38(5):518–562, 2019.

[12] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters. Towards learning hierarchical skills for multi-phase manipulation tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1503–1510. IEEE, 2015.

[13] M. Pantano, T. Eiband, and D. Lee. Capability-based frameworks for industrial robot skills: a survey. In *IEEE International Conference on Automation Science and Engineering (CASE)*, pages 2355–2362, 2022.

[14] P. M. Schäfer, F. Steinmetz, S. Schneyer, T. Bachmann, T. Eiband, F. S. Lay, A. Padalkar, C. Sürig, F. Stulp, and K. Nottensteiner. Flexible robotic assembly based on ontological representation of tasks, skills, and resources. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 18, pages 702–706, 2021.

[15] D. Bruckner, M.-P. Stănică, R. Blair, S. Schriegel, S. Kehrer, M. Seewald, and T. Sauter. An introduction to opc ua tsn for industrial communication systems. *Proceedings of the IEEE*, 107(6):1121–1131, 2019.

[16] N. Mansfeld, F. Beck, A. Dietrich, and S. Haddadin. Interactive null space control for intuitively interpretable reconfiguration of redundant manipulators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5368–5375. IEEE, 2017.

[17] 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*, 2(1):56–81, 2013.

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

[19] C. Ott, R. Mukherjee, and Y. Nakamura. Unified impedance and admittance control. In *2010 IEEE International Conference on Robotics and Automation*, pages 554–561, 2010-05. ISSN: 1050-4729. doi:10.1109/ROBOT.2010.5509861.

[20] A. Q. Keemink, H. van der Kooij, and A. H. Stienen. Admittance control for physical human–robot interaction. *The International Journal of Robotics Research*, 37(11):1421–1444, 2018. URL: https://doi.org/10.1177/0278364918768950, doi:10.1177/0278364918768950.

[21] N. Hogan. Impedance control: An approach to manipulation. In *1984 American Control Conference*, pages 304–313, 1984. doi:10.23919/ACC.1984.4788393.

[22] F. Franzel, T. Eiband, and D. Lee. Detection of collaboration and collision events during contact task execution. In *Humanoid Robots (Humanoids), IEEE-RAS 20th International Conference on*. IEEE, 2021.

[23] S. Li and D. Lee. Point-to-pose voting based hand pose estimation using residual permutation equivariant layer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11927–11936, 2019.

[24] J. Zhang, J. Jiao, M. Chen, L. Qu, X. Xu, and Q. Yang. A hand pose tracking benchmark from stereo matching. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 982–986, 2017. ISSN: 2381-8549. doi:10.1109/ICIP.2017.8296428.

[25] S. Yuan, G. Garcia-Hernando, B. Stenger, G. Moon, J. Y. Chang, K. M. Lee, P. Molchanov, J. Kautz, S. Honari, and L. Ge. Depth-based 3d hand pose estimation: From current achievements to future goals. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2636–2645, 2018.

[26] Z. Qiu, T. Eiband, S. Li, and D. Lee. Hand pose-based task learning from visual observations with semantic skill extraction. In *29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 596–603. IEEE, 2020.

[27] M. Nowak, T. Eiband, and C. Castellini. Multi-modal myocontrol: testing combined force-and electromyography. In *International Conference on Rehabilitation Robotics (ICORR)*, pages 1364–1368. IEEE, 2017.

[28] C. Nissler, N. Mouriki, and C. Castellini. Optical Myography: Detecting Finger Movements by Looking at the Forearm. *Frontiers in Neurorobotics*, 10, 2016. URL: https://www.frontiersin.org/articles/10.3389/fnbot.2016.00003.

[29] Giusti, Zeestraten, Icer, Pereira, Caldwell, Calinon, and Althoff. Flexible automation driven by demonstration: Leveraging strategies that simplify robotics. *IEEE Robotics & Automation Magazine*, 2018.

[30] National Aeronautics and Space Administration (NASA). TRL definitions. *Technical Report*, 2019. URL: https://www.nasa.gov/pdf/458490main_TRL_Definitions.pdf.

[31] European Union. Horizon 2020 - work programme 2014-2015. *Technical Report*, 2022-07-04. URL: https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf.

[32] A. Kramberger, A. Gams, B. Nemec, D. Chrysostomou, O. Madsen, and A. Ude. Generalization of orientation trajectories and force-torque profiles for robotic assembly. *Robotics and Autonomous Systems*, 98:333–346, 2017.

[33] F. J. Abu-Dakka, B. Nemec, J. A. Jørgensen, T. R. Savarimuthu, N. Krüger, and A. Ude. Adaptation of manipulation skills in physical contact with the environment to reference force profiles. *Autonomous Robots*, 39(2):199–217, 2015.

[34] A. Montebelli, F. Steinmetz, and V. Kyrki. On handing down our tools to robots: Single-phase kinesthetic teaching for dynamic in-contact tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5634. IEEE, 2015.

[35] V. Koropouli, D. Lee, and S. Hirche. Learning interaction control policies by demonstration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 344–349. IEEE, 2011.

[36] L. Rozo, P. Jiménez, and C. Torras. Force-based robot learning of pouring skills using parametric hidden markov models. In *9th Workshop on Robot Motion and Control (RoMoCo)*, pages 227–232. IEEE, 2013.

[37] M. Racca, J. Pajarinen, A. Montebelli, and V. Kyrki. Learning in-contact control strategies from demonstration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 688–695. IEEE, 2016.

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

[39] L. Peternel, T. Petrič, and J. Babič. Robotic assembly solution by human-in-the-loop teaching method based on real-time stiffness modulation. *Autonomous Robots*, pages 1–17, 2017.

[40] A. M. Schmidts, D. Lee, and A. Peer. Imitation learning of human grasping skills from motion and force data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1002–1007. IEEE, 2011.

[41] S. Stelter, G. Bartels, and M. Beetz. Multidimensional time-series shapelets reliably detect and classify contact events in force measurements of wiping actions. *IEEE Robotics and Automation Letters*, 3(1):320–327, 2018.

[42] T. Eiband, M. Saveriano, and D. Lee. Learning haptic exploration schemes for adaptive task execution. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7048–7054. IEEE, 2019.

[43] A. Stolt, M. Linderoth, A. Robertsson, and R. Johansson. Detection of contact force transients in robotic assembly. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 962–968. IEEE, 2015.

[44] F. J. Abu-Dakka, L. Rozo, and D. G. Caldwell. Force-based learning of variable impedance skills for robotic manipulation. In *IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–9. IEEE, 2018.

[45] L. Rozo, S. Calinon, D. Caldwell, P. Jiménez, and C. Torras. Learning collaborative impedance-based robot behaviors. *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 1(1):1, 2013.

[46] J. R. Medina, M. Lawitzky, A. Mörtl, D. Lee, and S. Hirche. An experience-driven robotic assistant acquiring human knowledge to improve haptic cooperation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2416–2422. IEEE, 2011.

[47] L. Rozo, S. Calinon, D. G. Caldwell, P. Jimenez, and C. Torras. Learning physical collaborative robot behaviors from human demonstrations. *IEEE Transactions on Robotics*, 32(3):513–527, 2016.

[48] R. H. Andersen, T. Solund, and J. Hallam. Definition and Initial Case-Based Evaluation of Hardware-Independent Robot Skills for Industrial Robotic Co-Workers. In *ISR/Robotik 2014; 41st International Symposium on Robotics*, pages 1–7, 2014.

[49] C. Lesire, D. Doose, and C. Grand. Formalization of Robot Skills with Descriptive and Operational Models. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7227–7232, October 2020. ISSN: 2153-0866. doi:10.1109/IROS45743.2020.9340698.

[50] H. Rafii-Tari, C. J. Payne, J. Liu, C. Riga, C. Bicknell, and G.-Z. Yang. Towards automated surgical skill evaluation of endovascular catheterization tasks based on force and motion signatures. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1789–1794. IEEE, 2015.

[51] J. D. Morrow and P. K. Khosla. Manipulation task primitives for composing robot skills. In *Proceedings of International Conference on Robotics and Automation*, volume 4, pages 3354–3359. IEEE, 1997.

[52] C. Rao, A. Yilmaz, and M. Shah. View-invariant representation and recognition of actions. *International Journal of Computer Vision*, 50(2):203, 2002.

[53] S. Wu and Y. F. Li. On signature invariants for effective motion trajectory recognition. *The International Journal of Robotics Research*, 27(8):895–917, 2008.

[54] M. Saveriano and D. Lee. Invariant representation for user independent motion recognition. In *2013 IEEE RO-MAN*, pages 650–655, August 2013. ISSN: 1944-9437. doi:10.1109/ROMAN.2013.6628422.

[55] Y. Guo, Y. Li, and Z. Shao. Rrv: A spatiotemporal descriptor for rigid body motion recognition. *IEEE Transactions on Cybernetics*, 48(5):1513–1525, 2017.

[56] D. Lee, R. Soloperto, and M. Saveriano. Bidirectional invariant representation of rigid body motions and its application to gesture recognition and reproduction. *Autonomous Robots*, 42(1):125–145, 2018.

[57] F. Despinoy, D. Bouget, G. Forestier, C. Penet, N. Zemiti, P. Poignet, and P. Jannin. Unsupervised trajectory segmentation for surgical gesture recognition in robotic training. *IEEE Transactions on Biomedical Engineering*, 63(6):1280–1291, 2016.

[58] 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*, 34(2):131–157, 2015.

[59] S. Krishnan, A. Garg, S. Patil, C. Lea, G. Hager, P. Abbeel, and K. Goldberg. Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. *The International Journal of Robotics Research*, 36(13-14):1595–1618, 2017. URL: https://doi.org/10.1177/0278364917743319, doi:10.1177/0278364917743319.

[60] C. Song, G. Liu, X. Zhang, X. Zang, C. Xu, and J. Zhao. Robot complex motion learning based on unsupervised trajectory segmentation and movement primitives. *ISA Transactions*, 97:325–335, 2020. URL: https://www.sciencedirect.com/science/article/pii/S001905781930343X, doi:10.1016/j.isatra.2019.08.007.

[61] N. Ahmidi, L. Tao, S. Sefati, Y. Gao, C. Lea, B. Bejar, L. Zappella, S. Khudanpur, R. Vidal, and G. D. Hager. A dataset and benchmarks for segmentation and recognition of gestures in robotic surgery. *IEEE Transactions on Biomedical Engineering*, 2017.

[62] D. Lee, C. Ott, and Y. Nakamura. Mimetic communication model with compliant physical contact in human—humanoid interaction. *The International Journal of Robotics Research*, 29(13):1684–1704, 2010.

[63] M. Chi, Y. Yao, Y. Liu, Y. Teng, and M. Zhong. Learning motion primitives from demonstration. *Advances in Mechanical Engineering*, 9(12):1687814017737260, 2017. URL: https://doi.org/10.1177/1687814017737260, doi:10.1177/1687814017737260.

[64] S. Dong and B. Williams. Learning and recognition of hybrid manipulation motions in variable environments using probabilistic flow tubes. *International Journal of Social Robotics*, 4(4):357–368, 2012.

[65] D. Paulius and Y. Sun. A survey of knowledge representation in service robotics. *Robotics and Autonomous Systems*, 118:13–30, 2019.

[66] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, A. Barrett, and D. Christianson. Pddl - the planning domain definition language. Technical report, Yale University, 1998.

[67] K. Ramirez-Amaro, Y. Yang, and G. Cheng. A survey on semantic-based methods for the understanding of human movements. *Robotics and Autonomous Systems*, 119:31–50, 2019. doi:10.1016/j.robot.2019.05.013.

[68] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager. A framework for end-user instruction of a robot assistant for manufacturing. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6167–6174. IEEE, 2015.

[69] 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)*, pages 564–571. IEEE, 2017.

[70] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bøgh, V. Krüger, and O. Madsen. Robot skills for manufacturing: From concept to industrial deployment. *Robotics and Computer-Integrated Manufacturing*, 37:282–291, 2016.

[71] M. N. Nicolescu and M. J. Mataric. Learning and interacting in human-robot domains. *IEEE Transactions on Systems, man, and Cybernetics-part A: Systems and Humans*, 31(5):419–430, 2001.

[72] F. Steinmetz, V. Nitsch, and F. Stulp. Intuitive task-level programming by demonstration through semantic skill recognition. *IEEE Robotics and Automation Letters*, 4(4):3742–3749, Oct 2019. doi:10.1109/LRA.2019.2928782.

[73] M. Wächter, S. Schulz, T. Asfour, E. Aksoy, F. Wörgötter, and R. Dillmann. Action sequence reproduction based on automatic segmentation and object-action complexes. In *13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 189–195, 2013.

[74] X. Wang. Learning planning operators by observation and practice. In K. J. Hammond, editor, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, University of Chicago, Chicago, Illinois, USA, June 13-15, 1994*, pages 335–340. AAAI, 1994. URL: http://www.aaai.org/Library/AIPS/1994/aips94-057.php.

[75] N. Abdo, H. Kretzschmar, L. Spinello, and C. Stachniss. Learning manipulation actions from a few demonstrations. In *2013 IEEE International Conference on Robotics and Automation*, pages 1268–1275, 2013. ISSN: 1050-4729. doi:10.1109/ICRA.2013.6630734.

[76] M. Diehl, C. Paxton, and K. Ramirez-Amaro. Automated generation of robotic planning domains from observations. *arXiv preprint arXiv:2105.13604*, 2021.

[77] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura. Incremental learning of full body motion primitives and their sequencing through human motion observation. *The International Journal of Robotics Research*, 31(3):330–345, 2012.

[78] I. Dianov, K. Ramirez-Amaro, P. Lanillos, E. Dean-Leon, F. Bergner, and G. Cheng. Extracting general task structures to accelerate the learning of new tasks. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 802–807. IEEE, 2016.

[79] K. Ramirez-Amaro, M. Beetz, and G. Cheng. Transferring skills to humanoid robots by extracting semantic representations from observations of human activities. *Artificial Intelligence*, 247:95–118, 2017.

[80] A. L. Pais, K. Umezawa, Y. Nakamura, and A. Billard. Task parameterization using continuous constraints extracted from human demonstrations. *IEEE Transactions on Robotics*, 31(6):1458–1471, 2015.

[81] A. Fern, J. M. Siskind, and R. Givan. Learning temporal, relational, force-dynamic event definitions from video. In *AAAI/IAAI*, pages 159–166, 2002.

[82] Y. Wang, Y. Jiao, R. Xiong, H. Yu, J. Zhang, and Y. Liu. Masd: A multimodal assembly skill decoding system for robot programming by demonstration. *IEEE Transactions on Automation Science and Engineering*, 15(4):1722–1734, 2018.

[83] Z. Su, O. Kroemer, G. E. Loeb, G. S. Sukhatme, and S. Schaal. Learning manipulation graphs from demonstrations using multimodal sensory signals. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2758–2765. IEEE, 2018.

[84] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5239–5246. IEEE, 2012.

[85] E. C. Grigore and B. Scassellati. Discovering Action Primitive Granularity from Human Motion for Human-Robot Collaboration. In *Robotics: Science and Systems*, pages –, 2017. doi:10.15607/RSS.2017.XIII.065.

[86] C. Willibald and D. Lee. Multi-level task learning based on intention and constraint inference for autonomous robotic manipulation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7688–7695, 2022.

[87] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

[88] D. A. Winter, A. J. Fuglevand, and S. E. Archer. Crosstalk in surface electromyography: Theoretical and practical estimates. *Journal of Electromyography and Kinesiology*, 4(1):15–26, 1994. doi:10.1016/1050-6411(94)90023-X.

[89] M. Nowak, T. Eiband, E. R. Ramírez, and C. Castellini. Action interference in simultaneous and proportional myocontrol: Comparing force- and electromyography. *Journal of Neural Engineering*, 2020. URL: http://iopscience.iop.org/10.1088/1741-2552/ab7b1e.

[90] A. Stolt, M. Linderoth, A. Robertsson, and R. Johansson. Force controlled robotic assembly without a force sensor. In *IEEE International Conference on Robotics and Automation*, pages 1538–1543. IEEE, 2012.

[91] L. Rozo, P. Jiménez, and C. Torras. A robot learning from demonstration framework to perform force-based manipulation tasks. *Intelligent service robotics*, 6(1):33–51, 2013.

[92] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal. Online movement adaptation based on previous sensor experiences. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 365–371. IEEE, 2011.

[93] O. Kroemer, C. H. Lampert, and J. Peters. Learning dynamic tactile sensing with robust vision-based training. *IEEE Transactions on Robotics*, 27(3):545–557, 2011.

[94] O. Kroemer, H. Van Hoof, G. Neumann, and J. Peters. Learning to predict phases of manipulation tasks as hidden states. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4009–4014. IEEE, 2014.

[95] T. M. Hagos, M. Suomalainen, and V. Kyrki. Segmenting and sequencing of compliant motions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[96] M. Kaboli, K. Yao, D. Feng, and G. Cheng. Tactile-based active object discrimination and target object search in an unknown workspace. *Autonomous Robots*, pages 1–30, 2018.

[97] G. De Chambrier and A. Billard. Learning search polices from humans in a partially observable context. *Robotics and Biomimetics*, 1(1):8, 2014.

[98] Y. Chebotar, O. Kroemer, and J. Peters. Learning robot tactile sensing for object manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3368–3375. IEEE, 2014.

[99] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal. Towards associative skill memories. In *12th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 309–315. IEEE, 2012.

[100] P. A. Schmidt, E. Maël, and R. P. Würtz. A sensor for dynamic tactile information with applications in human–robot interaction and object exploration. *Robotics and Autonomous Systems*, 54(12):1005–1014, 2006.

[101] Z. Pezzementi, E. Plaku, C. Reyda, and G. D. Hager. Tactile-object recognition from appearance information. *IEEE Transactions on Robotics*, 27(3):473–487, 2011.

[102] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of robot motion: theory, algorithms, and implementation.* MIT press, 2005.

[103] P. Falco, R. Jäkel, C. Natale, and R. Dillmann. Improvement of human hand motion observation by exploiting contact force measurements. In *Humanoids*, pages 141–146, 2011.

[104] J. Tegin and J. Wikander. Tactile sensing in intelligent robotic manipulation–a review. *Industrial Robot: An International Journal*, 32(1):64–70, 2005.

[105] L. Righetti, M. Kalakrishnan, P. Pastor, J. Binney, J. Kelly, R. C. Voorhies, G. S. Sukhatme, and S. Schaal. An autonomous manipulation system based on force control and optimization. *Autonomous Robots*, 36(1-2):11–30, 2014.

[106] R. Zöllner, M. Pardowitz, S. Knoop, and R. Dillmann. Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1535–1540. IEEE, 2005.

[107] D. Silver and J. Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[108] M. Hanheide, M. Göbelbecker, G. S. Horn, A. Pronobis, K. Sjöö, A. Aydemir, P. Jensfelt, C. Gretton, R. Dearden, M. Janicek, et al. Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence*, 247:119–150, 2017.

[109] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12(4):566–580, 1996. URL: https://doi.org/10.1109/70.508439, doi:10.1109/70.508439.

[110] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. *Department of Computer Science*, 1998.

[111] T. Abbas and B. A. MacDonald. Generalizing topological task graphs from multiple symbolic demonstrations in programming by demonstration (pbd) processes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3816–3821. IEEE, 2011.

[112] N. Abdo, H. Kretzschmar, and C. Stachniss. From low-level trajectory demonstrations to symbolic actions for planning. In *ICAPS Workshop on Combining Task and Motion Planning for Real-World App*, pages 29–36, 2012.

[113] S. R. Ahmadzadeh, A. Paikan, F. Mastrogiovanni, L. Natale, P. Kormushev, and D. G. Caldwell. Learning symbolic representations of actions from human demonstrations. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3801–3808. IEEE, 2015.

[114] R. Caccavale, M. Saveriano, A. Finzi, and D. Lee. Kinesthetic teaching and attentional supervision of structured tasks in human-robot interaction. *Autonomous Robots (AURO)*, 2017.

[115] M. L. Visinsky, J. R. Cavallaro, and I. D. Walker. Robot fault detection and fault tolerance: a survey. *Reliability Engineering and System Safety*, 46(2):139–158, 1994.

[116] B. R. Donald. A geometric approach to error detection and recovery for robot motion planning with uncertainty. *Artificial Intelligence*, 37(1-3):223–271, 1988.

[117] A. Rodriguez, D. Bourne, M. Mason, G. F. Rossano, and J. Wang. Failure detection in assembly: Force signature analysis. In *Automation Science and Engineering (CASE), IEEE Conference on*, pages 210–215. IEEE, 2010.

[118] G. E. Hovland and B. J. McCarragher. Hidden markov models as a process monitor in robotic assembly. *The International Journal of Robotics Research*, 17(2):153–168, 1998.

[119] E. Di Lello, M. Klotzbucher, T. De Laet, and H. Bruyninckx. Bayesian time-series models for continuous fault detection and recognition in industrial robotic tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5827–5833. IEEE, 2013.

[120] D. Park, Z. Erickson, T. Bhattacharjee, and C. C. Kemp. Multimodal execution monitoring for anomaly detection during robot manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 407–414. IEEE, 2016.

[121] R. Lin, E. Khalastchi, and G. A. Kaminka. Detecting anomalies in unmanned vehicles using the mahalanobis distance. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3038–3044. IEEE, 2010.

[122] E. Khalastchi, M. Kalech, G. A. Kaminka, and R. Lin. Online data-driven anomaly detection in autonomous robots. *Knowledge and Information Systems*, 43(3):657–688, 2015.

[123] E. Khalastchi and M. Kalech. A sensor-based approach for fault detection and diagnosis for robotic systems. *Autonomous Robots*, 42(6):1231–1248, 2018.

[124] A. Nakamura, K. Nagata, K. Harada, N. Yamanobe, T. Tsuji, T. Foissotte, and Y. Kawai. Error recovery using task stratification and error classification for manipulation robots in various fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3535–3542. IEEE, 2013.

[125] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal. Skill learning and task outcome prediction for manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3828–3834. IEEE, 2011.

[126] D. H. Grollman and A. Billard. Donut as i do: Learning from failed demonstrations. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3804–3809. IEEE, 2011.

[127] T. W. Liao. Clustering of time series data - a survey. *Pattern recognition*, 38(11):1857–1874, 2005.

[128] H. Izakian, W. Pedrycz, and I. Jamal. Fuzzy clustering of time series data using dynamic time warping distance. *Engineering Applications of Artificial Intelligence*, 39:235–244, 2015.

[129] C. Biernacki, G. Celeux, and G. Govaert. Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE transactions on pattern analysis and machine intelligence*, 22(7):719–725, 2000.

[130] C. Li and G. Biswas. Temporal pattern generation using hidden markov model based unsupervised classification. In *Advances in Intelligent Data Analysis: Third International Symposium, IDA-99 Amsterdam, The Netherlands, August 9–11, 1999 Proceedings 3*, pages 245–256. Springer, 1999.

[131] M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, and E. Keogh. Generalizing dtw to the multi-dimensional case requires an adaptive approach. *Data mining and knowledge discovery*, 31(1):1–31, 2017.

[132] J. Aleotti and S. Caselli. Trajectory clustering and stochastic approximation for robot programming by demonstration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1029–1034, 2005.

[133] D. Kappler, P. Pastor, M. Kalakrishnan, M. Wüthrich, and S. Schaal. Data-driven online decision making for autonomous manipulation. In *Robotics: Science and Systems*, 2015.

[134] L. Sauer, D. Henrich, and W. Martens. Towards intuitive robot programming using finite state automata. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 290–298. Springer, 2019.

[135] S. Niekum, S. Chitta, A. G. Barto, B. Marthi, and S. Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9. Berlin, Germany, 2013.

[136] Z. Materna, M. Kapinus, V. Beran, P. Smrž, and P. Zemčík. Interactive spatial augmented reality in collaborative robot programming: User experience evaluation. In *27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 80–87. IEEE, 2018.

[137] F. Steinmetz, A. Wollschläger, and R. Weitschat. Razer—a hri for visual task-level programming and intuitive skill parameterization. *IEEE Robotics and Automation Letters*, 3(3):1362–1369, 2018.

[138] S. Haddadin, S. Parusel, L. Johannsmeier, S. Golz, S. Gabl, F. Walch, M. Sabaghian, C. Jähne, L. Hausperger, and S. Haddadin. The Franka Emika Robot: A Reference Platform for Robotics Research and Education. *IEEE Robotics & Automation Magazine*, 29(2):46–64, June 2022. doi:10.1109/MRA.2021.3138382.

[139] S. Pieskä, J. Kaarela, and J. Mäkelä. Simulation and programming experiences of collaborative robots for small-scale manufacturing. In *2018 2nd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS)*, pages 1–4, April 2018. doi:10.1109/SIMS.2018.8355303.

[140] H. Nguyen, M. Ciocarlie, K. Hsiao, and C. C. Kemp. ROS commander (ROSCo): Behavior creation for home robots. In *2013 IEEE International Conference on Robotics and Automation*, pages 467–474, May 2013. ISSN: 1050-4729. doi:10.1109/ICRA.2013.6630616.

[141] S. G. Brunner, F. Steinmetz, R. Belder, and A. Dömel. Rafcon: A graphical tool for engineering complex, robotic tasks. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3283–3290, 2016.

[142] S. Manschitz, M. Gienger, J. Kober, and J. Peters. Learning sequential force interaction skills. *Robotics*, 9(2):45, 2020.

[143] M. Mühlig, M. Gienger, S. Hellbach, J. J. Steil, and C. Goerick. Task-level imitation learning using variance-based movement optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1177–1184. IEEE, 2009.

[144] M. Mühlig, M. Gienger, J. J. Steil, and C. Goerick. Automatic selection of task spaces for imitation learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4996–5002. IEEE, 2009.

[145] X. Gao, J. Ling, X. Xiao, and M. Li. Learning force-relevant skills from human demonstration. *Complexity*, 2019, 2019.

[146] S. Profanter, A. Breitkreuz, M. Rickert, and A. Knoll. A hardware-agnostic opc ua skill model for robot manipulators and tools. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1061–1068. IEEE, 2019.

[147] O. Heimann and J. Guhl. Industrial Robot Programming Methods: A Scoping Review. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 696–703, September 2020. ISSN: 1946-0759. doi:10.1109/ETFA46521.2020.9211997.

[148] V. K. Origanti, T. Eiband, and D. Lee. Automatic parameterization of motion and force controlled robot skills. In *Robot Intelligence Technology and Applications 6*, pages 66–78. Springer International Publishing, 2022.

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

[150] A. Pervez and D. Lee. Learning task-parameterized dynamic movement primitives using mixture of gmms. *Intelligent Service Robotics*, 11(1):61–78, 2018.

[151] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud. Learning compact parameterized skills with a single regression. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 417–422, October 2013. ISSN: 2164-0580. doi:10.1109/HUMANOIDS.2013.7030008.

[152] T. Matsubara, S.-H. Hyon, and J. Morimoto. Learning parametric dynamic movement primitives from multiple demonstrations. *Neural Networks*, 24(5):493–500, 2011. URL: https://www.sciencedirect.com/science/article/pii/S0893608011000566, doi:10.1016/j.neunet.2011.02.004.

[153] 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)*, 37(2):286–298, 2007.

[154] T. Eiband and D. Lee. Identification of common force-based robot skills from the human and robot perspective. In *IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pages 507–513. IEEE, 2021.

[155] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. Learning the semantics of object–action relations by observation. *The International Journal of Robotics Research*, 30(10):1229–1249, 2011.

[156] L. Talmy. Force dynamics in language and cognition. *Cognitive science*, 12(1):49–100, 1988.

[157] W. De Mulder. Force dynamics. In *The Oxford handbook of cognitive linguistics*. Oxford University Press, 2007.

[158] R. S. Jackendoff. *Semantic structures*, volume 18. MIT press, 1990.

[159] R. Jackendoff. Conceptual semantics and cognitive linguistics. *Cognitive Linguistics*, 7(1), 1996.

[160] J. Xiao and L. Zhang. Contact constraint analysis and determination of geometrically valid contact formations from possible contact primitives. *IEEE Transactions on Robotics and Automation*, 13(3):456–466, 1997. doi:10.1109/70.585907.

[161] H. Bruyninckx and J. De Schutter. Specification of force-controlled actions in the "task frame formalism"-a synthesis. *IEEE transactions on robotics and automation*, 12(4):581–589, 1996.

[162] M. Skubic and R. A. Volz. Acquiring robust, force-based assembly skills from human demonstration. *IEEE Transactions on Robotics and Automation*, 16(6):772–781, 2000.

[163] S. Luo, J. Bimbo, R. Dahiya, and H. Liu. Robotic tactile perception of object properties: A review. *Mechatronics*, 48:54–67, 2017.

[164] J. A. Marvel, R. Bostelman, and J. Falco. Multi-Robot Assembly Strategies and Metrics. *ACM Computing Surveys*, 51(1):14:1–14:32, 2018. URL: https://doi.org/10.1145/3150225, doi:10.1145/3150225.

[165] L. Wang, B. Schmidt, M. Givehchi, and G. Adamson. Robotic assembly planning and control with enhanced adaptability through function blocks. *The International Journal of Advanced Manufacturing Technology*, 77(1):705–715, March 2015. URL: https://doi.org/10.1007/s00170-014-6468-1, doi:10.1007/s00170-014-6468-1.

[166] S. S. M. Salehian and A. Billard. A dynamical-system-based approach for controlling robotic manipulators during noncontact/contact transitions. *IEEE Robotics and Automation Letters*, 3(4):2738–2745, 2018.

[167] C. S. Yitaek Kim and A. Kramberger. A framework for transferring surface finishing skills to new surface geometries. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.

[168] I. Iturrate, E. H. Østergaard, M. Rytter, and T. R. Savarimuthu. Learning and correcting robot trajectory keypoints from a single demonstration. In *3rd International Conference on Control, Automation and Robotics (ICCAR)*, pages 52–59. IEEE, 2017.

[169] F.-Y. Hsu and L.-C. Fu. Intelligent robot deburring using adaptive fuzzy hybrid position/force control. *IEEE Transactions on Robotics and Automation*, 16(4):325–335, 2000.

[170] S. Ahmad and C. N. Lee. Shape recovery from robot contour-tracking with force feedback. *Advanced Robotics*, apr 2012. URL: https://www.tandfonline.com/doi/abs/10.1163/156855391X00197, doi:10.1163/156855391X00197.

[171] J. Norberto Pires, G. Afonso, and N. Estrela. Force control experiments for industrial applications: a test case using an industrial deburring example. *Assembly Automation*, 27(2):148–156, January 2007. URL: https://doi.org/10.1108/01445150710733414, doi:10.1108/01445150710733414.

[172] A. Winkler and J. Suchý. Force controlled contour following on unknown objects with an industrial robot. In *2013 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pages 208–213, October 2013. doi:10.1109/ROSE.2013.6698444.

[173] F. J. Abu-Dakka, L. Rozo, and D. G. Caldwell. Force-based variable impedance learning for robotic manipulation. *Robotics and Autonomous Systems*, 109:156–167, 2018.

[174] H. A. Yanco, A. Norton, W. Ober, D. Shane, A. Skinner, and J. Vice. Analysis of Human-robot Interaction at the DARPA Robotics Challenge Trials. *Journal of*

Field Robotics, 32(3):420–444, 2015. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21568, doi:10.1002/rob.21568.

[175] K. Kronander, E. Burdet, and A. Billard. Task transfer via collaborative manipulation for insertion assembly. In *Workshop on Human-Robot Interaction for Industrial Manufacturing, Robotics, Science and Systems*, pages –, 2014.

[176] J. Xu, Z. Hou, Z. Liu, and H. Qiao. Compare contact model-based control and contact model-free learning: A survey of robotic peg-in-hole assembly strategies. arXiv, 2019. arXiv:1904.05240.

[177] M. Suomalainen, Y. Karayiannidis, and V. Kyrki. A survey of robot manipulation in contact. *Robotics and Autonomous Systems*, 156:104224, 2022.

[178] S. Haddadin, M. Suppa, S. Fuchs, T. Bodenmüller, A. Albu-Schäffer, and G. Hirzinger. Towards the robotic co-worker. In *Robotics Research*, pages 261–282. Springer, 2011.

[179] J. Stüber, C. Zito, and R. Stolkin. Let's push things forward: A survey on robot pushing. *Frontiers in Robotics and AI*, 7:8, 2020.

[180] C. Fellbaum. Wordnet. *The encyclopedia of applied linguistics*, 2012.

[181] D. Chicco, M. J. Warrens, and G. Jurman. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. *PeerJ Computer Science*, 7:e623, jul 2021. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8279135/, doi:10.7717/peerj-cs.623.

[182] B. C. Ross. Mutual information between discrete and continuous data sets. *PloS one*, 9(2), 2014.

[183] C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[184] T. Eiband, J. Liebl, C. Willibald, and D. Lee. Online task segmentation by merging symbolic and data-driven skill recognition during kinesthetic teaching. *Robotics and Autonomous Systems*, 162:104367, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0921889023000064, doi:https://doi.org/10.1016/j.robot.2023.104367.

[185] S. Calinon and D. Lee. Learning control. In P. Vadakkepat and A. Goswami, editors, *Humanoid Robotics: a Reference*. Springer, 2018.

[186] M. Cakmak and A. L. Thomaz. Eliciting good teaching from humans for machine learners. *Artificial Intelligence*, 217:198–215, 2014.

[187] N. Koenig, L. Takayama, and M. Matarić. Communication and knowledge sharing in human–robot interaction and learning from demonstration. *Neural Networks*, 23(8-9):1104–1112, 2010.

[188] A. Sena and M. Howard. Quantifying teaching behavior in robot learning from demonstration. *The International Journal of Robotics Research*, 39(1):54–72, 2020.

[189] R. Sheh. "Why did you do that?" Explainable intelligent robots. In *AAAI Workshop-Technical Report*, pages 628–634, 2017.

[190] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77, 2018.

[191] J. F.-S. Lin, M. Karg, and D. Kulić. Movement primitive segmentation for human motion modeling: A framework for analysis. *IEEE Transactions on Human-Machine Systems*, 46(3):325–339, 2016.

[192] D. A. Grant. The latin square principle in the design and analysis of psychological experiments. *Psychological bulletin*, 45(5):427, 1948.

[193] T. Eiband, C. Willibald, I. Tannert, B. Weber, and D. Lee. Collaborative programming of robotic task decisions and recovery behaviors. *Autonomous Robots*, pages 1–19, 2022.

[194] S. J. Lederman and R. L. Klatzky. Haptic perception: A tutorial. *Attention, Perception, & Psychophysics*, 71(7):1439–1459, 2009.

[195] G. Robles-De-La-Torre. The importance of the sense of touch in virtual and real environments. *Ieee Multimedia*, 13(3):24–30, 2006.

[196] M. O. Ernst and M. S. Banks. Humans integrate visual and haptic information in a statistically optimal fashion. *Nature*, 415(6870):429, 2002.

[197] G. Metta and P. Fitzpatrick. Better vision through manipulation. *Adaptive Behavior*, 11(2):109–128, 2003.

[198] Y. Hatwell, A. Streri, and E. Gentaz. *Touching for knowing: cognitive psychology of haptic manual perception*, volume 53. John Benjamins Publishing, 2003.

[199] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[200] F. L. Markley, Y. Cheng, J. L. Crassidis, and Y. Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4):1193–1197, 2007.

[201] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, 2013.

[202] A. Ude, B. Nemec, T. Petrić, and J. Morimoto. Orientation in cartesian space dynamic movement primitives. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2997–3004. IEEE, 2014.

[203] T. Eiband, M. Saveriano, and D. Lee. Intuitive programming of conditional tasks by demonstration of multiple solutions. *IEEE Robotics and Automation Letters*, 4(4):4483–4490, Oct 2019. doi:10.1109/LRA.2019.2935381.

[204] C. Willibald, T. Eiband, and D. Lee. Collaborative programming of conditional robot tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5402–5409. IEEE, 2020.

[205] A. Pervez and D. Lee. Learning task-parameterized dynamic movement primitives using mixture of gmms. *Intelligent Service Robotics*, pages 1–18, 2017.

[206] S. Calinon. Robot learning with task-parameterized generative models. In *Robotics Research*, pages 111–126. Springer, 2018.

[207] G. Maeda, M. Ewerton, T. Osa, B. Busch, and J. Peters. Active incremental learning of robot movement primitives. In *Conference on Robot Learning*, pages 37–46, 2017.

[208] S. S. Khan and M. G. Madden. A survey of recent trends in one class classification. In *Irish conference on artificial intelligence and cognitive science*, pages 188–197. Springer, 2009.

[209] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

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

[211] A. Naumann and J. Hurtienne. Benchmarks for intuitive interaction with mobile devices. In *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '10, page 401–402, New York, NY, USA, 2010. Association for Computing Machinery. URL: https://doi.org/10.1145/1851600.1851685, doi:10.1145/1851600.1851685.

[212] T. Eiband, F. Lay, K. Nottensteiner, and D. Lee. Unifying skill-based programming and programming by demonstration through ontologies. *Procedia Computer Science*, 232:595–605, 2024. 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023). URL: https://www.sciencedirect.com/science/article/pii/S1877050924000590, doi:https://doi.org/10.1016/j.procs.2024.01.059.

[213] T. Bachmann, O. Eiberger, T. Eiband, F. Lay, P. Angsuratanawech, I. Rodriguez, P. Lehner, F. Stulp, and K. Nottensteiner. Task-specific reconfiguration of variable

workstations using automated planning of workcell layouts. In *56th International Symposium on Robotics*, pages 250–257, 2023.

[214] IEEE Robtics and Automation Society. IEEE standard ontologies for robotics and automation. *IEEE Stan.*, 1872:1–60, 2015. doi:10.1109/IEEESTD.2015.7084073.

[215] S. R. Fiorini, A. Chibani, T. Haidegger, J. L. Carbonera, C. Schlenoff, J. Malec, E. Prestes, P. Gonçalves, S. V. Ragavan, and H. Li. Standard Ontologies and HRI. In *Human–Robot Interaction*, pages 19–47. Chapman and Hall/CRC, 2019.

[216] M. Iskandar, C. Ott, O. Eiberger, M. Keppler, A. Albu-Schäffer, and A. Dietrich. Joint-Level Control of the DLR Lightweight Robot SARA. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8903–8910, October 2020. ISSN: 2153-0866. doi:10.1109/IROS45743.2020.9340700.

[217] M. Tavassoli, S. Katyara, M. Pozzi, N. Deshpande, D. G. Caldwell, and D. Prattichizzo. Learning Skills from Demonstrations: A Trend from Motion Primitives to Experience Abstraction. *IEEE Transactions on Cognitive and Developmental Systems*, pages 1–1, 2023. doi:10.1109/TCDS.2023.3296166.