Interactive Learning of Probabilistic Decision Making by Service Robots with Multiple Skill Domains

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Sven R. Schmidt-Rohr

aus Heidelberg

Tag der mündlichen Prüfung: 6.7.2012
Erster Gutachter: Prof. Dr.-Ing. Rüdiger Dillmann
Zweiter Gutachter: Prof. Michael Beetz, PhD

Ich versichere wahrheitsgemäß, die Dissertation bis auf die dort angegebenen Hilfen selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und als kenntlich gemacht zu haben, was aus Arbeiten anderer und eigenen Veröffentlichungen unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, Mai 2012

Preface

Service robots are destined to revolutionize both the tertiary sector as well as households, profoundly changing society. Yet necessary autonomous behavior, orchestrating highly diverse physical interaction with complex environments, is far from being technically solved. True autonomy requires a technical system to be able to continuously select actions from a wide range of choices in an ever changing world. Such action selection is a decision-making procedure capable of dealing with incomplete knowledge about its environment as well as potential future courses of events.

This thesis makes a contribution towards deeper understanding of that challenge by presenting a decision making system design approach. The contribution is centered around two aspects. The first is modeling decision making in the face of incomplete information on top of diverse basic skills of a service robot. Second, based on such a model, it is investigated extensively, how to transfer complex decision-making knowledge into the system. Interactive learning, naturally from both demonstrations of human teachers and in interaction with objects, is the chosen paradigm, yielding decision-making models applicable by the robot. Investigated methods are implemented as actual software and fully integrated, leading to experimental evaluation on physical service robots.

Hopefully, the insight gathered through this thesis can support further research, one day leading to intelligent robots with the ability of sophisticated autonomous decision making, learning naturally like children from human teachers and independent exploration of the environment.

Karlsruhe,
2012

Sven R. Schmidt-Rohr
Karlsruhe Institute of Technology

Zusammenfassung

Die vorliegende Arbeit untersucht autonomes Entscheiden durch Serviceroboter mit verschiedenartigen Fähigkeitsdomänen wie Mobilität, Objektmanipulation und natürlicher Mensch-Roboter Interaktion. Dabei liegt der Fokus auf dem Zusammenspiel der Fähigkeiten und entsprechend abwägendem Entscheiden auf strategischer Ebene. Um die in jeder realen Umgebung vorherrschende Unvollständigkeit von Information zu berücksichtigen, sowohl bezüglich des gegenwärtigen Zustands der Umwelt, als auch möglicher zukünftiger Vorgänge, werden probabilistische Entscheidungsverfahren eingesetzt. Diese sind in der Lage die Unsicherheiten explizit zu repräsentieren und quantitativ abzuwägen.

Die Herausforderung ist, Robotermissionen, welche hochgradig unterschiedliche Fähigkeiten beinhalten, für solche probabilistischen Entscheidungsverfahren abzubilden. In der vorliegenden Arbeit wurde eine entsprechende Entscheidungsarchitektur entwickelt. Darüber hinaus ist die Akquise des nötigen Entscheidungswissens für beliebige Missionen durch den Roboter eine noch größere Herausforderung. Daher wird ein Prozess vorgestellt, in welchem dieses Entscheidungswissen aus einem Zusammenspiel von Lernen durch menschliche Lehrer, auch Programmieren durch Vormachen genannt und weitere Verfeinerungsanalysen erworben wird.

Dieses Vorgehen verspricht den Vorgang des Erwerbs von komplexem Planungswissen durch den Roboter sowohl praktikabel, also auch flexibel zu gestalten. Das vorgestellte System wurde komplett als Software auf physischen Servicerobotern realisiert, integriert und in diversen Experimenten evaluiert.

Im Folgenden werden Einsichten, Details und Beschränkungen des Konzepts diskutiert.

Acknowledgments

First and foremost, I would like to thank my parents for teaching me some fundamental values of life: commitment, initiative, candor and persistence. Many thanks to my siblings Axel, Klaus, Ute and Volker for instilling me with intellectual curiosity from the youngest age on and leading me through rocks, sand, mud, snow and mountains of LEGO bricks to find my destiny in being a "builder".

Many thanks to Prof. Rüdiger Dillmann for his support and providing me with so many opportunities in growing and learning during building, managing and teaching in a very inspiring environment. Many thanks to Prof. Michael Beetz for interesting discussions.

My deep gratitude goes to Rainer Jäkel for endless inspiration, first as my student and later as my colleague. It has been my honor and pleasure to work together with such a brilliant scientist, always reliable and helpful. I am very grateful to my student of many years, Gerhard Dirschl. Without his endless effort and tinkering, the huge infrastructure around a fully autonomous Albert robot would have been too much to build up alone. Countless thanks go to my colleague Martin Lösch, standing with me from the first minute to the last, keeping the flag of the team raised high in rough winds. My deep gratitude also goes to Steffen Knoop, my mentor of many years, for planting a seed in the old times and caring for others. Many thanks to Pascal Meissner, my student and later colleague of many years for making Albert see the world of objects. And many thanks to my colleague Alexander Kasper for making such an ugly robot look so nice in CG.

My gratitude goes to all my students, who assisted in writing three hundred thousand lines of code, many hardware tweaks and countless experiments: Fabian Romahn, Jonas Stahl, Laurenz Berger, Thorsten Mai, Christian Wischnewski, Gerhard Kurz, Tim Friedrich, Tobias Utz, Johannes Pelzer, Ghassen Megrehi, Martin Seidel, Reno Reckling, Jessica Kaufmann, Gerald Baier, Marcel Krüger, Matthias Mayr, Manuel Leuschner, Jochen Schäfer, Yaokun Zhang, Alex Yan, Michael Schütz, Stefan Maier and Werner Walter.

Thanks to the Adero robot keepers, Zhixing Xue, Steffen Rühl and Andreas Herrmann for support and fruitful collaboration in our project. Thanks also to Thilo Kerscher and Peter Steinhaus for support when needed, and to the cognitive automobile colleagues down the hallway, Tobias Gindele and Sebastian Brechtel, for countless interesting discussions involving probabilistic

modeling and decision making. Thanks to Stefan Vacek for a nice office companionship in my early days. Many thanks to the administrative assistants Isabelle Wappler, Christine Brand, Nela Redzovic and Diana Kreidler for taking care of so many things, much patience and many kind words!

.... and thanks to Albert the robot for hanging on for all these years, making my childhood dream come true. His retirement is well earned now.

Karlsruhe, 2012 *Sven R. Schmidt-Rohr*

Contents

1. Introduction

Automating manual everyday tasks in human-centered environments is a promising emerging technology, destined to reshape both professional as well as domestic fields profoundly. Among devices delivering such automation, service robots take a particularly versatile role. Service robots are machines that quite closely resemble humans in proportions and basic physical abilities. These abilities include relocation of their whole physical manifestation, physical interaction with objects in their vicinity and interaction with humans by natural means. However, to carry out duties that cannot be automated yet, robots have to gain another human ability: deciding which action to perform next, based on situations and goals.

Simple robots and life forms choose actions based on hardwired responses to environment perception. These responses have been created by a designer or the process of evolution. While this method of decision making may be efficient for simple tasks, it fails when confronted with more complex situations, action options and causal sequences. In these cases, which include performance of everyday tasks in human-centered environments, the autonomous robot needs to carefully consider several aspects. It has to reason about possible effects of its own actions, predict changes in the environment that are potentially only loosely coupled with its own actions and finally has to assess the uncertainty of its own knowledge concerning a certain situation. Eventually, reasoning about causality has to be combined with suitable motivations and goals, which means to perform its duties efficiently and robustly in the case of the service robot. This leads to a system for planning and decision making, a process in the information processing system of the robot with the ability to choose an action at any time during its duties in a way that optimally serves its owner's purpose.

A *mission* is defined as a single, self-contained objective, encompassing all tasks needed to perform a certain duty. Actions can be executed by the service robot by means of several *skill domains*, which are channels used to interact with its environment. The *situation*, the environment configuration at a certain point of time, is assessed by prediction as well as observation of the environment using different sensors which are part of distinct skill domains.

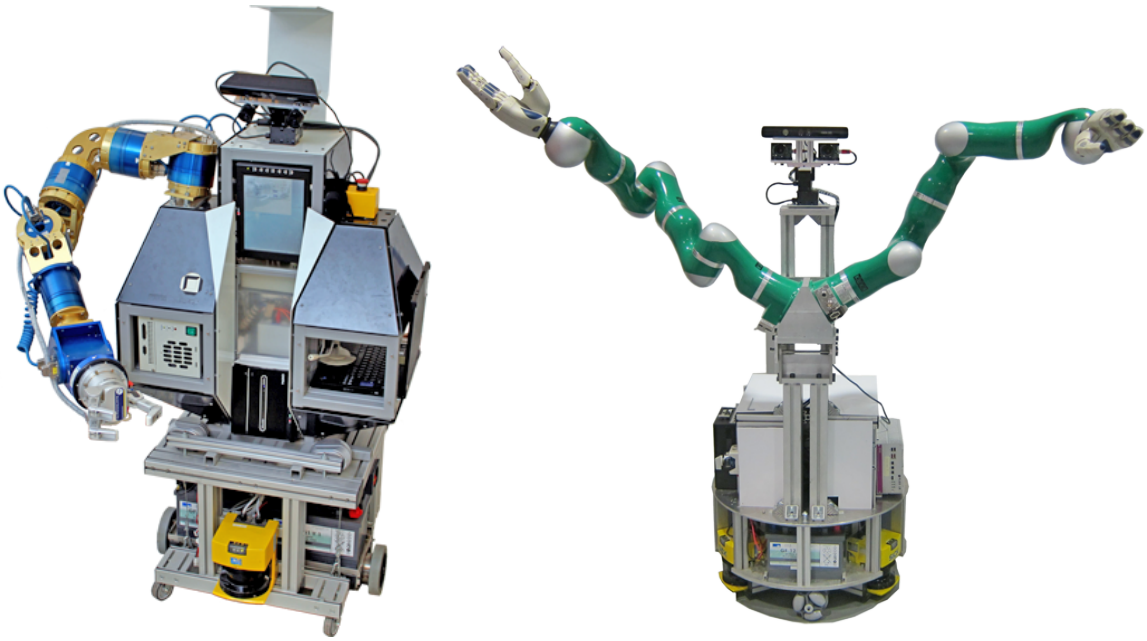


Figure 1.1.: Autonomous mobility, object manipulation and natural HRI service robots **Albert** (left) and **Adero** (right), used as evaluation platforms. - [125]

Typical service robots - as depicted in Figure 1.1 - cover three major skill domains:

- Autonomous mobility: relocating the whole robot within its environment.
- Natural human-robot interaction (HRI): spoken dialog with humans and gestures.
- Autonomous object manipulation: physical interaction with objects using grippers.

Thus, robot decision making has to consider typical dynamics in these quite different domains in a coherent way, leading to decision making for missions considering *multiple skill domains*.

Planning and decision-making techniques applied by artificial systems need formal, quantified representations of relevant aspects of missions. In missions with multiple skill domains, such a representation has to be homogeneous over all domains.

Discretization and abstraction are a way to handle that challenge. Suitable planning techniques for problems covering multiple domains are based on such representations. These representations are planning/systems models - in short *planning models* - suitable to compute decision making by an artificial agent such as a service robot. Yet, essential problems remain:

- How to design abstraction for decision making covering multiple robot skill domains?
- How can a service robot obtain a mission planning model in a feasible manner?

This thesis deals with these closely coupled questions, the former in Chapter 3, the latter in Chapter 4.

1.1. Approach Motivation

When designing planning model abstraction on a robotic system and devising a method to obtain the model, some further specification of the planning technique is necessary. For abstract, mission-level planning and decision making, methods with differing capabilities concerning the characteristics of the environment exist. According to [122], several characteristics can be distinguished, exhibited by the environment of an autonomous, rational agent: static – dynamic, episodic – sequential, deterministic – stochastic, discrete – continuous, fully observable – partially observable, single-agent – multi-agent. Hereby, an environment sufficiently modeled by a property mentioned first can be assumed to be less complex than one described by a property named second. Because the property of an environment classified as stochastic concerning courses of events has a strong impact on the complexity of planning methods, there is a tendency of choosing the deterministic assumption too aggressively.

Within the scope of this work, three basic arguments support the case that it is insufficient to assume that a real service-robot environment covering multiple skill domains is deterministic:

1. Considering courses of events as stochastic arises directly from physical properties of the world: quantum dynamics as well as the relationship between entropy and the arrow of time. The uncertainty principle in quantum dynamics prevents courses of events in any real world system to be deterministically predictable [54]. Yet, even when assuming that this property is attenuated in macroscopic environments like a setting in which a robot acts, the directed character of time and entropy in a physical system remain. These basically mean that causality is not symmetric when looking into the past and towards the future from a given present [84]. The present describes a low-entropy boundary condition with far fewer potential courses of events leading from some potentially unknown past to the present than may lead from the present to some potential - and certainly unknown - future. When modeling exponentially increasing potential courses of events - like future in contrast to past ones - deterministic models become insufficient. Instead, stochastic models can group causal dynamics in ways suitable for planning.

2. Experiments covering robot planning applications often show superior results when they are based on methods accounting directly for stochastic courses of events as discussed in Section 2.2.6. The more complex the setting, the more profound the stochastic nature of courses of events. Based on this data and trend, extrapolation leads to the conclusion that more complex robot missions envisaged in the future are only suitably covered by taking into account stochastic courses of events.

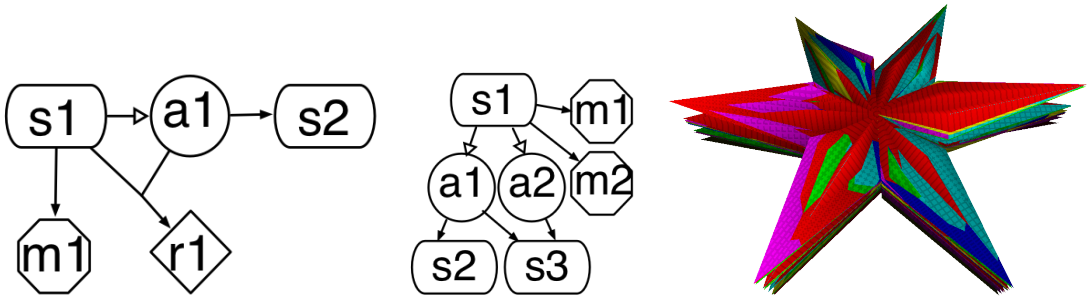


Figure 1.2.: Simplified schematic of POMDP flow of events – discussed in Section 2.2.3 – (left), simplified model scheme (center) and a decision-making policy visualization (right). - [125], [105]

3. Psychology research indicates that abstract human decision making is based on probabilistic representations to account for a stochastic world as discussed in Section 2.2.8. As evolution tends to employ simple methods where sufficient, it is likely that non-probabilistic planning is insufficient to cover abstract decision making in real-world settings.

Because of these points, little further argument will be made comparing stochastic against deterministic modeling alternatives in Chapters 3 and 4. Some experiments discussed in Section 5 include comparisons with deterministic techniques, otherwise the discussion will focus on how to tackle challenging questions arising from stochastic modeling.

Consequently, as the physical environment of a service robot has to be classified as stochastic and partially observable, methods for planning in deterministic and fully observable settings, discussed in Section 2.1.2, are less suitable. Instead, *probabilistic planning and decision-making* methods like *Markov decision processes*, discussed in Section 2.2 and shown schematically in Figure 1.2, can reflect real-world dynamics, especially the stochastic nature, in the planning model and often lead to superior performance under realistic circumstances. Yet, this capability leads to higher computational complexity of the planning process and a more complex model representation.

Hence, defining and creating a model for a certain mission becomes more intricate. An approach is necessary that allows for comfortable transfer of all necessary model parameters into the information processing system of the robot such that the model is then usable for planning. A promising approach for transfer of skill and task information into the information processing system of a robot is *Programming by Demonstration* (PbD), often known as *Learning from Observation* or *Imitation Learning*, discussed more closely in Section 2.4. Under this paradigm, skills or tasks a robot shall acquire, are demonstrated by a human domain expert, performing the task naturally by himself. A technical system records demonstrations with sensors, followed

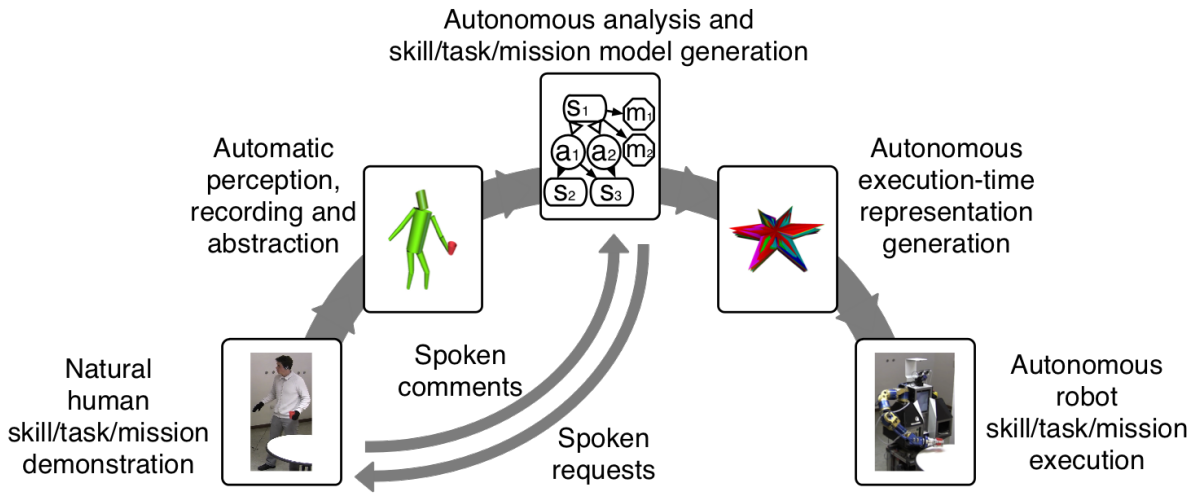


Figure 1.3.: General scheme of a typical PbD process. - [125]

by generation of a task description which can in turn be used by the robot to execute the task. Thus, PbD can be considered as an *interactive learning* technique, integrating different sorts of interactions between human and machine.

The strength of PbD is the comfortable, natural, explicit and implicit transfer of domain knowledge, as shown schematically in Figure 1.3, requiring no expert in robot programming. These obvious benefits suggest investigation of PbD for the generation of models used in probabilistic planning of service robot missions. However, there are also areas where PbD is lacking and the model generation process has to be supported by techniques beyond PbD. This provides all the elements for the investigation to be performed in this thesis:

Interactive learning [programming by demonstration and learning from experience] **of probabilistic decision making** [models] **by service robots considering multiple skill domains** [mobility, human-robot interaction and object manipulation].

1.2. Thesis Statement

Thesis: *Generation of probabilistic planning models for service robot missions in human centered environments is made feasible by analyzing human demonstrations of such missions.*

The statement is supported by a presentation of means to model robot service missions as abstract, probabilistic planning models. Techniques are discussed to generate models from analysis of recorded natural demonstrations, supported by further refinement. Evaluation covers exemplary missions, integrating several skills on real service robots.

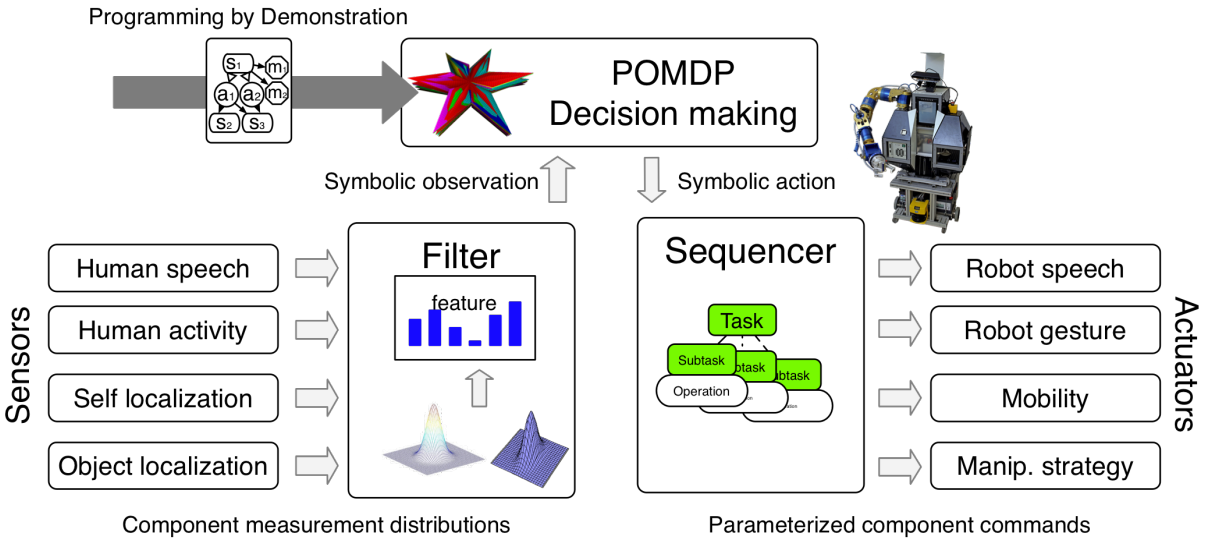


Figure 1.4.: Execution-time robot autonomy architecture as presented in Chapter 3. - [125]

1.3. Document Outline

The document consists of the following main chapters:

- Chapter 2 contains the discussion of the theoretical background and related work.
- Chapter 3 explains an approach to model service robot missions covering multiple skill domains suitable for probabilistic planning.
- Chapter 4 presents a concept to generate these planning models by means of PbD.
- Chapter 5 discusses experimental evaluation.
- Chapter 6 gives conclusions.

1.4. Concept Overview

In Chapter 3, the concept of an **execution-time information processing system** and an approach to define various aspects of a planning model for certain types of missions are presented.

The system architecture for autonomous decision making by a service robot handles execution-time application of probabilistic planning models and resulting action policies. Formal, parameterized models, describing a mission and generated by the PbD concept, as described in Section 4, are utilized by these information processing components of the robot while performing a mission. In this thesis, a typical three-layer **autonomous execution architecture** was developed, as described in Section 3.1. On the lowest layer, algorithmic components processing

sensor measurements into more abstract observations and components generating actuator commands from abstract execution requests, form a collection of skills. An intermediate layer compiles abstract observations into a unified belief-state description, required for decision making. The other part of that intermediate layer generates several component-directed execution requests for each abstract action. On top, probabilistic decision making applies a planning model and chooses abstract actions to be performed by the robot next, based on delivered belief states. All together, these three layers form the rational agent cycle of *perceive – reason – act* [122] generating active behavior while the robot performs a mission as depicted in Figure 1.4.

Available skill domains are provided by components processing sensor measurements or generating actuator commands for abstract actions. To consider aspects of the environment to which these components relate in the planning model, definitions have to be created which map environment aspects onto abstract planning model properties. Two fundamental abstract properties are *state* and *action*. A state is defined as a certain configuration of the world in which an agent is acting. A state encompasses all properties of the world relevant for decision making to distinguish it from other states. A concept to create quantified state descriptions for arbitrary sets of diverse skill domains is explained in Section 3.2.

In partially observable environments, a set of *measurements* does not guarantee to a robot that the world is in a certain state. Instead, imperfect sensor measurements and occlusions limit the robot's knowledge of the current state of the world. Yet, certain observations correlate with certain states with high probability. Such a correlation can be quantified as a likelihood for a state leading to a certain measurement. Probabilistic decision making utilizes that likelihood for both computation of a policy as well as belief state computation, used for policy queries. A measurement likelihood depends on the observed aspects of the world, sensor properties and processing algorithm characteristics. In this thesis, approximate **modeling of uncertainty in robot perception** for various components was investigated. Additionally, an exemplary perception component for localization of furniture was devised, which models uncertainty from raw sensor data to a more abstract measurement result, as explained in Section 3.3.

Decision making as applied in the presented concept selects an abstract action to be performed by the robot next, each time a previous action has finished. However, such an abstract action usually is a compound of more fine-grained actuator commands. To reason about possible effects of such an action during planning and also to execute the action, a representation for the contents of the abstract action is necessary. Thus, a previously developed hierarchical task representation **modeling elementary actions as robot tasks** was integrated into the system as discussed in Section 3.4.

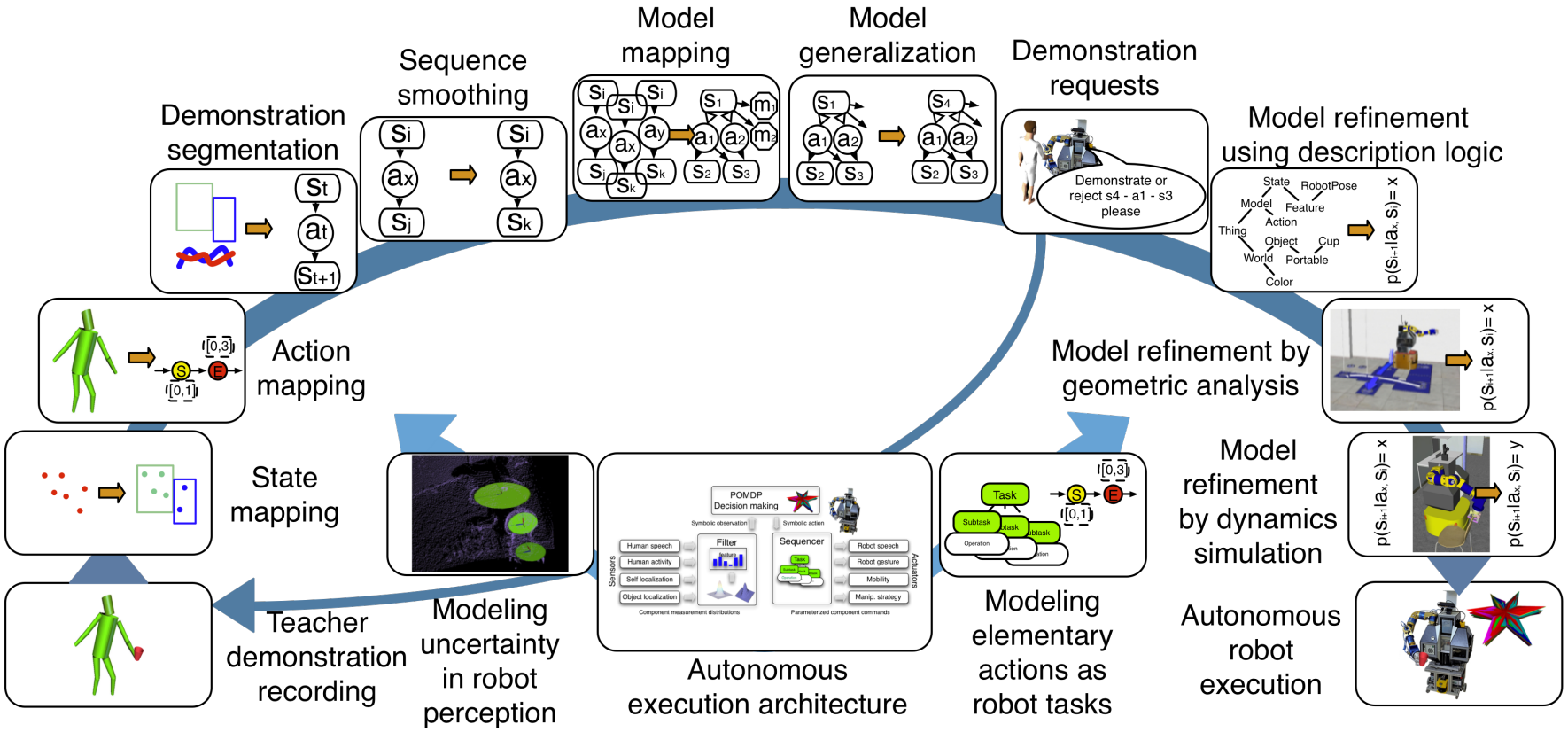


Figure 1.5.: Schematic view of the presented approach: modeling, discussed in Chapter 3, at the center, overarched by the PbD process presented in Chapter 4. - [125]

A mission model represents task knowledge, yet long-term knowledge and information reusable among several missions needs an extra representation. To account for such knowledge and ensure usability in the model generation process, a specialized knowledge base employing *Description Logic* was developed. An ontology contains core aspects of service robot missions in a hierarchical manner, relating tangible objects and intangible model aspects. New information can be added during learning while information can be inferred when generating planning models. The knowledge base is described in Section 3.7.

In Chapter 4, **Programming by Demonstration of probabilistic mission planning models**, a primary contribution of this thesis is discussed in detail. The PbD process, sketched in Figure 1.5, is complementary to the execution-time *perceive – reason – act* agent cycle. First, the robot observes a scene with it sensors. However, in the case of PbD, the setting is special as a natural human demonstration of a mission takes place. Additionally, the robot does not interfere actively in the course of events, but instead records the demonstration. After recording several demonstrations, reasoning takes place which generates an abstract model of the mission. Finally, that model can be applied during execution-time reasoning when the robot performs the mission itself. Thus, learning takes place in a phase distinct from execution and thus can be considered as a type of offline learning. Yet, as the PbD approach involves comments from the human to the robot and also from the robot to the human, the process is tightly interactive.

Observing and **recording human demonstrations** is the first step in a PbD process. In the present work, observation is robot-based, with the robot actively observing the scene including acting human teachers and objects. One exception is spoken dialog, where headsets are used to clearly distinguish different speakers. Body motion of at least one human teacher, the *Robot Role Demonstrating Human* (RR) is measured by a human skeleton motion tracking system utilizing a 3D point cloud depth sensor. On top of motion tracking, a freely available system, developed in the lab, classifies abstract activities of the RR, based on the motion. Additionally, motion activities of a human teacher representing a human interacting with the robot, a *Human Interaction Role Demonstrating Human* (HR), 6D poses of relevant objects in the scene as well as spoken utterances of both RR and HR can be observed. Data of each type is recorded and can be used for further processing after demonstrations. Exemplary setups are shown in Figure 1.6, while details are described in Section 4.1.

Raw data recorded from demonstrations is available in different representations, depending on the skill domain. Some data – such as human poses – is represented by continuous values (coordinates) while other – for instance spoken utterances – is available as discrete symbols. However, recordings have to be segmented and mapped to discrete state – action sequences, the

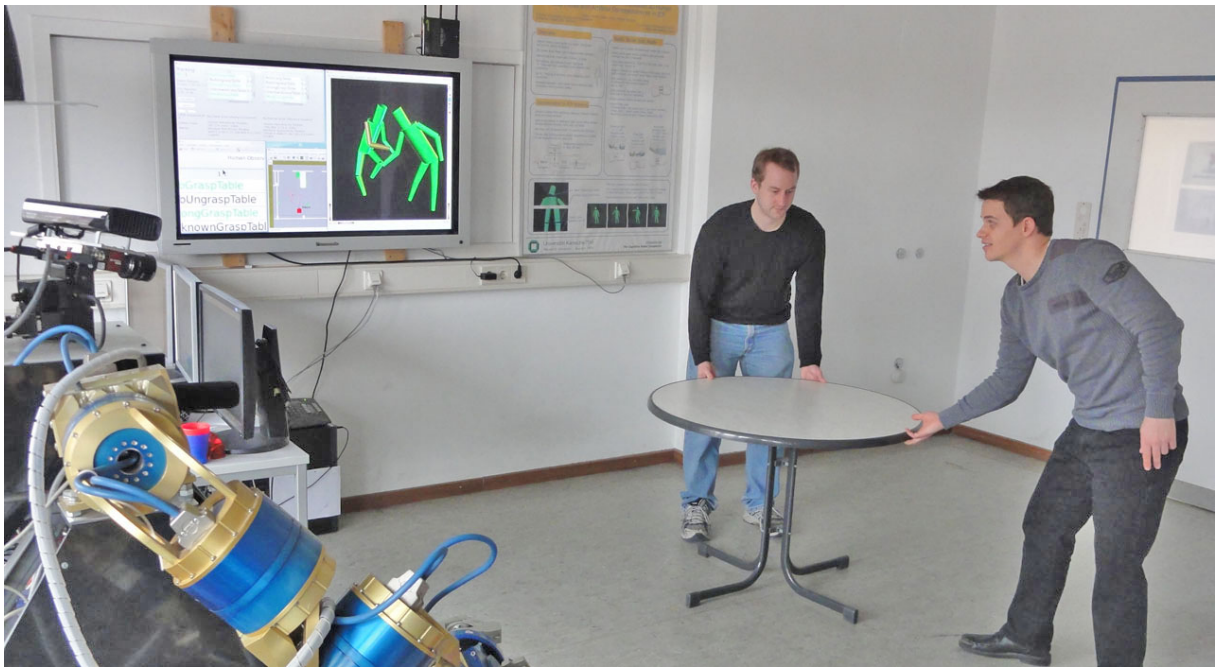


Figure 1.6.: Observation of a natural human mission demonstration by the robot. - [125]

underlying representation of the planning model. To achieve **state mapping**, relating continuous values with discrete state symbols, discretization rules have to be defined. For efficient discretization, optimized for a certain mission and to automate as many aspects of planning model generation as possible, these rules have to be determined automatically. This aim is achieved by applying clustering methods to the recorded data, combined with cross-analysis of values between various skill data as presented in Secion 4.2. As a result, mappings are generated from world configurations as observed by robot perception to abstract state symbols. These mappings can be used for both segmentation of demonstration recordings as well as belief-state computation during mission execution by the robot.

As with state symbols, action symbols have to be mapped to corresponding real-world robot behavior. Especially concerning the domain of object manipulation, specific motion properties of individual human manipulation actions may differ from the robot execution skill motion. Thus, mapping an observed human manipulation activity to a parameterized robot manipulation skill is not trivial. Both a component to classify observed human manipulation activities symbolically and a component to execute manipulation activities in varying scenes, outlined in Section 2.7, have been developed in cooperation with the presented mission PbD concept. For both components, a varying, but finite set of available elementary manipulation activities can be generated. To be of use in the mission PbD process, a correspondence **action mapping** between two such sets has to be generated for each mission. Such a technique, described in detail in Section 4.3, was investigated: it compares the geometric motion structure between

classified human activities and executable robot activities. After application to recorded data, all manipulation action symbols refer to executable robot skills.

Using discretization for skill-domain-specific recorded data, **demonstration segmentation** can transfer each recorded demonstration into a state - action sequence as explained in Section 4.4. These sequences then represent courses of events on the level of abstraction inherent in resulting planning models.

While execution-time perception is considered as partially observable, usually the assumption is made that demonstration-time observation is sufficiently modeled as fully observable. To deal with demonstration setups where observation results show that this assumption cannot hold, **sequence smoothing** of observed, segmented demonstration can deal with some severe observation errors. Unlikely or causally impossible observed action effects can be smoothed by modeling observed demonstration sequences as Hidden Markov Models. Dynamics of the process concerning action effect probabilities and observation likelihoods are determined by a meta-model for a setting, stored in the background knowledge system. Application of the technique to exemplary missions is described in Section 4.5.

A set of segmented demonstration sequences describes typical sequences of events in a certain mission. Relative frequencies of certain states resulting from an action in a specific state – which is the origin state of the resulting action transition – among the set of sequences are an indication for transition probabilities concerning mission specific aspects. Similarly, primary mission goals are indicated by these demonstration sequences. In the analysis process step, a preliminary (PO)MDP model is generated by analyzing the set of demonstration sequences. State and action sets, defining the mission as well as mission-specific transition probabilities and primary goals to be included in the reward model are determined at this point as explained in Section 4.6. However, the information resulting from **model mapping** can only be regarded as a skeleton model because on the one hand, typical sets of demonstrations usually lack information about all possible courses of events, while information specific to robot capabilities is not included at all. Thus, the preliminary model has to be completed by the following steps.

An abstract POMDP model of a mission has to contain all relevant information about probabilistic correlations related to robot actions and observations as well as all costs, goals and constraints corresponding to the mission. On the other hand, a set of human demonstrations containing *some typical* courses of events usually does not cover all courses of events that are hypothetically possible. While a model derived only from an initial set of demonstrations might be sufficient to perform a mission, flexibility in courses of events and thus options to perform the mission can be more limited than necessary. To explore the model space for potential action

results which were not demonstrated, yet could be relevant for decision making considering alternative, suitable course of events, a **model generalization** analysis is applied to the preliminary model. Potential transitions are derived from similar observed ones and rated with confidence values, indicating the likelihood of their occurrence. Details about this process can be found in Section 4.7.

Generalization of observed action effects can only estimate unobserved effects and thus a verification mechanism is mandatory. Such verification has to include the human teacher as a primary source of causal information, yet needs to consider its limited availability. Regarding these aspects, verifying generalization is achieved by relevance analysis of generalized transitions and compiling highly critical effects into potential demonstration sequences. These sequences are then transformed into visual or spoken **demonstration requests** to human teachers which may decline demonstrations as invalid or perform them within the scope of the request. By these means, generalized transition hypotheses can be pruned and the demonstration process can be steered by the system to cover decision-critical aspects of the model. Request generation is discussed in Section 4.8. In summary, generalization and verification by means of further requested demonstrations can together lead to more flexible mission models.

Eventually, a further channel of information can be used to gather knowledge from human demonstrators. While passive observation of human activities - and also dialog between RR and HR teachers - delivers mostly implicit human knowledge about the mission and domain, additional **spoken comments** by teachers may deliver further, explicit action knowledge. Explanatory comments directed towards the observing robot can deliver further descriptive information about states and actions, not easily derived from task observation alone as outlined in Section 4.9. In the scope of the presented system, explicit spoken comments are especially useful for state and action classification during incremental learning of background knowledge as described next. Together, request generation by the robot and human comments during demonstrations can lead to a spoken, bi-directional, interactive PbD process.

Some information about action-effect correlations in a mission cannot be derived from human demonstrations as capabilities of human teacher and service robot may differ in relation to action effect prior probabilities. A class of effects that can be summarized as robot action-execution errors is especially affected. To include information about such effects that do not occur during human demonstrations, comprehensive, mission independent knowledge as stored in the background ontology is used. Therefore, a process component analyzes the actions within the preliminary mission model and infers additional model information from related aspects in the ontology. As a result, **model refinement using description logic** adds information such

as error states, recovery actions, error transitions and action cost penalties to the preliminary model. It is based on description logic inference techniques, explained in Section 4.10.

Robot-specific knowledge concerning action effect prior probabilities as derived by inference on background knowledge is not a precise estimate of real-world dynamics. Particularly in the domains of mobility and manipulation, specific geometric properties and layouts of the scene as well as path- and motion-planning characteristics define these effect probabilities. To improve estimates of these effect prior probabilities and thus improve the model quality in relation to the real world, further analysis of the actions in an observed mission is necessary. In this process step, the geometric setup of demonstration time scenes is used to apply navigation path planning and manipulation motion planning within these scenes. Then, execution uncertainty metrics on the geometric representations can be computed to derive action effect prior probabilities in the abstract planning model. Finally, **model refinement by geometric analysis** updates the abstract planning model with these values as described in Section 4.11.

Further refinement of action effect priors can be achieved by complementing geometric analysis with execution trials in physical dynamics simulation. This approach has been used for the domain of object manipulation. Scene setups as observed during demonstrations are replicated with diverse variations in a simulation environment. Subsequently, the robot executes elementary actions within the simulation, while observation and execution uncertainty is applied. Finally, the effects of the execution within the dynamics simulation are evaluated and relative frequencies computed for the action effect model. More precise representation of real-world dynamics in the mission model is thus achieved. A detailed presentation of the approach of **model refinement by dynamics simulation** is given in Section 4.12.

After these final model refinement steps, the preliminary POMDP model has been transformed into the final planning model on which action selection policy computation will be based. While such a model can never be a perfect representation of real-world dynamics, its information is suitable for the service robot to assess risk and opportunities within a mission as shown in the evaluation, Section 5. It can finally be used to compute a policy, utilized during **autonomous robot execution** of a mission.

In Chapter 5, a detailed evaluation of individual process stages in simulation or the real world is presented. Finally, evaluation of selected missions, utilizing the whole PbD process and based on real observation of physical demonstrations as well as physical world execution on service robots with multiple skill domains underlines the feasibility and portability of the presented

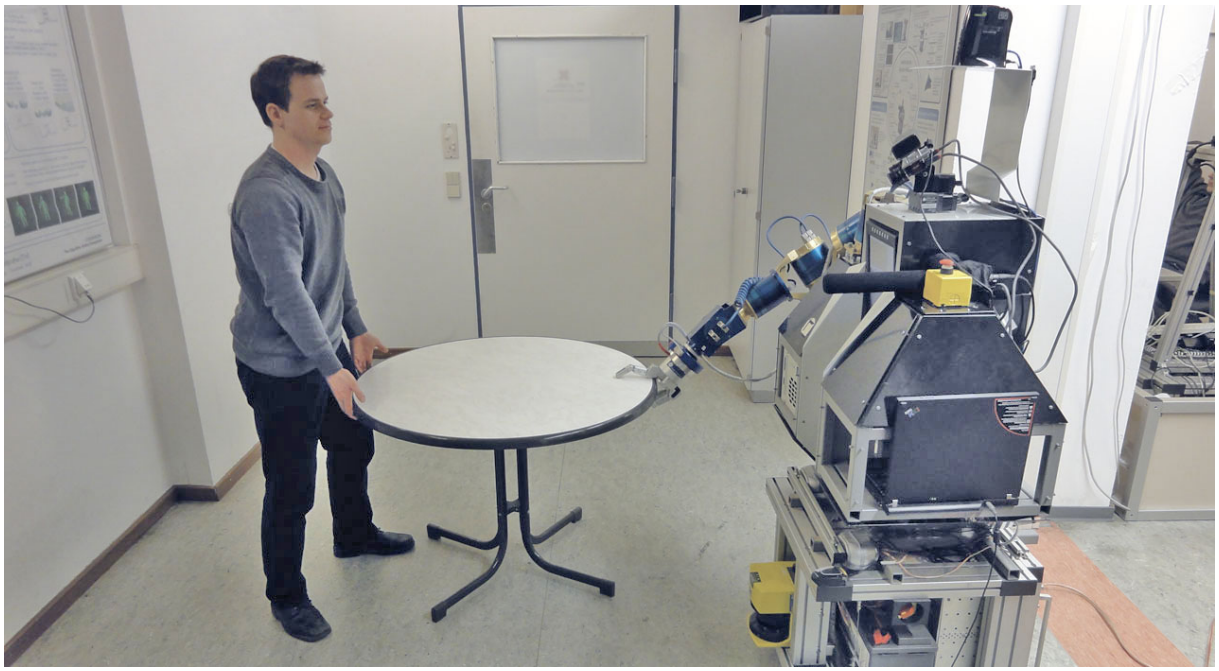


Figure 1.7.: Robot executing a mission as described in Section 5.1.2. - [125]

process. It also shows completely implemented integration of all process stages in software. Exemplary execution is shown in Figure 1.7.

1.5. Thesis Contribution

The contribution of this thesis is to model probabilistic decision making of service robots with multiple skill domains and, most importantly, the presentation of a comprehensive process to generate mission-specific models from observation of human demonstrations. Symbol grounding generation, model structuring with interactive exploration and further refinement by background knowledge as well as geometric analysis work together in a single, coherent process. The thesis presents insights for systems with multiple skill domains, considerations regarding necessary process operations in such a PbD system and describes exemplary techniques of various kinds to perform these operations. It also shows how to bring together paradigms from a wide range of fields in robotics. The whole process is fully integrated and evaluation is shown on physical-world service robot systems with a diverse set of skills.

Thesis contribution: This thesis contributes the concept and system of *Probabilistic Mission Planning Model Programming by Demonstration* (PMPM-PbD).

2. Theoretical Background and Related Work

Several different aspects of autonomous robotics play an important role in the concept developed and presented by this thesis. As indicated by its title, major aspects are *interactive learning*, *probabilistic decision making* and *autonomy for service robots*.

The goal of autonomy, the ability to act flexibly in complex, dynamic environments as motivated in Section 1, is fundamental to modern robotics. Thus, there have been a multitude of research activities in this area, covering basic aspects of the organization of information-processing systems to enable autonomy, irrespective of specific reasoning techniques. Section 2.1 covers these aspects, including alternative reasoning techniques.

Probabilistic decision making, an essential concept utilized in this thesis, has been investigated in a large variety of studies addressing both theoretical foundations as well as applications in robotics. In PMPM-PbD, the framework of Markov Decision Processes has been applied to cover probabilistic decision making by an autonomous robot. Relevant literature dealing with that concept is discussed in Section 2.2.

Moreover, the interactive learning technique Programming by Demonstration has been utilized in the presented concept to comfortably generate models for probabilistic decision making. Programming by Demonstration and imitation learning have been investigated intensively for a wide range of aspects in robotics, especially for autonomous manipulation, but also human-robot interaction. Section 2.4 discusses prominent research and its relation to PMPM-PbD.

Further learning techniques in robotics, related to methods used in the refinement steps of PMPM-PbD are discussed in Section 2.5 and organization of background knowledge is discussed in Section 2.6. Finally, novel robot skill components developed in cooperation with and utilized by PMPM-PbD are presented in detail in Section 2.7.

2.1. Autonomous Behavior by Service Robots

Autonomy of a robot is difficult to quantify, yet has been interpreted in the community as the ability of a robot to adapt its activities to varying and not precisely defined situations. A core principle of autonomy is the ability of a robot to shape its actions by reasoning based on perception of a certain situation and prior knowledge. Depending on the methodology used to attain autonomy, models of a situation and prior knowledge as well as reasoning techniques may

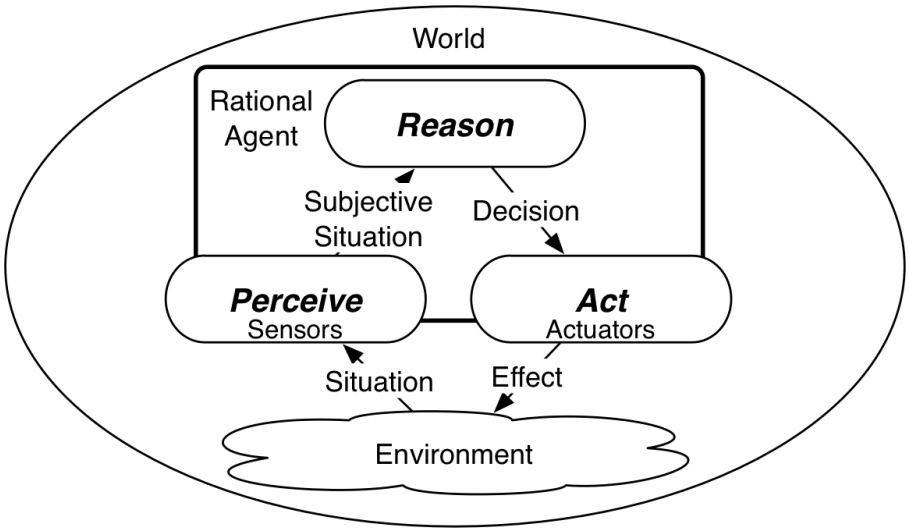


Figure 2.1.: Scheme of the rational agent concept, the foundation of autonomy. - [125]

differ significantly. Additionally, autonomous reasoning can be classified as being either rather "low level" or "high level". The former is usually skill-specific such as navigation path planning or manipulation motion planning while the latter deals with shaping activities globally, which may include selection of a skill domain. As the main focus of this thesis is on the latter, it is discussed in detail in the next sections.

Global control for autonomy can be achieved by following one of several existing paradigms. Inspired by nature, *behavior-based control* imitates reasoning as performed by living animals, coding action as interlinked sets of schemas. It is discussed in Section 2.1.1. Originating from logic in philosophy and based on predicate logic descriptions of the world, reasoning by application of discrete planning as discussed in Section 2.1.2 is a paradigm very distinct from behavior-based control. Reasoning with Markov Decision Processes, discussed in Section 2.2, relaxes the assumptions about the world made by logic-based planning.

Organizing perception of the surrounding world, skill domains, reasoning and low-level actuator control in a complex system like an autonomous robot is not straightforward. Thus, defining the layout of the information processing architecture for an autonomous service robot has been much investigated in literature. Section 2.1.3 presents some architectures designed for requirements similar to the system utilized in PMPM-PbD.

A rational agent [122], sketched in Figure 2.1, is an information processing system observing its environment through sensors. Based on observations, it reasons to perform some actions to achieve a set of goals while keeping to a set of constraints. These actions can then be executed by actuators, which modify the environment. Consequently, the agent performs *perceive – reason – act* cycles.

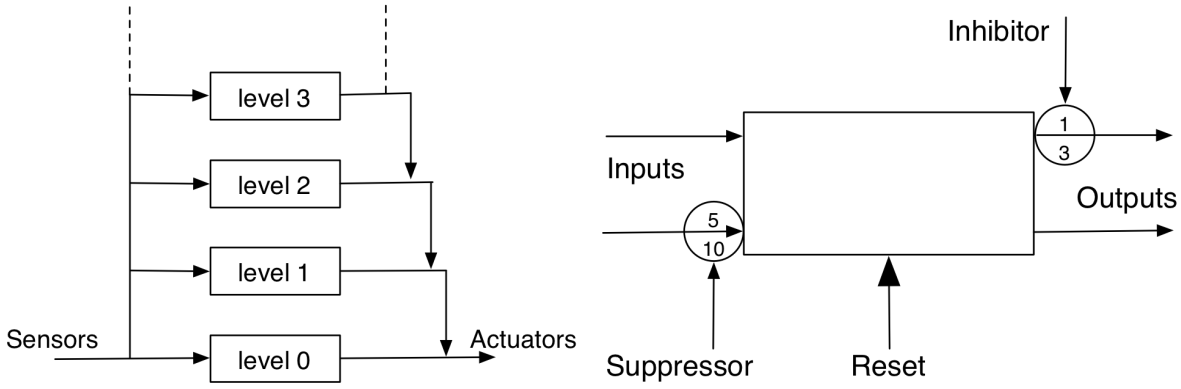


Figure 2.2.: Schematic view of the subsumption architecture (left) and a module (right). - [20]

2.1.1. Behavior-based Control

A distinct paradigm investigated for robot autonomy is behavior-based control. First introduced in [20], reasoning about robot actions based on perception is distributed among specialized, loosely coupled controller components on different levels of abstraction. The intention is to consider multiple goals as well as multiple noisy sensors – and thus perception skill domains – that cannot be easily achieved by straightforward logic-based planning (see Section 2.1.2). The *subsumption architecture* as presented by [20],[21] is a hierarchical reasoning system for an autonomous mobile robot, providing a flexible number of hierarchy layers, shown in Figure 2.2. Autonomy to the extent provided by a layer is self-contained in the system up to that layer level. A layer can subsume – hence the name – a behavior at a level lower into its controller architecture and replace its input signal or inhibit its output signal. By structuring the system this way, each layer forms a complete rational agent together with its lower layers, unaware of, but influenced by the agents on top. Within a layer, behavior design can be quite arbitrary and complex. Modeling individual components within a behavior controller is not clearly defined.

It is argued in [21] that the incremental nature of constructing the system and thus its reasoning capabilities from the ground up, gets rid of the need for a central representation on which reasoning is based. Instead, intelligent behavior arises from individual, manageable behavior blocks and their interactions. Finally, the pure form of behavior-based control argues for no explicit representation, that is a model, of the world that includes actions of the robot. Thus, there are no symbolic tokens representing any aspects in short-term or long-term memory. Hence, without explicit memory and representations, explicit reasoning aboukt the past and the present hardly exists in such a system.

Investigation of details, challenges and practical implications concerning the concept of behavior-based control has been extensive [95]. A key aspect is the design of individual behaviors, where

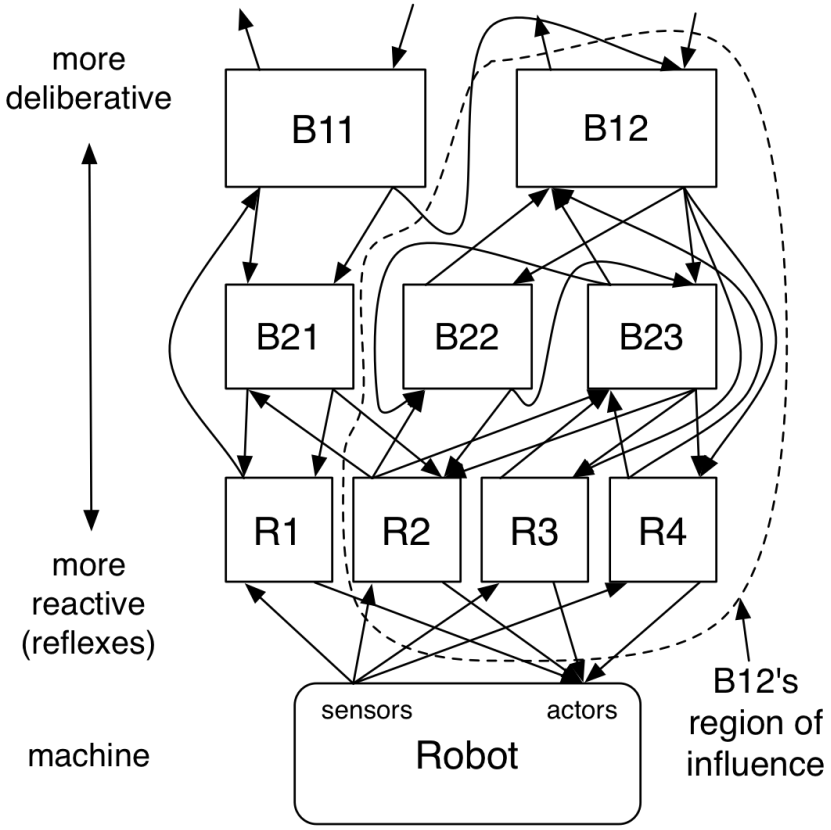


Figure 2.3.: Schematic view of a behavior network. - [3]

preferring behavior interaction with the environment compared to interaction among behaviors, leveraging the strength of the concept, has given best results. Accordingly, however, coordination of behaviors is difficult to manage with rising complexity and number of behaviors. Lacking a common framework for formal description of individual behavior dynamics, options are limited to assess and evaluate behavior interaction analytically. Emergence of new behavior and reasoning capabilities from interaction with the environment can be achieved by learning. Because this capability is so important in robotics, supervised learning and reinforcement learning paradigms have been coupled with behavior-based control. Modifying interaction schemes between behavior modules is a technique to adapt the reasoning capability of a robot. Yet, here again complexity of many interacting behaviors is difficult to manage. Applications of behavior-based control have been manifold. They include biologically inspired navigation for driving, swimming or flying mobile robots, behavior of robot swarms as well as biologically inspired imitation learning.

A clear design method guiding the process of determining layers, individual behaviors and interactions between behaviors is presented for the application domain of an autonomous walking machine in [3]. Here, emphasis is put on the interaction of behaviors with each other and

only reactive reflexes, forming the lowest layer, connect directly to the environment via sensors and actuators. Interaction of high-level, deliberate behaviors with behaviors on lower levels is grouped in so-called regions of influence. By these means, the network gets some basic structure and complexity of cross-interactions is somewhat limited as shown in Figure 2.3. As walking machines are characterized by several similar kinematic structures, the proposed design method is organized based on kinematic blobs, e.g. legs, in a top-down manner. Therefore, the design method is viable, but very application specific.

A more implementation specific presentation of behavior-based control of robot teams is given in [72]. The discussion includes details on how to allocate behaviors in realistic settings and how coordination within a robot team arises from behaviors of individual robots.

Finally, the domain of robot manipulation, quite distinct from autonomous navigation, has been investigated in [39]. The idea is to model behaviors as control-loops that are based on elementary sensor features of the environment, such as contact surfaces of the hand. As with all behavior-based approaches this stands in contrast to heavily model-based planning approaches. High-level behaviors correspond to more abstract manipulation tasks like transferring an object from one hand to the other. These behaviors are composed of an interplay of more basic reflex behaviors which also lead to more robust handling of exceptional situations.

It can be concluded that the viewpoint of behavior-based control gives some important insights regarding autonomy of robotic systems. Balancing multiple, potentially conflicting, goals and considering multiple noisy sensors is important and neglected in pure logic-based planning. Additionally, the introduction of hierarchy and self-sufficient, very fast simple reflexes can increase robustness of autonomous behavior. Yet, some difficulties arise, too. Inhomogeneous modeling of controllers within a certain behavior lacking formal descriptions of their dynamics and complexity of many interacting behaviors make autonomy difficult to manage in realistic settings. Apart from that, there is no explicit knowledge about the world and the robot itself as a guidance to reason explicitly about its past and future. Pure behavior-based control has thus been less investigated in recent times given the availability of more powerful robot systems, but some conceptual elements have remained in modern control architectures.

2.1.2. Logic-Based Planning

In stark contrast to behavior-based control, planning by autonomous robots is heavily based on explicit models of the world. Planning has the aim of deriving actuator commands from reasoning about potential courses of events in the world based on the model and a perceived current or past state as well as aims to be achieved by the robot. Because it has to reason

explicitly about causality in the world, such a planning model needs to cover two essential aspects of the world:

1. A *state* represents a set of similar configurations of relevant properties of the world.
2. An *action* represents a set of actuator activities by the robot.

By reasoning about how actions transform one state of the world into another, a planning technique may find a sequence of actions that leads from one state, e.g. one that reflects a current situation, to a desired state, which represents a goal situation. Thus, for a given model, planning turns into a search problem. However, the type of search depends on the model representation. Two main types of world models for planning can be distinguished:

1. *Continuous* representations of states and actions.
2. *Discrete* representations of states and actions.

Both types of planning are widely employed in robotics, yet typically in different areas of application. Continuous planning is very suitable for skill domain specific tasks like deriving motions for robot navigation or object manipulation. Thereby, physical objects surrounding the robot and the robot itself are modeled by quite precise approximations of their continuous geometry. Resulting actions are represented by continuous motion trajectories that can be implemented by motor commands. Although skill domain specific motion planning is not the main focus of this thesis, more abstract models need to represent the action effects resulting from motion planning components. Motion planning is outlined in the context of manipulation strategies in Section 2.7.

When deriving global, strategic action choices, homogeneous continuous modeling of the world and search within that model quickly becomes infeasible. On the other hand, discrete representations are a way to simplify the model of the world and only focus on relevant aspects for action selection. Discretization usually results in a finite set of states and thus a finite *state space*. This means, all configurations of the world are represented by a finite set of tokens, usually along with a discrete, finite set of actions and a discrete representation of time.

Planning in such discrete world models reduces the search problem to finding one or several potential sequences of actions which may lead from one discrete state of the world to another. Such a search problem can be pictured as searching within a directed graph, where nodes (vertexes) represent states and edges represent actions. Hence, graph-based search techniques like forward, bi-directional and heuristic search can be applied in such discrete planning problems with deterministic action effects.

One way to formulate the semantic connection between such a graph model and the world is logic. One of the first systems for logic-based planning was STRIPS [45]. The discrete set of states of the world is represented by logic expressions consisting of *instances* classifying certain things in the world and *predicates*, assigning to the instances a certain configuration or relation. Actions are represented by a precondition expression that describes certain aspects of a state that have to hold for applying the action. The postcondition expression describes the aspects of a state that are changed by the action. Subsequently, search can retrieve a sequence of actions such that the operations turn a certain start expression into a desired goal expression.

While pure logic-based planning has some severe disadvantages concerning its complexity and its assumptions about the world (see Section 2.2 for a discussion), it is simple and yet powerful for reasoning far into the future, incorporating long sequences of actions. Because of its discrete, symbolic nature it has been investigated for global, strategic planning in robotics. Challenges in robot application of logic-based planning are manifold. Modeling planning domains and generation of specific models is one area of investigation. The representation of continuous aspects of the world concerning both states and actions as well as interplay with reasoning components on lower levels of abstraction, incorporating either continuous motion planning or behavior-based control reasoning are further areas of interest.

Establishing a suitable model for logic-based planning in a robot setting is challenging. Compact expressions for states have to be determined and action operations have to be defined which adequately describe world dynamics. A paradigm to acquire such a model is machine learning. Furthermore, learning can be used to improve the planning process for an existing model. There are several areas in model generation, incremental model enhancement and planning improvement where some types of machine learning can be applied.

Although not evaluated on a real robotic application domain in [159], PRODIGY gives an overview on a wide range of learning approaches suitable for improving both the planning model and planning with the model. Improving the planning efficiency can be accomplished by explanation-based analysis of planning traces, generation of abstraction hierarchies or by deriving analogies. Learning the planning model is less specific to the approach of logic-based planning and encompasses principles also used in PMPM-PbD. Domain knowledge can be acquired by means of *Observation*, *Apprentice* or *Experiment*.

"Observation" analyzes the task execution of an expert agent and records action effects, thus learning action operators. Although the presented technique is tailored for logic-based planning, the fundamental principle of Observation in PRODIGY is learning from observation like performed in PMPM-PbD, discussed in Section 4.6.

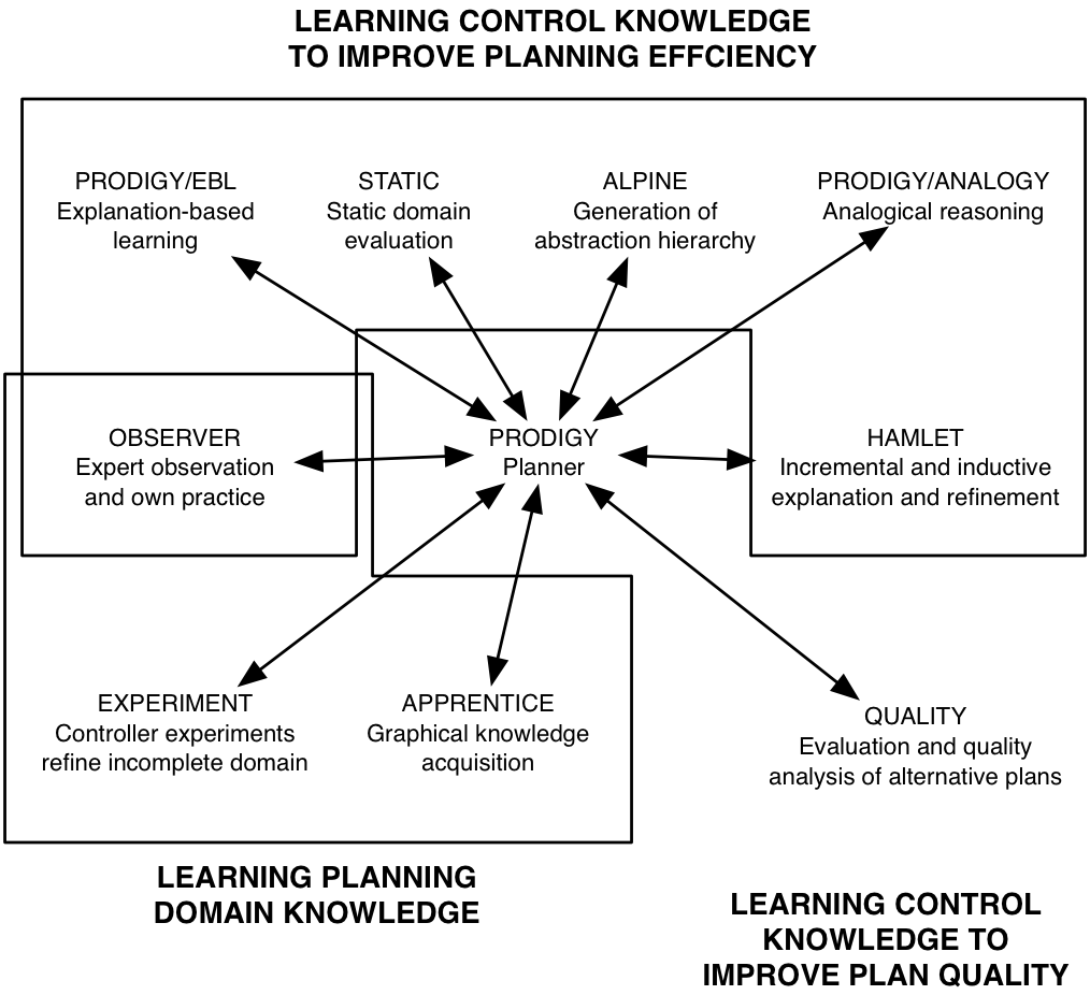


Figure 2.4.: The PRODIGY architecture. - [159]

"Apprentice" in PRODIGY is presented as a graphical interface through which a user can guide both model acquisition and planning itself. The fundamental principle of explicit user hints is also applied in PMPM-PbD with spoken action annotation comments during demonstrations as described in Section 4.9

Finally, "Experiment" in PRODIGY subsumes all types of techniques where the agent evaluates behavior of the real world to retrieve knowledge about action effects. That fundamental principle is also utilized in PMPM-PbD with learning from simulated experience as described in Section 4.12.

In summary, [159] already presents all important, fundamental approaches for an agent to acquire domain knowledge for planning. Yet in that work, implemented techniques are specific to logic-based planning and tested on simple AI problems, rather than real robots. Subsequently, much further robotics-centered research has taken place for learning by observation, apprentice

and experiment as discussed in Sections 2.4 and 2.5.

While model generation and thus learning is one of the big challenges to leverage symbolic planning for robotics, another is to adequately handle planning when facing continuous time and dynamic courses of events as present in real-world environments. In stark contrast to behavior-based control, pure logic-based planning is quite restricted regarding continuous and dynamic flow of events. It assumes discrete time steps leading from one distinct state to another by application of a certain action and plans long sequences of actions by assuming that the causal effects of these actions will be exactly as planned according to the model. Extra effort has to be made to tackle such aspects, which are prevalent in real robotic applications.

Both mentioned characteristics of time and causality are explicitly modeled, for instance using MDPs (see Section 2.2) or extensions to the application of logic-based planning as presented in [19]. According to the technique and results described, the challenges can be tackled by several modifications to STRIPS-like logic-based planning. Foremost, the assumption that an initial planning state has to be completely defined by a boolean expression is relaxed. Instead, unspecified variables in a state expression are allowed. Furthermore, sensor models explicitly specify which aspects of a state can be potentially perceived by knowledge-gathering activities. Subsequently, the planning process can continuously monitor state updates during execution and consider wether to pursue an existing plan or to start replanning to generate a new one.

While logic-based planning has been investigated intensively in the field of intelligent agents, application in complex robotics problems has been limited. Recent research applying logic-based planning for strategic autonomy in robot missions, including autonomous object manipulation has shown reasonable application to be a steep challenge. Representing continuous planning on lower levels of abstraction in logic-based planning expressions and managing the interplay between different ways of handling time and space are essential difficulties.

An idea to combine the continuous motion space of a robot manipulator with the abstract, symbolic domain space suitable for logic-based planning is presented in [27]. Manually designed mapping from logic-based strategic planning to continuous manipulator motion planning as in most other approaches is replaced by automatic generation of axioms for logic-based planning. For each object in a certain workspace, motion planning automatically generates abstract action expressions usable by the strategic planner. A limitation of this approach is the complexity of generating all the action-effect expressions in complex settings.

In an approach presented by [38], the strategic logic-based planner and the motion planner are also tightly coupled, yet the motion planner is queried by the strategic planner on demand

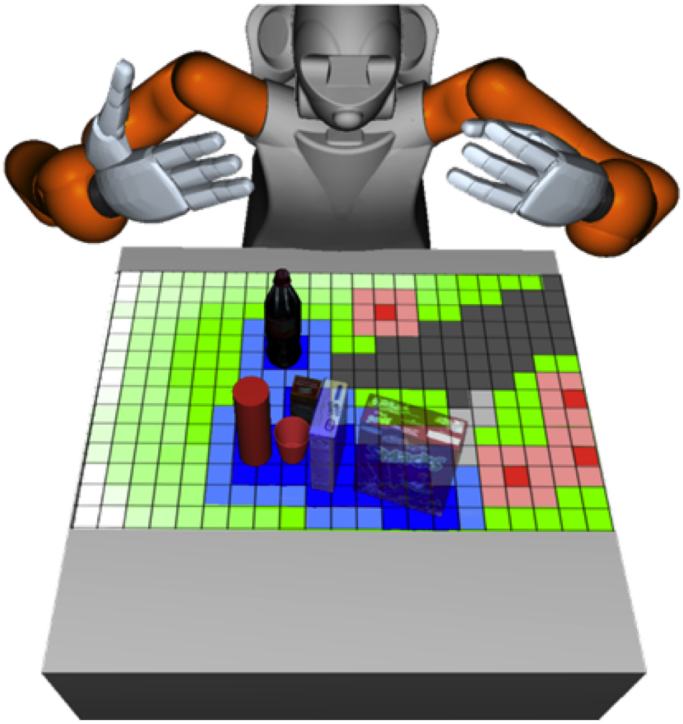


Figure 2.5.: Manipulation specific discretization in reasoning using logic-based planning. - [120]

during the planning process. Expressions in the logic-based planner are therefore extended by so-called *semantic attachments*, which represent evaluation of symbols by the motion-planning component. The system is shown to work on simulated, yet complex pick-and-place tasks.

Another system integrating logic-based planning and motion planning is presented in [120]. Both motion and grasp planning are thoroughly investigated for abstraction into symbolic expressions. To rearrange blocked objects on a table, discretization of both object poses and possible grasps is utilized as shown in Figure 2.5. Discretization serves as a heuristic to create the symbolic search space, while real motion planning is used to evaluate the actual feasibility of operations in a given scene. Monitoring validates the resulting plan during execution.

It can be concluded that logic-based planning is a suitable tool for strategic decision making, but extensions and interplay with other methods are needed to solve real robotics challenges. Critical aspects include balancing multiple goals, considering noisy sensor inputs, combining reactive low-level behavior or low-level motion planning with abstract planning, dealing with continuous time and acquisition of model knowledge. Thus, much recent research has concentrated on related, yet more powerful planning methods (see Section 2.2) and applications to hybrid, hierarchical architectures as described in Section 2.1.3.

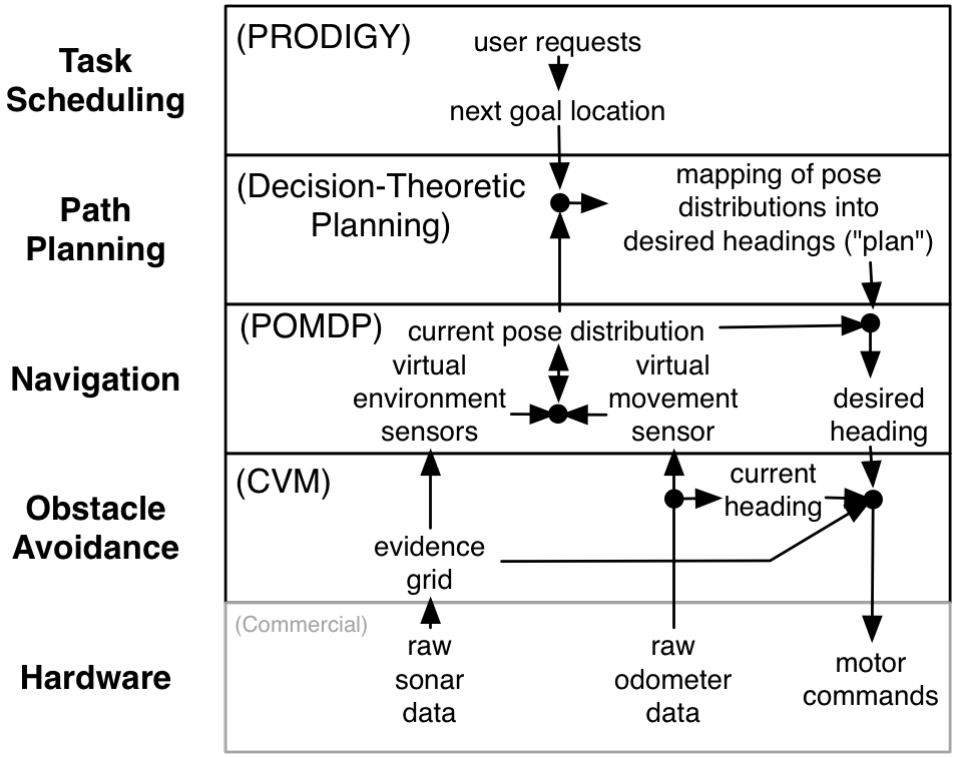


Figure 2.6.: A layered architecture for office delivery robots. - [139]

2.1.3. Architectures for Autonomous Behavior

As a result of the insight that fast, reactive control, continuous motion planning and symbolic, strategic planning all have their strengths and weaknesses and are well suited for different aspects of robotics, a lot of effort has been spent on investigating how to use all these paradigms together on a single robot system. To organize component interplay, information flow, knowledge modeling, but also learning and overall system design in a structured manner in such hybrid systems, a clearly defined architecture is necessary. Most architectures for autonomous robots are modular and often hierarchically layered. Usually, autonomy is achieved by application of distinct paradigms in different sets of modules. Many architectures have been presented in the past and concepts are still being evaluated and enhanced further.

A simple, yet quite successful concept is *three-layer architectures*. In this strictly hierarchical architecture as discussed in [49], a reactive *skill* or *control layer* sits at the bottom, a reactive *sequencing layer* resides in the middle and a *planning* or *deliberative layer* is at the top. By these means, the strengths of behavior-based control or related techniques as present in the skill layer can be brought together with explicit, symbolic planning techniques at the deliberative layer, while the sequencing layer connects these two and manages multiple skill domains. Basically,

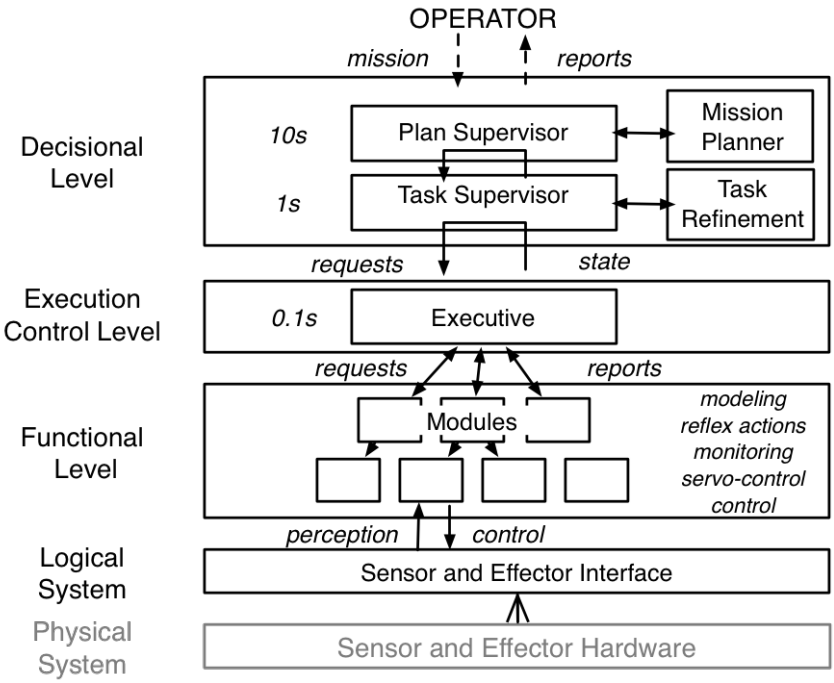


Figure 2.7.: A typical three layer architecture. - [2]

the execution-time architecture as presented in Section 3.1 is such a three-layer architecture. The broadest classification of architectures is the distinction of behavior-based, hierarchical and hybrid as given by [30]. Most currently employed architectures are some type of hybrid, which is true even for most three-layer architectures. However, actual hybrid architectures may differ significantly from each other.

A practical example of an architecture providing autonomy for a mobile robot is given in [139]. It is highly hierarchical and having only one skill domain leads to quite tailored functional layers, depicted in Figure 2.6. Direct access to the hardware is managed by an obstacle-avoidance behavior control layer. On top sits a navigation management layer that uses a POMDP (see Section 2.2.3) for fine-grained decision making about driving motions. Superior to that layer is a decision-theoretic path-planning layer, while the top-most layer is formed by a logic-based planning system. In contrast to some other architectures, this one is clearly shaped by the algorithmic functionalities of its components.

An example of a typical, hierarchical three-layer architecture, scalable to robots with multiple skill domains is described in [2] and shown in Figure 2.7. On the lowest, functional level, behavior-based control modules directly access the hardware and implement perception processing and reflex actions. These modules are managed by an executive control layer, having a cycle time of approximately 100 ms. On top, a deliberative layer is connected to the executive

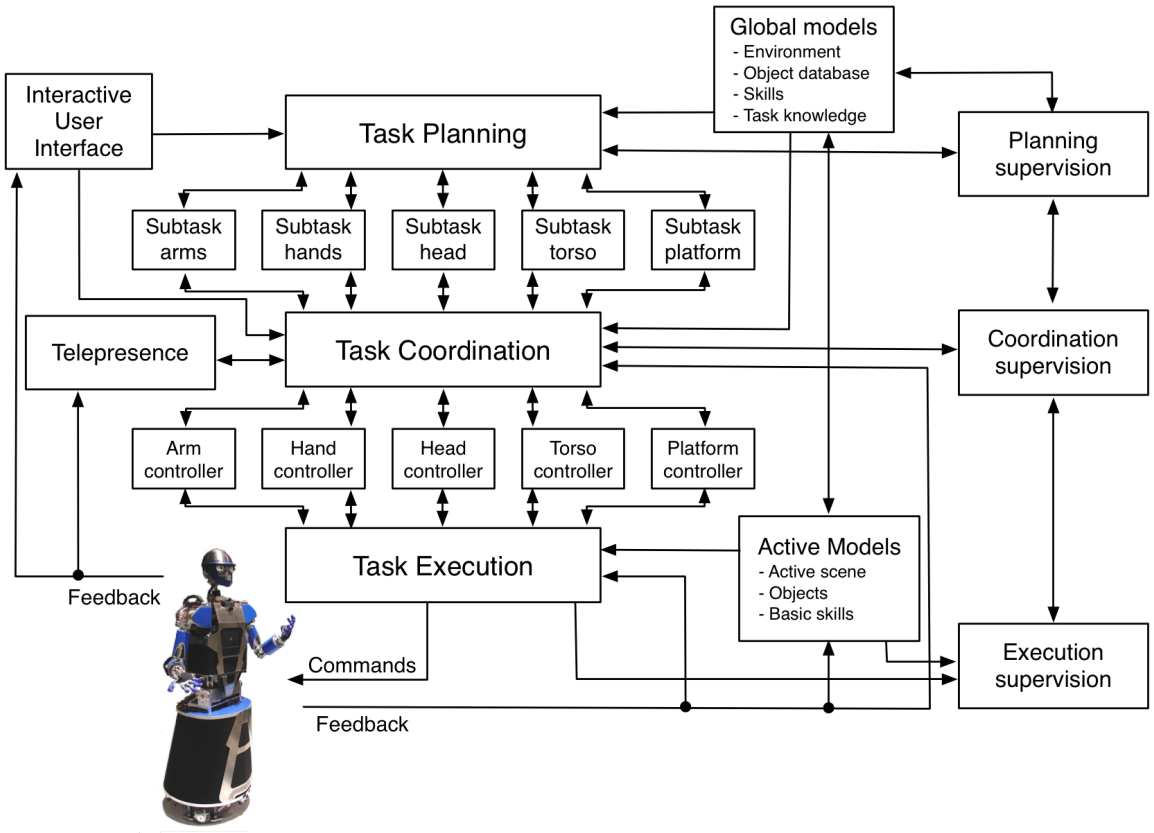


Figure 2.8.: The control architecture of the humanoid robot ARMAR-III. - [6]

layer and consists itself of two parts: a task manager with a cycle time of approximately 1 s and a logic-based planner with cycle times around 10 seconds.

More recent presentations of architectures for autonomous robots focus less on a general paradigm, but instead present more details about actual implementation and interplay of various algorithmic components.

Another architecture, SAPHIRA discussed in [79], is more hybrid in style. Independent perceptual routines and action behaviors can access a common short-term memory space that makes the system ideal for robots with multiple skill domains. On a more abstract level, a deliberative *Procedural Reasoning System* (PRS) coordinates specialized planners and behaviors.

When including all the major domains: robot mobility, natural human-robot interaction and object manipulation on a single, autonomous robotic system, clarity and an unambiguous structure of the architecture become most important. The architecture for the humanoid robot ARMAR-III (see Figure 2.8) presents such a typical three-layer architecture in [6]. A skill layer, a sequencing layer and a deliberative planning layer are hierarchically organized and connected to explicit, model-based memory.

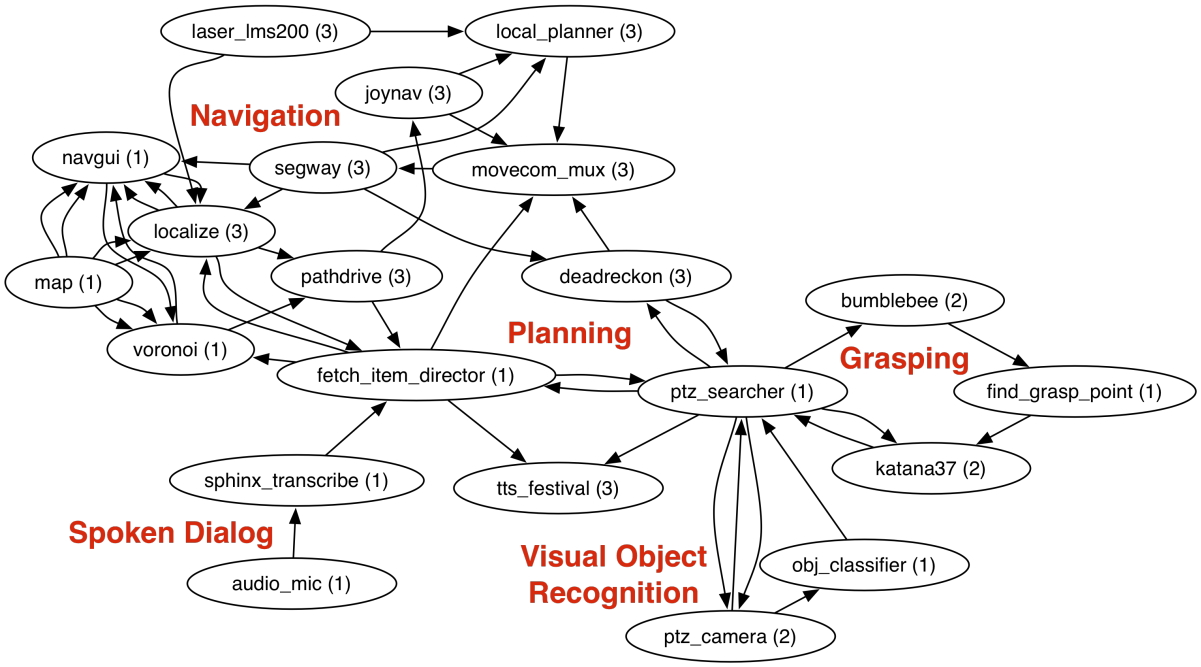


Figure 2.9.: STAIR: a behavior-based architecture. - [114]

An interesting architecture based on a collection of rational agent components has been developed for the ISAC humanoid robot [107]. By having a set of loosely coupled agent modules that perform independent processing of algorithmic routines, the behavior-based control paradigm is applied within that architecture. However, these agent modules can also access several common memory and model structures. With this memory, the robot can map suitable actuation sequences onto certain situations that are perceived configurations of the world. Thus, the ISAC architecture gives some insight into integration of a behavior-based control module approach with an explicit environment model memory.

A system, STAIR, for a fully multimodal robot that is more closely related to the behavior-based paradigm is presented in [114] and shown in Figure 2.9. Yet, corresponding modules still have access to explicit model-based memory, and deliberative planning is integrated.

Tailored to perform simple tasks very robustly in long duration missions including autonomous mobility and human-robot interaction, the RoboX system presented in [70] takes a very different approach. Basically a two-layer architecture, the system consists of a supervisory sequencer, processing a certain mission coded as a finite state machine. Available modalities are encapsulated by low-level modules, performing sensor processing and actuator control for certain aspects of the robot. The results show that such a simple architecture is easy to handle and robust in clearly defined and not too complex scenarios.

Further elaboration of actual data flow, interfacing and coordination among modules of a behavior-based control architecture is presented by [16]. It gives a detailed definition of nec-

essary control modes for behaviors, a network-capable communication layer, an arbitration module that manages module activity and a resource manager that controls access to actuators.

Although the architectures presented are quite varied, some common characteristics reoccur most of the time. Modularity and hierarchy are schemes, central to both behavior-based control as well as hierarchical sequencing and planning architectures. Additionally, hardly any practical behavior-based architecture is completely without explicit model knowledge about the world.

2.1.4. Considering Autonomy for Robots with Multiple Skill Domains

Analysis of prior research into autonomy can give some insights for designing a system providing autonomy for a service robot covering multiple skill domains. Behavior-based control highlights the importance of assessment of multiple goals by the robot, the need to consider noisy sensors and the benefits of a layered control hierarchy. While those aspects are neglected by classical logic-based planning, it underlines the strength of symbolic abstraction to manage complexity. Introducing distinct, discrete symbols helps to manage real-world complexity by application of explicit simplified models thereof. Based on such a model, directed planning can consider effects of actions far into a potential future. Actually implemented architectures tend to combine strengths of both paradigms, often with a behavior-based control approach managing robot hardware and a planning system on top, coordinating behaviors. These insights have been taken into account in the design of the execution-time architecture, described in Section 3.1 and for the choice of the formal decision-making framework (see Section 2.2).

2.2. Markov Decision Processes

Hierarchical control architectures providing autonomy for service robots with multiple skill domains usually have an abstract decision-making component as discussed in Section 2.1.4. Such a component provides strategic action choices, selecting tasks or behaviors that cover only a small temporal subset of behavior within a mission. Usually, such a component derives the choice by deliberative reasoning about environment models and robot action effects.

Logic-based planning as discussed in Section 2.1.2 provides such capabilities and has been examined in several exemplary robot-control systems (see Section 2.1.3). However, classical logic-based planning was originally developed within classical artificial intelligence (AI) research which often makes some fundamental, specific assumptions about the world in which an intelligent agent is acting. These assumptions usually have serious consequences when applied in realistic robot activity settings.

Two crucial assumptions relate to the fundamental type of model of world states and robot actions as defined in Section 2.1.2. In classical logic-based planning, an abstract action performed in a certain world state always leads to exactly one certain result state. Additionally, the reasoning robot is assumed to always know which state the world is currently in. These two assumptions lead to the ability to find a sequence of actions in the model that will certainly transfer the current state of the world into the desired goal state. For states that are abstract representations of certain world configurations in a complex robot setting and abstract actions representing a set of complex actuator behaviors therein, these assumptions usually do not hold.

Consequently, one is led to introduce *uncertainty* about the true, intrinsic current state of the world at a certain moment as well as predicting the resulting effect of an action performed in some specific state. Thus, the robot cannot be sure that a certain sequence of actions will transform the world into a desired goal configuration. Instead, another reasoning paradigm has to be introduced which is still able to provide reasoning about potential action effects in the future, but is able to consider these uncertainties as well as weigh multiple goals.

2.2.1. Bayesian Paradigm

To consider uncertainties when reasoning about world states and robot actions, a formal definition and quantitative descriptions of those uncertainties are necessary. Probability theory and probability calculus can provide this in a sound manner and allows to calculate with uncertainties described that way.

However, before exploring ways to calculate with those probabilities, the fundamental question how probabilities can represent uncertainties has to be illuminated. Basically, there are two paradigms defining how probabilities represent real-world events. The *Frequentist* view defines a probability as the asymptotic result of a series of trials, tending to infinity, in a stochastic process. By these means, it is impossible to retrieve values for robotic domains as the exact conditions of states and actions vary all the time.

Hence, another view is prevalent in robotics: the *Bayesian* paradigm. Here, a probability is seen as a measure of information about real-world events. Probabilities as present in models and resulting from further computations model the expectation or assessment of autonomous systems in view of real-world events. From a frequentist point of view, actual values can often be just coarse approximations of the values that a hypothetical asymptotic experiment would yield. However, the robot is able to perform better when considering and calculating with Bayesian probabilities than when neglecting uncertainty at all. Bayesian probabilities modeling stochastic real-world characteristics can be acquired by all kinds of robot learning and analysis of all kinds, but do not represent the asymptotic result of a precisely controlled experiment.

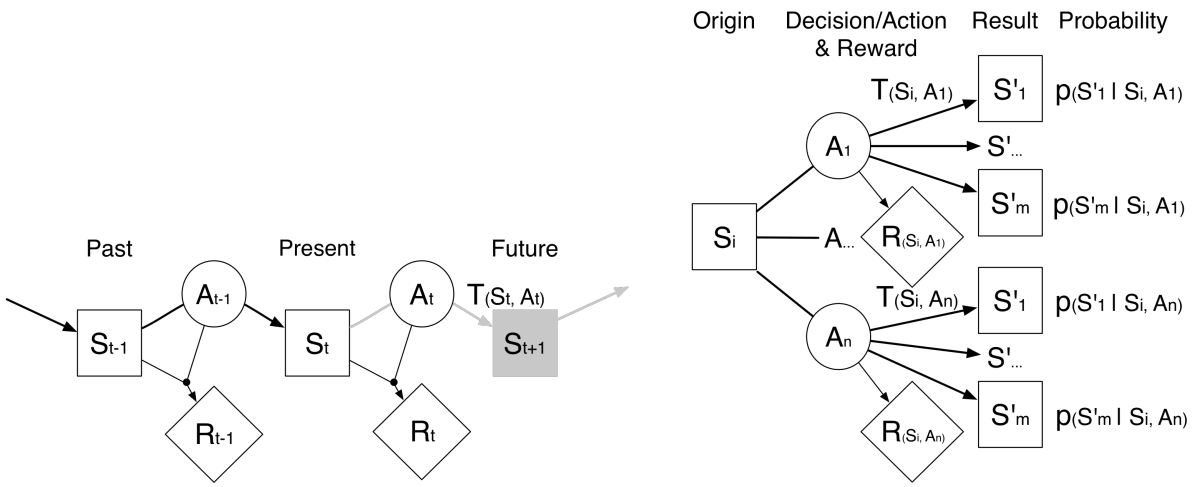


Figure 2.10.: MDP model (right) of courses of events (left). - [122]

2.2.2. Fully Observable Markov Decision Processes

Applying the Bayesian concept to robot action courses of events leads to a stochastic action model with not just one certain outcome s' of an action a performed in a certain state s. Instead, several different outcomes can be possible with the stochastic frequency of each possible outcome modeled as a Bayesian expectation probability. Each result s' of an action a in world state s can be considered as a *transition* t(s, a, s') of the world state as depicted in Figure 2.10. If the probability of the transition depends only on the current state s of the world when performing action a, it can be modeled by a Markov process of the first order [11].

Defining the probabilities of all possible outcomes of each available action in each modeled state leads to the full *transition model* T: $S \times A \times S \rightarrow p(s'|s,a)$. Such a transition model supersedes a deterministic action operation model as used in classical logic-based planning. Because action effects are now considered as stochastic, a reasoning agent cannot find a sequence of actions that will certainly lead from a current state to a specific goal state. The only deterministic element remaining is the choice of an action to be performed next. But when reasoning about future events, the robotic agent has to consider multiple possible sequences of events and thus states and actions.

From considering multiple courses of events arises the need to weigh several courses of events against each other and thus quantify their *utility* U for the robot. Along with such a need comes the opportunity to include and rate secondary goals and constraints. Consequently, the utility of performing a certain action a in a certain state s can be defined, modeling goals, constraints and action costs, generally speaking the overall motivation of the autonomous robot. Defining it for all available pairs of states and actions in the form of positive values (rewards) or negative values (costs, penalties), leads to a so-called *reward model* R: $S \times A \rightarrow r(s,a)$.

When reasoning about potential future chains of events, the agent is then able to quantify a course of events as the sum of all individual rewards. By these means, multiple potential chains of events can be combined by calculating with their quantitative values. Thereby, even with stochastic action effects, an agent can plan ahead when having a model (S, A, T, R). This leads to decision making within the framework of a *Markov Decision Process* (MDP) [59]. Within such an MDP, a robotic agent can reason about potential future sequences of events and select actions which maximize the expected sum of future rewards over some period of time. Action selection during autonomous execution of a robot task is based on a function, called *policy* P, which contains the most promising action for each situation in a scenario. A robotic agent selects the action to be executed next based on the policy, then assesses how the state of the world developed after that action and subsequently selects the best action for this new state. As the world is not deterministically predictable, reasoning encompasses *risk minimization* and *reward maximization*.

Policy computation can be performed in two distinct manners. One approach does not utilize an explicitly known transition model T, but calculates an asymptotically optimal policy by means of *reinforcement learning* (RL). The agent acts in a simulated or real environment and gets feedback signals depending on its action choices. Such a signal can, for instance, directly be derived from an explicit reward model. Based on these signals, computation improves the policy successively towards the optimal policy. Reinforcement learning can be performed in a distinct learning phase or for improvement during the main task execution stage. While avoiding the need for an explicit transition model of the whole domain is a big advantage, the need for a large number of reinforcement signals to compute a good policy is a huge disadvantage.

On the other hand, the model-based approach takes fully defined transition and reward models to compute an asymptotically optimal policy by value iteration as described in Section D.1. Policy computation can be performed offline, before autonomous execution with the policy serving as a universal plan which is queried by the robot each time a previously chosen action has terminated. Some techniques for very large numbers of states and actions perform online computation of the policy in relevant parts of the state and action space. For a discrete state and action MDP, the policy is a simple lookup table which assigns an optimal action to each state: $a_j = \pi(s_i)$. Consequently, value iteration does not need any, potentially extensive, trial stage. Yet it can be considered as a challenge to create a fully defined transition and reward model for a specific mission.

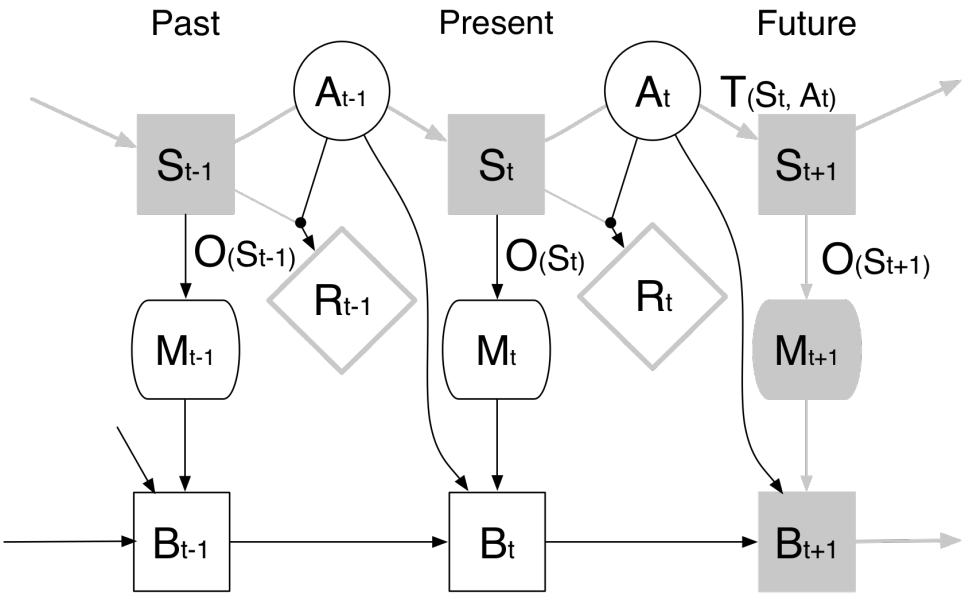


Figure 2.11.: POMDP model of courses of events with aspects hidden to the agent in grey. -[122]

2.2.3. Partially Observable Markov Decision Processes

Policies for fully observable MDPs assign exactly one optimal action to each state. Thus, it is assumed that a robotic agent knows exactly which state the world is in at any moment. Such omniscience about world states can only be correctly assumed for some artificial game settings, but not generally for real-world robot settings. Uncertainties about states can arise from both sensor measurement noise and limitations as well as more fundamental occlusions, such as not seeing through a wall. Instead of assuming the world to be fully observable, it has to be assumed to be*partially observable*, thus relaxing the requirements for the robot's knowledge about world states.

This leads to a model concept where both effects of actions and perception of states have to be considered as stochastic. Fully observable MDPs can be extended into a more general framework: the *partially observable Markov decision process* (POMDP). It introduces a separation of the true, intrinsic state of the world on the one hand and a subjective *belief state b* which is internal to the robotic agent on the other.

True states of the world are still defined as in fully observable MDPs and the domain has to be modeled such that one distinct states holds at any time. However, the robotic agent has a belief b which contains a Bayesian probability for each state in the state space, representing the agents likelihood belief that this is the true state of the world at a moment of time.

During execution of a task or mission, the agent updates its belief by Bayesian filtering, fusing prediction of action effects based on the known transition model with noisy measurements of

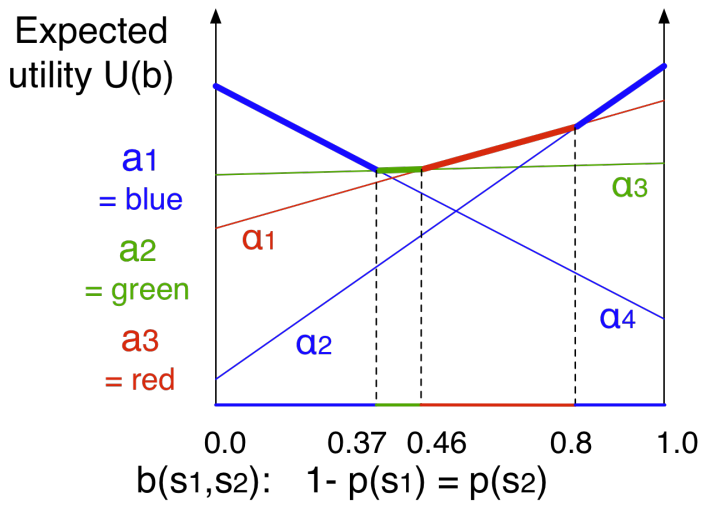


Figure 2.12.: POMDP value iteration policy visualized in a 2D slice. -[126]

the world. For computation of this fusion, quantitative information about measurement errors has to exist. Therefore, the MDP model has to be extended by a set of *measurements* M that the robot agent can perceive and an *observation model* O which relates measurements and states. Within the observation model, entries denote the probability that a certain state s_i generates a measurement m_k: $O(s_i, m_k) = p(m_k|s_i)$. With this extension of the MDP, a POMDP is defined as (S, A, M, T, R, O) and the runtime extension b [7], [144]. A POMDP flow of events is sketched in Figure 2.11.

Having only indirect, imperfect knowledge about the state of the world massively complicates application and computation of a decision policy. As the belief probability distribution is the primary knowledge of the robot about the world, decision making has to be based on the belief. The POMDP policy has thus to be a function based on the belief. For a discrete state space, all possible belief distributions form a continuous, but bounded domain for the policy. Interpreted geometrically, with each state forming a vector space dimension, the space of all beliefs forms a bounded simplex as the sum of all probabilities is always 1.0. Each possible belief is a point within that simplex. A policy for decision making has thus to assign an action to each belief point. Consequently, one needs a continuous, gapless definition that assigns parts of the belief space to certain actions.

Computing POMDP policies is far more complex than computing MDP policies. The representation has to stretch over the whole belief space and reasoning has not only to consider stochastic action outcomes, but also differing information uncertainty the robot may have about the world state. Because of this increased complexity, reinforcement learning is hardly usable in this context. Instead, value iteration can combine transition probabilities, reward values and observation probabilities from the explicit process model in its computation process. In such

techniques, explicit representation of utility values over the belief space is given by linear functions α, representing the choice of a certain action and expected future utility for some sets of potential sequences of events as shown schematically in Figure 2.12, 2.13.

The policy is formed by the convex maximum of a set of linear functions representing the action with the highest expected utility over a part of the belief space [24]. Computing optimal policies even for limited horizons, that is considering a limited number of potential future actions, as well as small state and action spaces is highly intractable. The computational complexity for calculating optimal policies is 2-EXPSPACE, which is doubly exponential in memory space [71]. Therefore, computation of exactly optimal policies is practically infeasible for even the smallest realistic mission models. Instead, approximate methods have become popular that compute only approximately optimal policies as discussed in Section 2.2.5.

Another limitation of typical POMDP application is, as with logic-based planning, the need for discretization and abstraction for both state and action space. As previously discussed, discretization is a paradigm to simplify and combine real-world aspects under common labels to cope with its complexity. For some challenges, however, discretization is an insufficient approximation of the state space. To tackle such domains under the framework of POMDP decision making, investigation into POMDP modeling for continuous domains has been performed.

Monte-Carlo POMDPs [154] sample a set of representative points from a continuous state space and compute a policy based on these points. By using a finite set of points, the technique resembles a discrete representation, yet is more flexible. On the other hand, complexity is still very high and application is only feasible for very specific problem domains.

Another approach presents a more general theoretical framework for computation of approximately optimal policies for continuous POMDPs [111]. Hereby, transition, reward and observation model are represented as Gaussian mixtures and cover continuous state spaces. The belief is either defined by representative points or gaussian mixtures defined over a finite subset of the continuous state space. While the technique is sound, both modeling and computational expenses are very high and further application to real-world challenges remains limited.

In summary, modeling continuous POMDPs in a way to compute applicable policies in a reasonable amount of time is not yet sufficiently understood to apply them to practical problem domains. Most applications thus use discrete state and action POMDP representations as discussed in Section 2.2.6. Therefore and because of focus on abstract, strategic decision making, the representation of discrete POMDPs has been chosen for the system presented in this thesis, presented in Section 3.

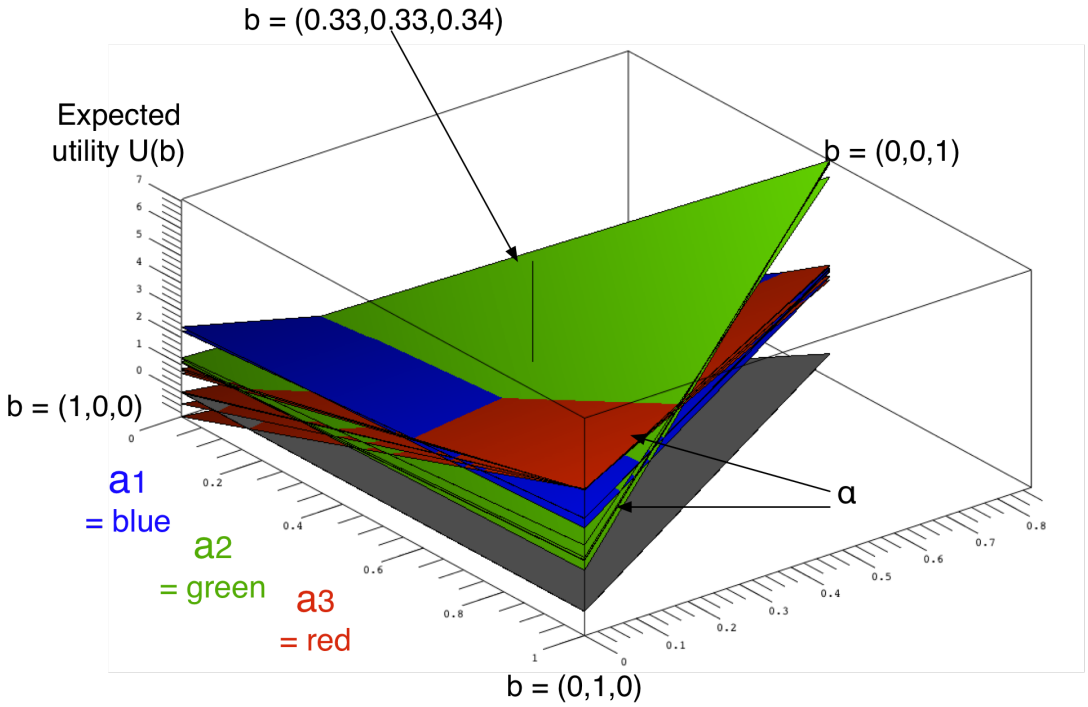


Figure 2.13.: POMDP value iteration policy visualized in a 3D hyperslice. - [126]

2.2.4. Mixed Observability Markov Decision Processes

Combining the strengths of fully observable and partially observable Markov decision processes within a single representation led to the development of the mixed observability Markov decision process (MOMDP) [101]. The idea is to benefit from less complexity concerning both modeling and policy computation for fully observable MDPs with respect to aspects of a problem domain where the assumption of fully observable state properties is sufficient. Only those aspects that cannot be sufficiently assumed to be fully observable preserve a POMDP-style model.

As a result, a MOMDP contains a factored state representation with both a fully observable part X and a partially observable part Y. There are two transition models, $T_x : X \times Y \times A \times Y \rightarrow p(x'|x,y,a)$ and $T_y : X \times X \times Y \times A \times Y \rightarrow p(y'|x,y,a,x')$. An observation model O is only defined for partially observable sub-spaces.

By these means, reasonable approximately optimal policies can be computed for larger state spaces and more complex transition models than for pure POMDPs. Yet, modeling robotic problems by these nested transition models is difficult and not always applicable.

2.2.5. Approximately Optimal Policy Computation for POMDPs

POMDPs make fewer fundamental simplifying assumptions about the problem domain than logic-based planning or MDPs for modeling abstract decision making of autonomous, intelligent agents. Thus, they are a promising framework for autonomous planning and reasoning in the real world. The greater scope comes at the price of higher complexity in both modeling and policy computation. Since computing exactly optimal policies is intractable, a research focus is on the development of efficient algorithms to compute approximately optimal policies. Only with such algorithms, application of POMDPs becomes feasible in real-world domains.

Many different approaches to compute approximately optimal policies have been investigated [1]. As with MDP policy computation, the most general distinction can be made between the class of model-based algorithms and those that do not contain any explicit POMDP model representation. Among the former type, two groups of techniques can be further distinguished: those which compute explicit utility values along with a policy and those computing only an immediate policy, a belief to action-choice mapping without utility values.

One of the simplest approaches among model-based value computation is the *most-likely-state* heuristic [1]. An MDP policy is computed based on the model. This policy takes the state with the highest probability in the belief (the most likely state) as input to retrieve an action. Differentiation between varying probabilities of less likely states in the belief, which is usually an important aspect of POMDP reasoning, does not occur, leading to unsatisfactory performance.

The Q_{MDP} heuristic [1] considers a belief only for one step and assumes the state to be fully observable afterwards. It is a very crude approximation of an optimal policy, which fails in many situations with uniform beliefs.

Tackling the complexity of belief updates during POMDP value iteration can be achieved by discretization of the continuous belief space. Only a finite set of representative belief probabilities is chosen for value function updates, as shown in Figure 2.14, significantly reducing the computational complexity of the update. Yet, the value function is still defined over all of the belief space. The simplest method to choose those points is to apply a regular grid [1], but with large state spaces, belief-space dimensionality becomes too high for this technique to cover belief space sufficiently.

Instead, more sophisticated ways to choose those points have to be applied. Dynamic assignment of belief points to update the value function during lookahead leads to point-based value iteration techniques. Intelligent ways to determine regions in belief space, reachable within the model and furthermore likely to be reached, thus relevant for a policy, can significantly improve policy computation speed and quality. Several approaches exist as discussed further below.

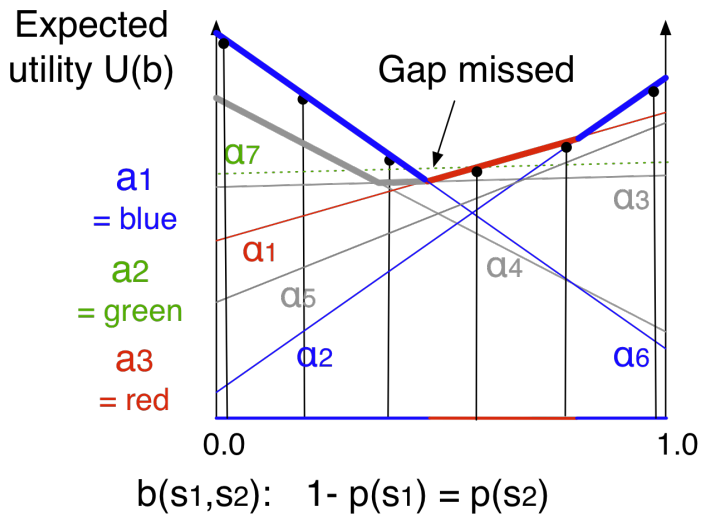


Figure 2.14.: Grid-based value iteration policy concept visualized in a 2D slice. - [126]

Direct policy learning is a model-based alternative to value iteration techniques while policy gradient search is model-free [1]. As with MDP reinforcement learning techniques, policy gradient uses experience of the agent to gradually improve the policy. Because a large amount of experience is necessary, leading to slow convergence towards good policies, such a paradigm is not practical for abstract control of a service robot covering multiple skill domains.

Continuing progress is made concerning point-based value iteration, the most practical class of techniques for policy computation of general purpose POMDPs.

Point-based value iteration, introduced by [108], calculates the value function, representing the expected utility of a future action sequence, only for a representative set of belief points. For each belief point, exactly one linear function α (described in Section 2.2.3) is computed in each iteration step for each belief point. By these means, the number of α, thus memory and computational requirements are significantly reduced compared to exact value iteration. On the other hand, each α is still defined over all of the belief space, leading to a fully defined value function as the convex maximum of the set of linear functions α.

Performance of point-based value iteration depends on the selection of belief points. The better a sampling set is chosen to reflect important areas of a belief space, the better the approximation of the resulting policy to a hypothetical exactly optimal policy. Important regions of the belief space are those likely "visited" by the robot agent during execution, called reachable belief $R(b_0)$. Areas where different optimal action choices occur at very similar belief distributions are relevant, too.

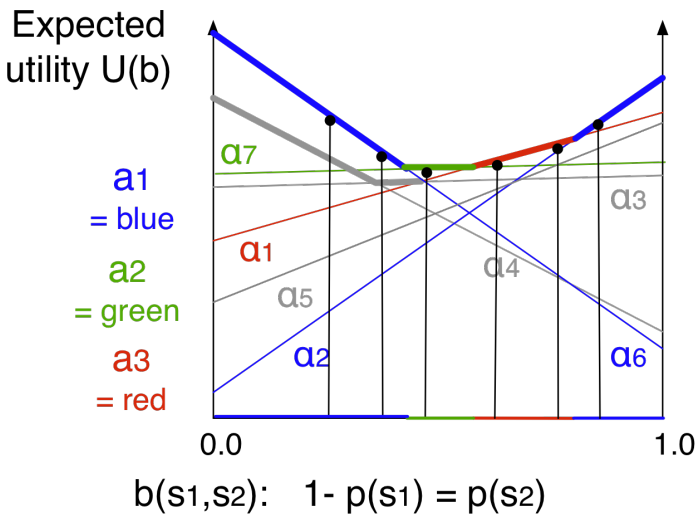


Figure 2.15.: PBVI value iteration policy concept visualized in a 2D slice. - [108], [126]

In standard PBVI, belief points are sampled incrementally by simulating execution of the current policy by the agent based on the POMDP model. Resulting new points are then ranked for relevance, and those below a pruning parameter are cut off. PBVI guarantees the policy to converge towards the exactly optimal policy, making it an anytime algorithm. Computation can be terminated after a given number of iterations or time, leading to an approximate value function with a guaranteed error ε. A more explorative approach to sampling belief points for value iteration, PERSEUS, is presented in [145].

An elaborate heuristic to select actions and observations for belief point updates, heuristic search value iteration (HSVI), is given by [141] and [142]. In this case, both upper and lower bounds representing the approximation of the exactly optimal value function are updated in each iteration step. Those bounds help to select belief points for updates in areas of the belief space where the error is still relatively large and another optimal action choice most likely. HSVI performs superior to PBVI in most classic benchmark challenges.

The concept of HSVI is further extended by the *Successive Approximations of the Reachable Space under Optimal Policies* (SARSOP) algorithm, presented in [80]. In contrast to standard PBVI, this algorithm strongly distinguishes between general reachable belief $R(b_0)$ and optimally reachable belief $R_{\pi^*}(b_0)$. Sampling of belief points is concentrated in the parts of the belief space that are reachable with the currently computed policy. As the computed policy converges towards the optimal policy, representative belief points concentrate increasingly in regions relevant for the exactly optimal policy. This algorithm can be considered as mature, performs superior to HSVI and is used as policy computation algorithm in the system presented in this thesis.

Another algorithmic development is region-based value iteration, described in [88]. Instead of belief points, ellipsoid belief regions serve as support structures for value function updates. However, the results presented do not show superior performance to HSVI and a comparison to the even superior SARSOP is not given. Utilizing the topological order of a domain for policy computation is explored in [34]. Although results are promising, the process of topological ordering is complex and has been unsuccessfully tried for mission models investigated in this thesis. Apart from value iteration, policy-gradient search has been further developed as presented in [1]. Yet, there are no results given that are comparable to the performance of point-based value iteration algorithms.

In summary, algorithms for computation of approximately optimal policies with guaranteed error bounds enable the application of POMDP reasoning in realistic robotic domains. The approximation error given by such algorithms is clearly comprehensible for any model, making automatic evaluation of the quality of a policy feasible. Thus, it can always be assessed if an expedient policy can be computed in a given timeframe for a certain robot mission POMDP model. Remaining approximation errors are usually small enough to be negligible in practice and are far smaller than modeling errors. As all these well-performing policy computation algorithms need an explicit POMDP model, acquisition of a model, representing real-world causality as closely as possible, is very relevant. This task is a focus of the present thesis and discussed in Section 4. As a drawback, existing algorithms are still limited when coping with very large numbers of states and actions. Future research has to focus on hierarchical representations and online methods focussing on policy updates on local belief regions.

2.2.6. Utilization of POMDPs for Autonomous Robots

While POMDPs have been investigated extensively in terms of policy computation and factored representations, their application in robotic domains has been rather limited so far. On the one hand, application is made feasible only by recent progress in policy computation algorithms. On the other hand, modeling state and action spaces for robotic problems in a general manner seems to be a hard challenge. Furthermore, generating an explicit mission model necessary for both policy and execution-time belief computation is usually complicated.

Autonomous navigation is the most widespread application domain. Most policy computation and factored representation techniques discussed in the previous Sections have been evaluated on simulated or real robot navigation problems. The state space is usually formed by grid-based discretization of a 2D navigation environment and actions encompass different directions

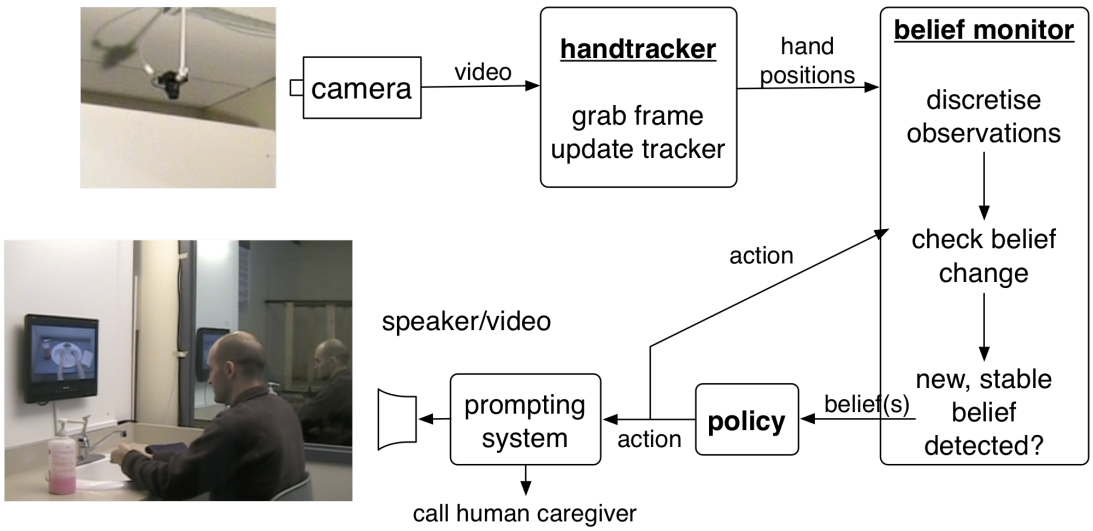


Figure 2.16.: The architecture for dementia patient handwashing support using POMDPs. - [56]

the robot can drive towards [47]. Action effect uncertainty is modeled by varying driving action results. Observation uncertainty is modeled by offsets in grid-point localization. Positive rewards are given for reaching certain goal points or catching other agents while negative rewards are given for bumping into walls or traps. Such transition, reward and observation models are very simple, thus easy to create manually and reuse. Topological navigation investigated in [41] introduces the use of episodic memory for POMDP decision making.

Natural human-robot interaction and autonomous object manipulation are much more difficult to model and less investigated. In addition, interdependencies between skill domains can make a factored representation unsuitable, increasing complexity of transition models as discussed in Section 3.4.

A full system discussed in [110], [109], integrating the skill domains of autonomous navigation and natural human-robot interaction, utilizes POMDP decision making for strategic mission autonomy. Designed for elderly care support, the system is able to cope with indoor navigation uncertainty and speech recognition errors. It resembles the system presented in Section 3, yet encompasses fewer skill domains and just a manually created, quite simple mission model.

Not really a robotic application, but nonetheless multimodal human-machine interaction, assisting dementia patients with a POMDP reasoning system is presented in [56]. In this application, actions are formed by spoken utterances or calling for human help, while the state space models situations of a human hand washing process. This approach is interesting as it models quite abstract states and actions. States are relevant world configurations which can be distinguished by perception components. Uncertainties arising from sensor signal processing characteristics are explained in detail and the observation model derived from these character-

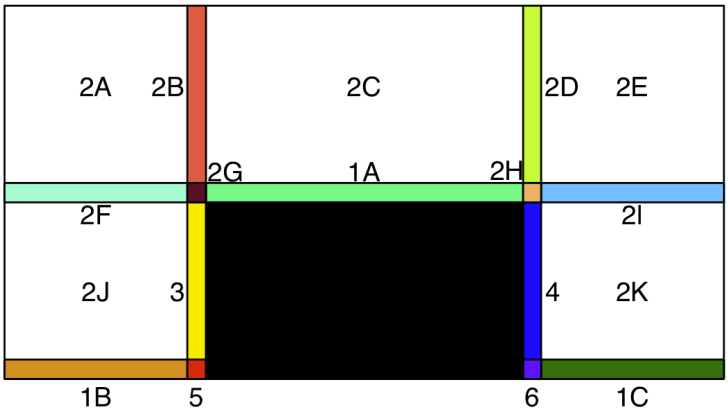


Figure 2.17.: Model of the state space in Grasping POMDPs with the object in black and free space in white. The view is a schematic vertical cut. - [60]

istics. In this way, this work is related to the approach described in Section 3.3. The processing architecture is shown in Figure 2.16. The whole transition model is defined manually, however, limiting flexibility and scalability to new missions.

Another non-robotic human-machine interaction domain with autonomous decision making in a dialog system using POMDPs is discussed in [162].

Modeling a simple grasping problem with a proprioceptive, elastic gripper is explored in [32]. In this case, states are defined by various contact configurations between a single hand and a single object. Actions represent different motor command strategies for individual degrees of freedom. Thus, a very low level robotic manipulation scenario is modeled by this POMDP.

A low-level grasping problem is formulated as a POMDP in [60]. Similar to grid-based modeling of autonomous navigation, state discretization is based on geometric regions around an object (see Figure 2.17) which is to be grasped by the robot. These regions are not of uniform size and contacts are also factored in. Actions comprise distinct, small hand or finger movements. Such a very specific modeling technique is restricted to grasping objects with a simple shape.

Scaling POMDP grasping to more complex geometric objects is discussed in [61]. Discretized object poses in cartesian space define the state space while a set of manipulation motions acquired by offline motion planning constitute the action set. Observations arise from touch sensors measuring certain contact configurations between robot gripper and object. Certain manipulation actions may reduce belief uncertainty more than others, leading to preferred selection of those actions in more uniform belief states. Thus, information gain is effectively handled by this problem modeling technique. However, fixed offline computed motions limit flexibility and lead to a high computational effort for more complex manipulation problems

in practice. In contrast, more flexible manipulation strategies are used as representations for elementary manipulation actions in this thesis, as discussed in Section 3.4.

An unconventional approach to model visual information gain tasks by means of hierarchical POMDPs has been investigated in [146]. Action selection chooses among operators which trigger certain image processing algorithms to classify and localize different objects on a table surface. Emphasis is placed on the observation model design, while the transition model is extremely simple. Therefore, the POMDP design is unusual and also very problem-specific, yet scales well to larger scenes. This technique is evaluated extensively against continual planning and shows superior performance. Because of its specificity, especially concerning the transition model, extending that method to more general service robot missions is not straightforward.

In summary, efforts to model autonomous robotic navigation, human-robot interaction or object manipulation as POMDPs have led to varying approaches, both low- and high-level. State- and action-space definitions are sometimes intricate and problem-specific, and model generation is performed manually. Consequently, more generic approaches to model scenarios covering multiple skill domains and methods to generate associated models automatically have to be investigated as discussed in Sections 3 and 4.

2.2.7. (PO)MDP Model Generation

For most generic robotic POMDP application domains using model-based policy computation, efficient generation of POMDP mission models is the most obvious bottleneck. In most investigated applications, all model elements (S, A, M, T, R, O) have been defined explicitly by an expert designer while also having a simple internal structure. Handling generic robot domains, as is the focus of this thesis, in such a manner is infeasible. Automatic generation of model elements either has to create some or all model elements from scratch or incrementally improve an initial model in learning trials or during actual autonomous mission execution. The latter kind closely resembles reinforcement learning techniques, yet model learning may converge faster than policy learning.

MEDUSA, discussed in [68], is a framework that improves a given initial POMDP model for a mobile robot by execution-time learning. The focus is on combining the strength of model-based policy computation, relieving the need for excessive learning queries as in model-free reinforcement learning, with the online learning capabilities of the latter. Advantages are both in improving an initially incompletely known model as well as adapting models in environments with non-stationary dynamics. Model elements (S, A, M, R) are assumed to be known a-priori,

which are usually the simpler model elements, but still have to be defined manually for any new mission. Model elements T and O are continually improved by MEDUSA.

Learning is achieved by incrementally improving a Dirichlet distribution over a set of model hypotheses. Such a set of models is chosen initially based on prior knowledge about T and O. Subsequently, a policy is computed for each model. During execution time, a model is sampled at each time step from the Dirichlet distribution. Based on the policy computed from that model, an action is selected to be performed next. Model learning is achieved by incrementally improving the Dirichlet distribution. For this purpose, the robot needs information about the true state of the world at a given point which is introduced by a so-called "oracle query". In MEDUSA, it is assumed the robot is able to query a human or high precision sensor system from time to time as an "oracle" and retrieve the intrinsic state of the world during learning. While this is a strong assumption, oracle query is not necessary at each time step. However, empirical results show that a significant number of oracle queries are necessary for a model to converge to a sufficiently good model.

In summary, MEDUSA presents an interesting concept to tackle the hard challenge to derive model elements T and O automatically by a robot. Yet, learning is limited by necessary initial model knowledge, the size of the hypotheses model set and resulting policy computation complexity as well as a significant number of oracle queries.

A similar framework for learning non-stationary models, Bayes-Adaptive POMDPs is presented in [118]. Yet in contrast to MEDUSA, which is based on selecting from a set of standard POMDP models and has to query oracles, Bayes-Adaptive POMDPs incorporate learning more fundamentally into the model structure. Experience vectors θ, representing frequencies of all transitions encountered so far by the robot and ψ, representing encountered observations, are fused with the basic state space. This leads to a meta-state space $S' := S \times T \times O$ which is countably infinite. A policy can be computed for a finite approximation of S'. It is shown how experience vectors can be obtained and approximately optimal policies can be computed online. However, computational effort for solving realistic scenarios is immense and the lookahead planning horizon remains very small even for simple missions. Real, generic robot scenarios have not been discussed.

Learning transition models for complex, factored fully observable MDPs is discussed in [149]. Conditional probabilities are obtained incrementally by reinforcement learning. Experiments show the suitability of the approach up to state numbers typical for models of robot missions.

However, a high number of learning steps is necessary and no robotic application has been discussed.

In summary, online model learning methods are only sparsely investigated. Some prior model knowledge, for instance of state and action space as well as definition is usually necessary, and learning inefficient in realistic problem domains. Some ideas and insights are nonetheless valuable when considering other types of learning like Programming by Demonstration.

2.2.8. Human Probabilistic Decision Making

When considering autonomous, abstract level decision making for multimodal service robots, a comparison with human intellectual abilities is inevitable. A long-term goal is to reach human performance in everyday tasks, yet technical paradigms to reach those abilities may or may not differ strongly from human intelligence. Therefore, in robotics research some schools of thought try to imitate reasoning methods in the human brain closely, while others follow a more formal or engineering-guided paradigm. While POMDP model-based decision making can be considered to be of the latter type, recent cognitive science studies show compelling evidence for high-level reasoning in the human brain using similar techniques.

Evidence for use of inference on Bayesian causal models in human reasoning about action selection is presented in [51]. Experiments with test subjects support the theory that human decision making can be described accurately by inference on Bayesian graphical models, which is closely related to POMDP decision making. Crucial properties include action-effect probability-model knowledge, reasoning by means of conditional probabilities and assessment of related rewards. A model of Bayesian learning in human cognition is given by [75]. Experimental results support the theory that structural domain knowledge of humans is acquired by Bayesian statistical learning. Such a concept is related to techniques used to learn factored POMDP representations and model knowledge in general. Further insight into human decision making in the face of uncertainty can be achieved by studying carefully crafted decision-making experiments with test subjects as given in [85]. Among different candidates, a generative Bayesian inference model serves best to explain human choices. Generally, Bayesian models seem to be well suited for modeling abstract-level human cognition [52] as well as human sensorimotor control [164].

Consequently, POMDP decision making, which is basically a specific way to handle Bayesian reasoning on real-world information with uncertainties, is closely related to human abstract-level decision making according to current cognitive-science research. Such a paradigm thus seems to be suitable for handling complex behavior up to human-level intelligence.

2.2.9. Conclusions about (PO)MDPs

POMDPs are a sound, formal framework to model decision making of autonomous, artificial systems coping with real-world uncertainty in action effects and environment observation. Experimental evidence supports the theory of humans using similar reasoning techniques for abstract-level decision making.

In the domain of autonomous robots, approximate mission models and algorithms computing approximately optimal action selection policies have proven to be of superior performance compared to other paradigms. However, these algorithms need an explicit, prior POMDP model for policy computation. Manual model generation is infeasible for autonomous systems covering multiple skill domains while performance of online learning methods has not been convincing. Subsequently, moving model generation to an interactive, highly specialized learning process with further refinement steps is preferable. Programming by Demonstration as discussed in Section 2.4 can serve that purpose.

In general, moving a lot of computational effort into an offline stage to generate powerful execution-time classifiers like universal decision policies has also proven successful in other areas of intelligent systems. Such an example is the well-performing human motion tracking algorithm for the Kinect sensor [138]. Massive offline learning of very complex, but fast classifiers outperforms any more complicated online-time tracking method with no or little prior learning.

Specifically, the process described in Sections 3 and 4 is designed based on these insights.

2.3. Modeling States, Actions and Uncertainty in Robotics

Some POMDP specific approaches to model states, actions and uncertainty in robot application domains are discussed in Section 2.2.6. Beyond POMDP applications, a multitude of paradigms exist in the literature. Some techniques are applied in the system presented:

- Clustering to generate abstract, discrete state descriptions from samples in continuous domains.
- Flexible manipulation action representations to model discrete, abstract actions as bundles of trajectories in continuous domains.
- Observation uncertainty processing for object localization to model partially observable real settings.

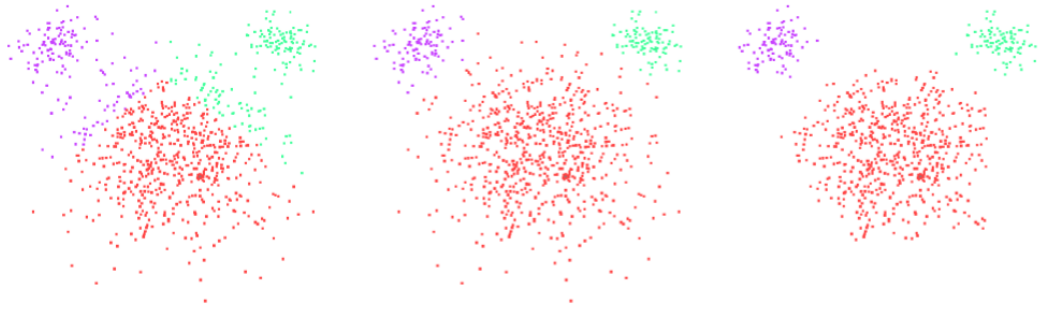


Figure 2.18.: Comparison of clustering methods, k-means on the left, EM at the center and DBscan on the right, on the standard data set "mouse". - [147]

2.3.1. Clustering Methods

Clustering methods can be used to group sets of points in continuous space into conceptual classes as illustrated in Figure 2.18. Application within the scope of PMPM-PbD clusters human and object pose recording data points into conceptual sets from which discrete states and corresponding grounding can be derived. There are two ways to use clustering: determining which points belong to which cluster with a fixed number of clusters k given on the one hand and automatic identification of both cluster number k as well as point membership on the other. Clustering methods are discussed further in Section D.2.

2.3.2. Representations for Actions Considering Uncertainty

Object manipulation actions, encompassing bundles of trajectories in cartesian space, configuration space or more abstract task spaces can be represented by Gaussian Mixture Models (GMMs). It is suitable for both modeling information uncertainty concerning a set of trajectories as in imitation learning discussed in Section 2.4.1 as well as modeling stochastic motion controller behavior by GMMs. GMM representations of trajectories are further explained in Section D.3.

2.3.3. Object Localization Uncertainty Representations

Object localization is a vast field of research in machine vision and robotics. For the system presented in this thesis, localization of objects for manipulation by onboard robot stereo cameras and depth sensors is most relevant. Furthermore, explicit representations of object localization uncertainties have to be computed for use by probabilistic decision making. While any real-world object-localization procedure leads to pose and type detection uncertainties, thorough treatment of uncertainties is insufficient in many systems.

However, some techniques exist which detect poses and types of objects robustly in noisy depth sensor data while computing explicit confidences for pose- and type-classification data delivered. A thorough probabilistic process chain to detect furniture objects in highly noisy Swissranger time-of-flight depth sensor data is presented in [156]. An approach to learning and locating large objects in dense 3D laser data point clouds is presented in [121]. It gives probabilistic estimates of type and pose. Furniture localization confidence estimation, presented in Section 3.3.4, is based on these insights.

2.4. Programming by Demonstration

Transferring knowledge concerning tasks into the information processing system of a robot is challenging for any algorithmic technique providing (semi-)autonomous action selection capabilities. Simple, manual programming methods like direct programming by a domain expert or textual programming by a robot expert fail for autonomy of service robots in complex domains. Instead, machine learning is the paradigm of choice, constructing model structures or parameters from algorithmic analysis of environment observations.

Two main paradigms can be distinguished for robot learning: autonomous exploration of the world with incremental model refinement on the one hand and learning from a human teacher on the other hand. Some methods like MEDUSA, discussed in Section 2.2.7, can be considered as hybrid methods. Learning by autonomous exploration does not need human interference, while learning from a human teacher can be far more efficient. Both paradigms have been explored thoroughly in many different areas of robotics. In conclusion, the more complex the application domain, the better learning from human teachers performs in comparison to fully autonomous exploration. Autonomous manipulation and abstract decision making covering multiple skill domains are two complex problem domains where autonomous robots have to perform tasks in typical human centered environments.

Both complexity of tasks and readily available human expertise have lead to a teacher-based learning paradigm called *Programming by Demonstration* (PbD) [35]. Its main idea is to observe natural human demonstrations of tasks with a sensor setup, analyze these observations and finally derive a task model which can be used by a robot for autonomous execution of that task as sketched in Figure 2.19. Additionally, a human teacher may give explicit verbal or motion-based hints simplifying observation analysis. Humans usually have extensive implicit and explicit task knowledge in domains that are of interest for service robots with multiple skill domains. However, encoding such knowledge in machine-usable form is extremely difficult

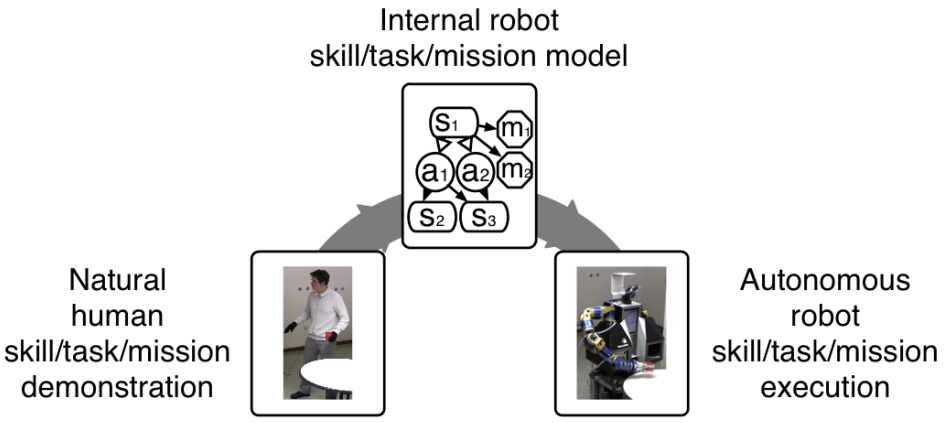


Figure 2.19.: Schematic view of Programming by Demonstration (PbD). -

even for robot experts. Instead, deriving relevant model properties from algorithmic analysis of human task execution is feasible. In summary, the intelligent technical system tries to build its internal representation automatically from a natural expression of human domain knowledge instead of a human trying to shape internal, sometimes subconscious task knowledge into a technical representation.

While PbD is just a general paradigm, many different specific techniques have been investigated for various robot application domains. Depending on the problem, sensor setups and thus observation precision may differ. Accordingly, task representations depend on the application focus as do analysis methods.

Classification of certain techniques is most distinct concerning level of abstraction as discussed in Section 2.1 and concerning level of autonomy. Some techniques put a focus on simplicity and precision, adapting less well to new situations, leaving the robot with less flexibility and autonomy. Others perform more complex analyses to generate task models that provide a high degree of autonomy. Prominent techniques and their focus are discussed in the following.

2.4.1. Probabilistic and Dynamic Manipulation-Level Imitation Learning

Manipulation-level PbD, in specific instances often called imitation learning, centers on the acquisition of motion knowledge for robot manipulators from the observation and analysis of human manipulation demonstrations. Manipulation-level imitation learning has the aim of generating compact skill representations that can be used to execute desired manipulation skills by a robot. Thereby, a skill encompasses a set of very similar manipulation motions that have a certain effect on a set of objects. Such a representation should be able to adapt the execution of a skill to slightly different execution scenes. However, concerning scalability, many typical imitation-learning methods are quite limited. Further desirable attributes of imitation learning

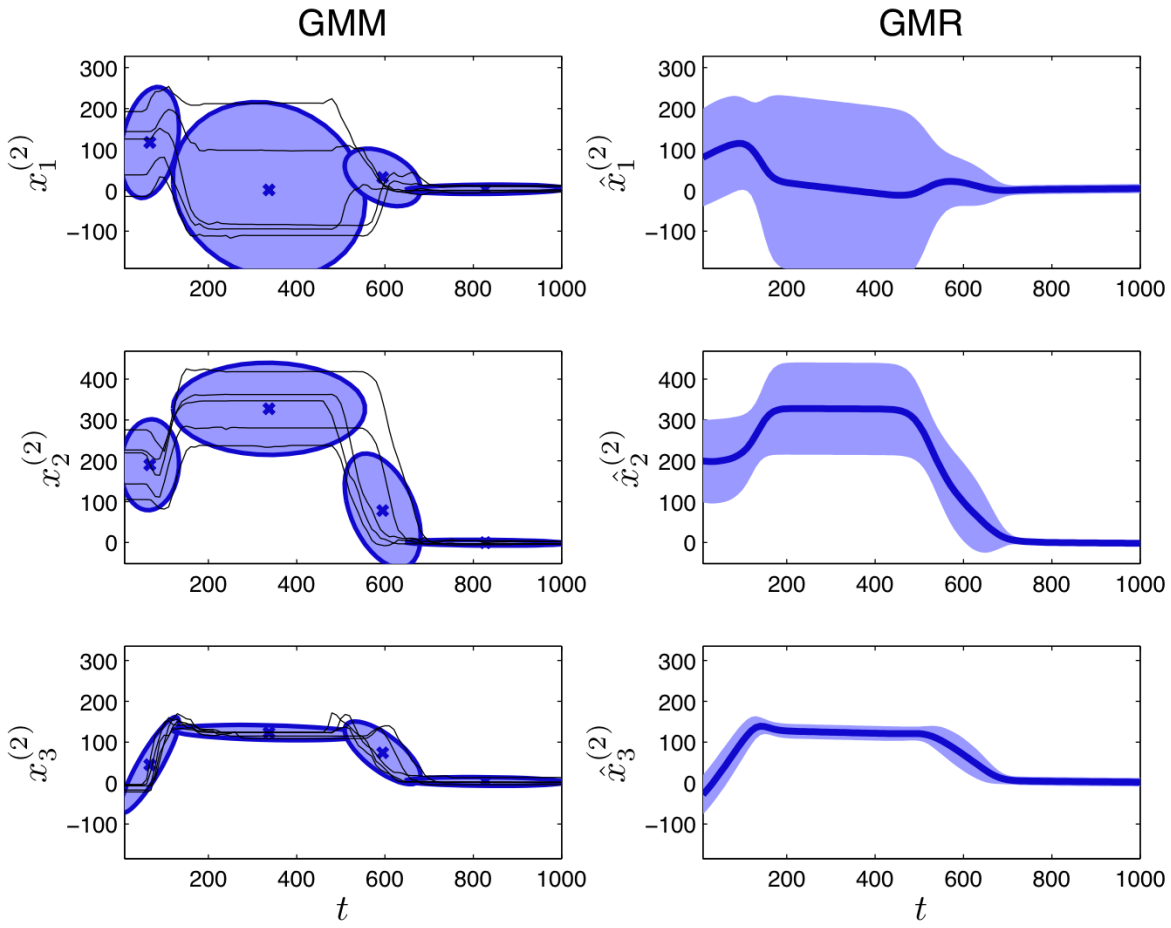


Figure 2.20.: Gaussian mixture model (GMM) and Gaussian mixture regression (GMR) representations of a manipulation demonstration trajectory bundle. - [23]

methods are the need for only a small number of human skill demonstrations, size of the internal skill representation, little processing time for execution and a *naturalness* of the resulting robot motion. Such a naturalness criterion encompasses trajectories and joint velocities that are labeled *human-like* by interacting persons. While that latter quality is hard to define apart from questionnaire studies, it plays an important role in motion-level imitation learning research.

Most manipulation-level imitation learning research uses high-precision, specialized demonstration observation setups. These can be visual, marker-based tracking systems which are able to record human motions with sub-millimeter precision and at high frequency. Another option are exoskeleton, play-back type recording systems, where humans actively push around passive manipulators. Robot-based, marker-less machine vision techniques are currently insufficient for most motion-level imitation learning domains.

Concerning compact representations of motion skills, two types have been most prominent in recent investigations: probabilistic imitation learning on the one hand and dynamic imitation

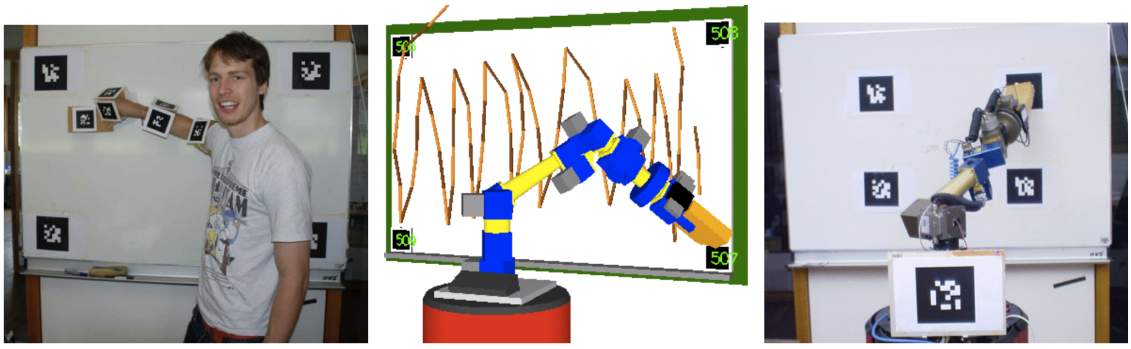


Figure 2.21.: A setup of learning manipulation skills by Dynamic Bayesian Network (DBN) skill models. - [43]

learning on the other.

A probabilistic imitation learning approach is introduced by [13]. The aim is to learn a representation of trajectories, optimally reflecting a set of demonstrated trajectories when executed by a kinematic robot system. The imitation learning process has to compute control parameters that optimally generate execution trajectories. Optimally is defined as minimizing a distance metric between the set of demonstrated trajectories and trajectories generated by the controller. Representing such a metric during the learning stage is achieved by a linear combination of likelihood-weighted constraint cost functions on simple controllers. Such a simple controller could be minimizing a certain joint constraint, for instance. The likelihood gives the probability that this specific simple controller takes part in generating the optimal trajectory. Because the execution strategy is defined by weighted likelihood controllers, it is a probabilistic method. Accordingly, probabilistic learning methods have been applied to compute likelihoods from demonstration datasets. In that given work, hidden Markov models (HMMs) were trained to reflect likelihoods among more abstract controllers.

A more generalized framework of the probabilistic imitation learning approach is presented by [23]. This framework is focussed on multi-joint robot manipulators, like robot arms. Hereby, the motion controller likelihood distribution is represented by a Gaussian Mixture Model (GMM) which is both valid for task-space (e.g. cartesian relation between objects) as well as joint space (e.g. arm motion trajectory) constraints. Graphically (see Figure 2.20), the Gaussians of the GMM form a kind of tube that spans demonstrated trajectories. Means of Gaussians reside in the center of those tubes as do typically generated trajectories. Extensions exist to consider lokal obstacles with trajectories being sampled from unobstructed parts of the GMM.

Another, similar technique, discussed in [43], also uses values representing constraints both in cartesian space as well as joint space within the scope of a probabilistic framework. Yet, this

framework utilizes Dynamic Bayesian Networks (DBNs) for skill representation. Constraint values are represented as observable variables within the DBN that can generate robot execution actions. In the given framework, all variables have been assumed to be represented by Gaussians, leading to a similar representation compared to the approach discussed previously. In the learning stage, mean and variance are computed from analysis of the corresponding constraint during sets of human skill demonstrations. Larger variances lead to larger allowed variances in task execution concerning that specific constraint variable. In each time step, joint configurations are computed from the DBN that maximizes the corresponding likelihood. Obstacles not present during demonstrations can be considered when including additional constraints. However obstacle configuration covered by this is limited to compared to planning-based techniques. Figure 2.21 shows a setup used for learning with this approach.

Further insights into probablistic imitation learning are given by [136]. It presents an architecture for learning probabilistic models in the form of HMMs serving both as an explanatory model for demonstrations and generative model for skill execution. Likelihoods of certain joint configurations following certain scene states for successfully executing skills are derived from analysis of demonstrations. Results were evaluated using simple skills on a small, humanoid robot.

Another compact representation of qualitatively similar sets of motions that can be used for learning from demonstrations and then robot-movement generation is called *Dynamic Motion Primitives* (DMPs) as introduced in [123]. Like probabilistic controller representations that can generate motions in joint space, DMPs are compact, mathematical descriptions of joint space controllers. Individual DMPs can be parameterized and combined to generate any desired robot movement. Like the GMM representation, it has the advantage of using compact, homogeneous basic elements for simplicity and combining multiple of elements to achieve descriptions of arbitrarily complex motions. In contrast to probabilistic approaches, the formulation of DMPs is based on the description of dynamic systems and thus differential equations. Therefore, a DMP can easily describe a typical motor control problem.

Imitation learning can be used to learn relevant DMP parameters. As DMPs are especially suitable for highly dynamic motions, learning such skills as, for instance, hitting a ball with a tennis racket or bipedal walking has been demonstrated successfully. On the other hand, sequential object manipulation tasks with complex obstacles are more difficult to solve with this approach.

Both probabilistic and dynamic imitation learning are PbD paradigms that are centered around fast, human-like execution of robot manipulation skills. Advantages are compact represen-

tations, simple learning even from few demonstrations, fast execution planning and resulting motions that feel familiar to interacting humans.

On the other hand, there are several common disadvantages. First of all, these sub-symbolic representations are focussed on robot manipulator motions and thus do not include other skill domains. Considering complex manipulation skills, scalability of learned skills onto scenes with greatly differing obstacle configurations or objects to be manipulated is usually very difficult to achieve. Combining several learned aspects to motions with completely new qualities is also not straight forward. Probabilistic methods usually do not guarantee that motions occur within certain, hard constraints which lowers robustness within real robotic tasks.

Therefore, the techniques presented are quite unrelated to the objectives presented in this thesis, yet since they are also PbD techniques, some inspiration can be gained from them.

2.4.2. PbD of Abstract Sequence Descriptions of Tasks

In contrast to sub-symbolic imitation learning techniques, more abstract representations that can be generated from analysis of observations are heavily based on discrete sets of symbols. Symbols are used to reference certain world states and robot actions as defined in Sections 2.2 and 2.1.2. It should be noted that other overall PbD classification schemes exist. A classification into *Mapping Function*, *System model* and *Plans* is given in [5]. The classification as discussed in this section does not directly map onto that classification which has a different emphasis. The distinction between system model and plans is especially unclear in that survey for MDP-based models.

The simplest form of abstract representations are fixed action sequences. More complex representations include finite state machines (FSM), partial order plans and hierarchical representations.

An early system to generate simple sequential, abstract task descriptions by user interactions is discussed in [36]. In that system, the demonstration method is still limited to user interaction on a computer, but it shows ways to organize task information into abstract blocks.

Natural human demonstrations of manipulation tasks were used as input for the system described in [168]. World-relative movement of both human hands as well as finger angles are recorded by data-gloves with more than 10 Hz. In irregular intervals, snapshots of object poses in the scene are made with cameras around the scenes. In contrast to imitation learning approaches and a technique using exactly the same setup developed later (see Section 2.4.3), this detailed information is not used to learn trajectory-level representations, but an abstract, hierarchical task description.

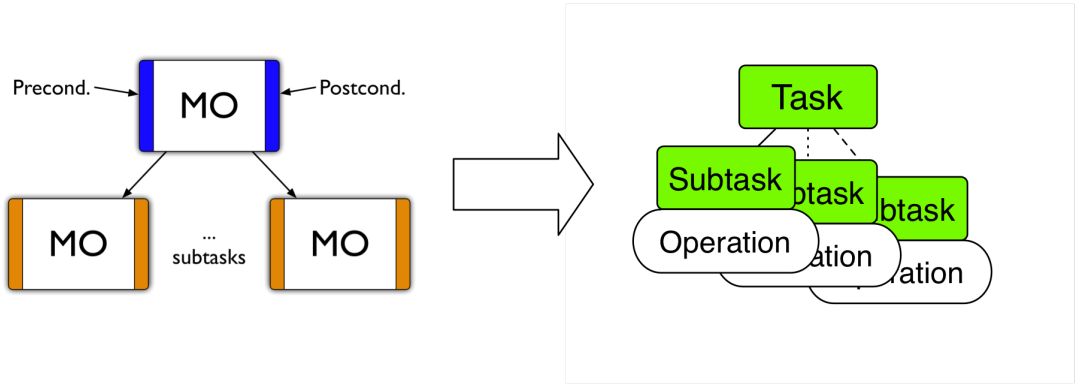


Figure 2.22.: Scheme of transforming the intermediate macro operatior (MO) representation into a robot executable flexible program (FP). - [76]

To achieve this, the learning system *IPOR* first performs a segmentation process on the data time series. A characteristic time point (e.g while picking a cup: between the approach motion of the hand and the grasping motion by the fingers) can be detected by threshold analysis on different data sources. These data sources are provided by the observation mechanism, for instance for picking a cup hand velocity and finger joint angle velocities. When the approach motion ends, the hand velocity drops and subsequently finger joint angle velocities increase when the grasping motion starts, leading to a segmentation point between approach and grasp elements. Such segmentation point characteristics can be trained for different situations using dynamic Bayesian networks (DBNs). Additionally, the segmentation process provides a rough classification of basic abstract task elements.

After segmentation, a sequence of basic, discrete, abstract blocks reflecting a demonstrated task is available. These blocks are the most basic elements available and thus called *elementary operators* (EOs). Each EO can be characterised further by more detailed classification of demonstrated motion and relative object configuration in the scene. Furthermore, the sequence can be organized into a hierarchical representation, composing associated groups of EOs into so-called *macro operators* (MOs). These form branches in a tree with a root representing a demonstrated task.

Such a hierarchical representation permits merging of smaller, learned tasks into larger ones and also some limited form of branching. Yet, the IPOR-MO does not allow for full scale planning and does not consider uncertainties in action execution and scene observation. There is also little trajectory-specific information contained in the EOs. While the first aspect is dealt with in this thesis, the latter aspect is solved by learning manipulation strategies (see Section 2.4.3).

An approach to extend the MO paradigm towards partially ordered plan sequences is presented in [102]. Variances in subtask order can be inferred from multiple demonstrations and the MO representation is extended accordingly.

Retrieval of executable programs from MOs can be achieved by generating so-called *Flexible Programs* (FPs) as shown in [76], sketched in Figure 2.22. These executable programs have the same hierarchical properties as MOs, yet contain some additional information. Leaves in an FP tree correspond mostly to EOs, thus skill parameters have to be added in a way that a leaf can be executed by the robot. Furthermore, MOs usually do not contain information about active environment observation, e.g. looking for objects. Thus, such information is added by a rule-based process while generating FPs. As a standalone method, FPs suffer the same limitations as mentioned for MOs, yet are very suitable in a layered system with POMDP strategic decision making on top and manipulation strategies executed by leaves. Such a layered system is discussed in Section 3.

Another layered system with basically two PbD layers has been presented in [99]. On the lower layer, probabilistic imitation learning of elementary skills takes place as discussed in the previous section. Individual skills are stored in a *Movement Primitive Memory* and can be utilized by the higher layer. That layer basically learns abstract sequences of primitives that can be stored in a procedural memory. During execution such sequences can be replayed.

Apart from natural human demonstrations, GUI-based programming of abstract sequences is also an option as demonstrated by [40]. Basic, elementary behaviors can be assembled into larger programs by a user.

In [137], a more general game scenario is discussed. Thereby, PbD is assumed to take place in a setting where the human teacher takes part in game situations in which an artificial agent is also actively involved. Therefore, the learning system does not only have a passive observer role during learning, but execution and learning are more closely coupled. A teacher is described as an actively involved mentor who is not necessarily helpful. Within this setting, the agent creates an abstract, probabilistic MDP representation of the game dynamics and aims. The aspect that relates most to some techniques described in this thesis is the option of the agent asking the mentor for help. This is related to the assessment of model aspects for relevance and interactive demonstration requests as discussed in Section 4.8.

Learning some aspects of human demonstrations on the most abstract, strategic level for control of a service robot covering all relevant skill domains is discussed in [152]. Concerning the level of abstraction and area of application, it is therefore one of the works most closely related to the system discussed in this thesis. In that system, KNOWROB, PROLOG is used

as a representation of knowledge and inference as a way to help planning and decision making. Examples of such knowledge are regions in which a human stands when grasping certain objects or elements of the process of laying the table [153]. Geometric information from human demonstrations is encoded in mappings to certain relations and these in turn can be represented in PROLOG predicate logic. This aspect is related to the state mapping concept, discussed in Section 4.2.

As shown, there is a variety of approaches for learning abstract task representations by means of PbD. Representations that do not allow true planning are, however, limited in their application flexibility.

2.4.3. Programming by Demonstration of Planning Models

Learning planning models from observation allows even greater execution-time flexibility, as planning models are superior to most other representations in that respect as discussed in Section 2.1.2. It should be noted that the definition here is the one used in the planning literature, including motion planning, logic-based planning and model-based (PO)MDP planning, in contrast to the definition used in the PbD survey [5], which accounts only for logic-based planning.

On the sub-symbolic manipulation level, an approach that creates constraint-based motion planning representations is described in [67]. As this technique is closely integrated with the system discussed in this thesis, it is explained in more detail in Section 2.7. In summary, that technique shows superior ability to generalize over differing scenes and objects compared to paradigms of imitation learning of motion primitives.

Learning abstract-level plans from human demonstrations for a small humanoid robot is described in [158]. Although associated basic skills are very simple, interesting tasks sequences are learnt. Each demonstration sequence is analyzed for action effects, resulting in logic-based planning style predicates for each action. Analysis is centered around the state of objects with which the robot interacts, and the action skill performed for interaction. Therefore, resulting action predicates also focus on changes of object states. These action descriptions can then be used during execution time to plan action sequences which have not occurred during demonstration. Generating such a plan model therefore allows for more powerful generalization of observed demonstrations than simpler sequence representations.

In general, learning planning models from demonstrations enables to derive execution-time behavior that may differ significantly from demonstrated sequences. This flexibility offered by the planning process makes learning of planning models on both motion and abstract level very interesting for further investigation.

2.4.4. Conclusions about Programming by Demonstration

In summary, a multitude of research has shown that PbD is a suitable way of transferring complex task-description knowledge into the information-processing system of a robot. Especially for task environments where robots perform a wide range of complex activities that are very natural to the human, such as everyday household tasks, PbD is the only feasible method. Both motion-level representations as well as abstract, strategic decision-making models have been investigated.

There is a tendency to explore ever more powerful task representations, both to capture more aspects of the task from demonstration as well as to enable more flexible application to new situations during execution. The latter aspect is usually called *generalization capability* because it generalizes a task descriptions to new scenes and situations compared to demonstrated ones. Such features are desired on any level of abstraction. Planning models as defined here – or under the classification in [5] also system models – are most suitable for achieving those ends. Furthermore, on the strategic level, integrating all relevant skill domains of anthropomorphic service robots: autonomous mobility, autonomous object manipulation and natural human robot interaction in a PbD framework has been little investigated.

The system discussed in this thesis aims to achieve progress in these directions.

2.5. Learning from Robot Trials in Physical Dynamics Simulation

Apart from PbD, learning by experience in autonomous environment exploration is a way in which a robot may acquire new task knowledge, as mentioned in Section 2.4. Combining both methods is especially interesting by generating a preliminary task description by PbD and then improving that description by application and evaluation of the task in new situations.

Performing robot tasks in the real world is both slow, difficult to control in a precise manner and very difficult to evaluate concerning task effects. An alternative is to use dynamics simulation for both mobility and manipulation tasks. While such a simulation may not precisely reflect real-world dynamics, setting up situations is fast and simple, all parameters can be controlled and it is straightforward to assess effects on all objects in the scene. Because of these advantages, physical dynamics simulation has become popular for robot learning from experience.

2.5.1. Capabilities and Limitations of Dynamics Simulation for Robots

In contemporary robotics, dynamics simulation usually refers to articulated rigid body dynamics. Such a restriction does not consider flexible body dynamics like compliant soft bodies,

fluids and fractures. While there is research into simulating compliant soft body dynamics for robot perception and action-effect prediction, in the following only articulated rigid body dynamics is considered.

Dynamics simulation takes a scene of objects, each described by a set of parameters:

- Geometric model in the form of a surface mesh.
- Friction model in the form of surface friction coefficients.
- Mass model in the form of a mass distribution.

While executing the dynamics simulation, actuators can be used which exert an immediate force onto certain objects, for instance a robot arm or finger element. Subsequently, objects in motion can exert forces and torques onto other objects they are in contact with. Important aspects of a dynamics simulation are the properties of Newtonian physics: acceleration, impulse, contact points (force contacts), friction between objects and momentum, gravity as well as damping. As with all numerical solutions, dynamics simulation solves computation of parameters in discrete time steps and with diverse discrete approximations of continuum physics. Collision point computation and momentum computation typically is performed by distinct components. The biggest challenge when using existing dynamics simulation solutions is choosing a suitable trade-off between between computation speed and precision, that is approximation of real-world behavior. Furthermore correct parameterization is difficult.

Open Dynamics Engine (ODE) [143] is a mature system especially suitable for robot simulation as it has a well-established framework for modeling special-purpose joints as present in robot actuators. ODE uses a constraint-based framework to compute forces during collision. Simple scenes with a single robot and several rigid objects can be computed in real-time with resulting effects resembling real-world properties. However, exact real-world behavior cannot be replicated that way.

2.5.2. Learning Robot Manipulation Effects

Because dynamics simulation is a rough, but not precise approximation of real-world dynamics, it is especially suitable for computing probabilities for coarse types of action effects as discussed in Section 4.12.

A technique to learn transition probabilities for arbitrary object manipulation in a tower-building robot setting is presented in [82]. An MDP-like representation is generated, which can

then be used by the robot during execution-time decision making. The coarse-grained nature of effect categories makes learning from a limited number of trials feasible.

2.6. Description Logic-based Background Knowledge for Service Robots

When working with abstract-level planning models and especially considering the aspect of learning them, the need for techniques arises which organize knowledge represented in such plans further. Within planning models, both logic-based and MDP style, information about states and actions is limited to just the aspects needed for planning. Aspects which become interesting when incrementally improving models, comparing models or creating new models from parts of existing ones, may be missing from the planning models themselves. One of the most important aspects in this context are relationships and similarities between states as well as actions. Using similarity metrics, additional information can be inferred for novel states and actions based on information about related, well-known states and actions. This insight leads to the need for a *knowledge base* that is able to store and organize information about states and actions, valid beyond single missions. Such a system has to be able to integrate newly learned knowledge incrementally, while providing an elaborate interface for information query.

Description Logic (DL) [9] is the most prominent methodology for organizing symbolic, abstract knowledge in a way usable for further reasoning. The framework provides techniques to model relations, to add new information incrementally and to query the knowledge base by various means. Basic knowledge hierarchy is formed by an ontology, a hierarchical structure of concepts, with the most general concept $\top$. These concepts are much more general and flexible than states and actions in planning models. In fact, states and actions may be concepts somewhere *within* a DL ontology. Such a conceptual hierarchy is created by so-called terminological axioms *TBox* that describe concepts and their roles. Roles can include data properties. Instances of concepts are described by assertional axioms *ABox*, including concept affiliation and data properties. Typically, new ABox data can be added incrementally, but it is also possible to extend the concept hierarchy by means of the TBox. Knowledge can be extracted using queries, retrieving instances and data properties by evaluating expressions.

In robotics, a DL system is usually used to retrieve further data potentially associated with an observed world state. Thus, during execution time, the ontology is queried with some processed, symbolic items of the current state or action options. The response in turn provides additional information for selecting a suitable action.

OMRKF, a comprehensive DL architecture for a service robot is explained in [151]. In this case, the whole architecture for autonomous behavior consists of the reasoning system. The on-

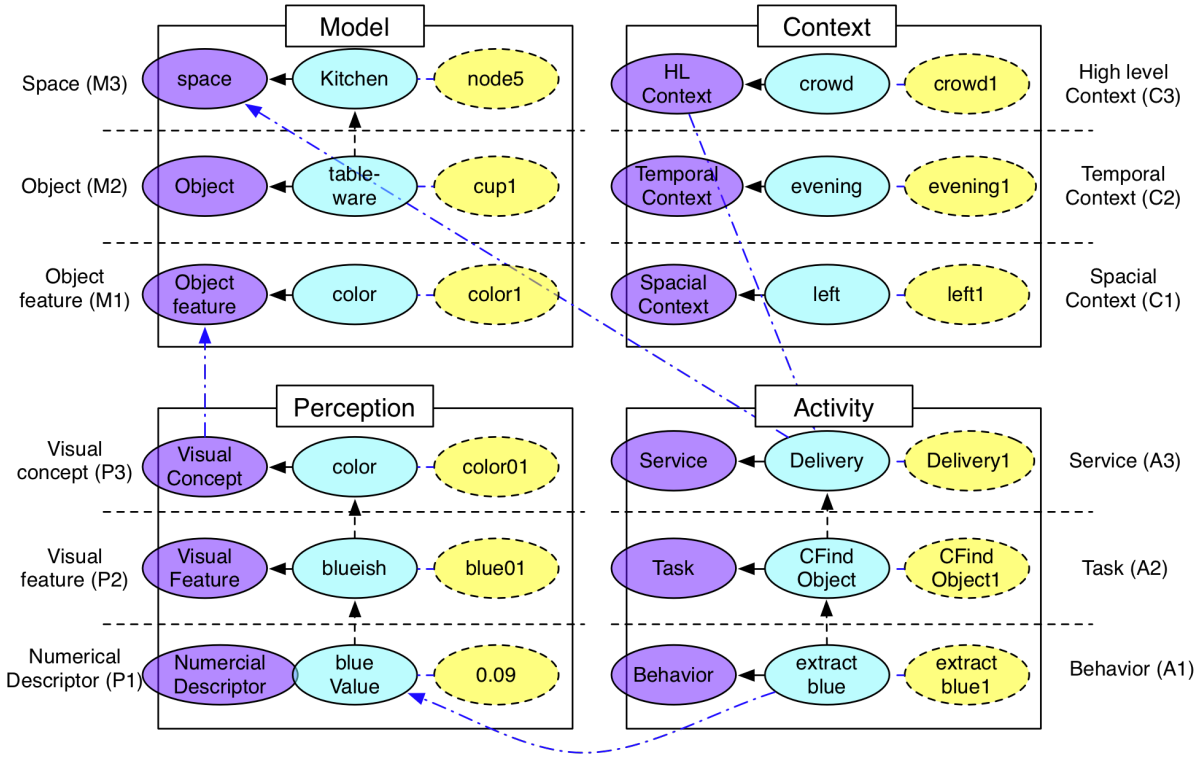


Figure 2.23.: Scheme of the OMRKF framework. - [151]

tology is divided into four levels of knowledge, called *Perception*, *Model*, *Context* and *Activity* as shown in Figure 2.23. Each level in turn is divided into three different layers of abstraction. For instance, on the lowest layer of Perception, image-processing filters are described as DL rules, while the highest layer contains color or texture rules. Accordingly, the execution side Activity level consists of a *Service* layer, a *Task* layer and a *Behavior* layer that consists of very basic control elements. The Model level contains object and scene description rules and the Concept level more abstract spatial and temporal relations. During execution time, an abstract command rule is processed by expanding axioms within the Model and Context levels. These in turn can expand Perception and Activity axioms. In the end, axiom processing generates *if - then* constructs and basic perception data retrieval or hardware control commands which leads to the robot executing a program. However, flexibility and speed of the concept are not discussed in detail and these are known problems with online DL processing remain unaddressed.

The KNOWROB system [152] also organizes all kinds of abstract information within an *Encyclopedic knowledge* base, depicted in Figure 2.24. Three basic types of knowledge are distinguished: *Action models* for reasoning; *Instances of Objects, Actions, Events* which can be selected by reasoning; and *Computable classes and properties*, which are proxies for external algorithmic skills that can be executed. By this arrangement, the KNOWROB system provides strategic mission- and task-level autonomy, while skills are provided by specialized algorithms.

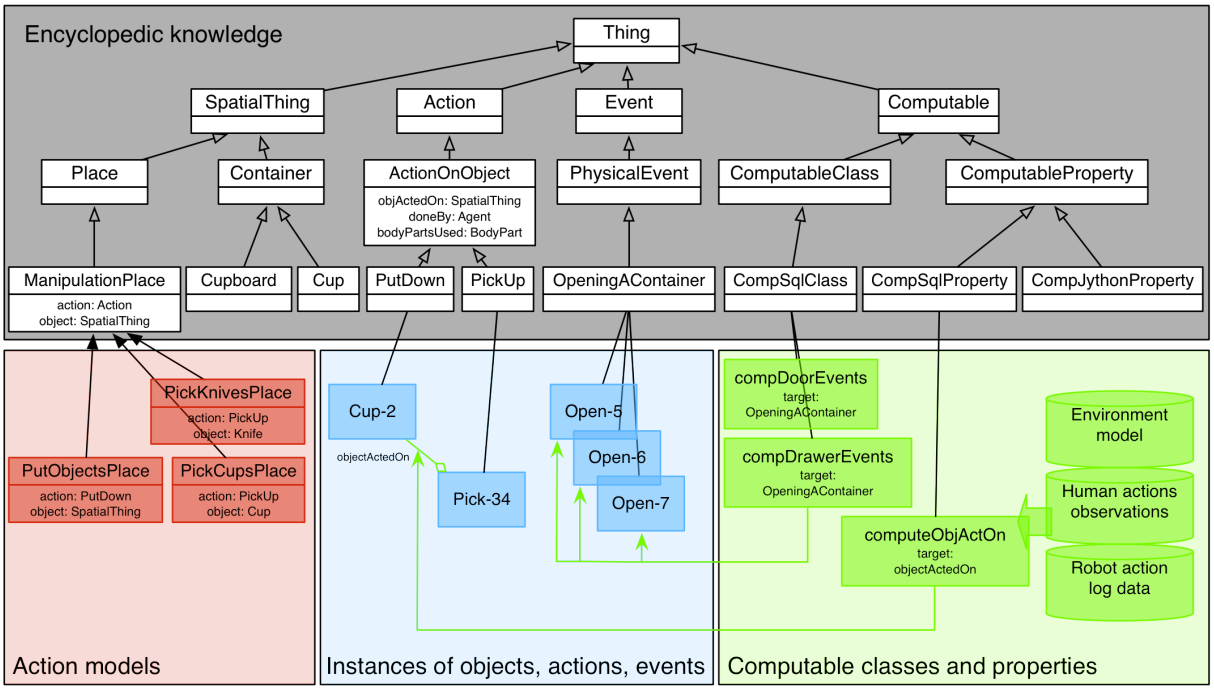


Figure 2.24.: Scheme of KNOWROB. - [152]

Logic axioms representing mission goals can be evaluated until computable elements are generated, providing perception and execution skills.

A more general framework for reasoning is provided by ORO [86]. It is meant to support, but not replace, specialized perception, human interaction and planning. Three layers constitute the ORO architecture: a reasoning *Back-end*, providing processing of axioms; *Modules* which contain basic ontology structures; and a *Front-end* that provides a practical interface to access reasoning from a robot autonomy architecture such as planning. While it has been shown how this architecture can be put to service, its usability depends mostly on well-defined, extensive Modules, which are still little investigated. A typical use is queries during execution time, although the concept may also be used for offline learning.

Further investigations of organized background knowledge have taken place in the context of planning for autonomous underwater vehicles [104], the control of a small humanoid robot [26] and learning grasp affordances [10]. In [100], it is claimed that the importance of ontologies will increase with ever more complex robotic systems, skills and missions as it is currently the most mature paradigm to organize and retrieve such knowledge.

Most systems discussed in this section use an ontology to retrieve information applied to online planning during execution time of a robotic mission. Additionally, most ontologies have a general-purpose design. However, this leads to the systems being exposed to two main dis-

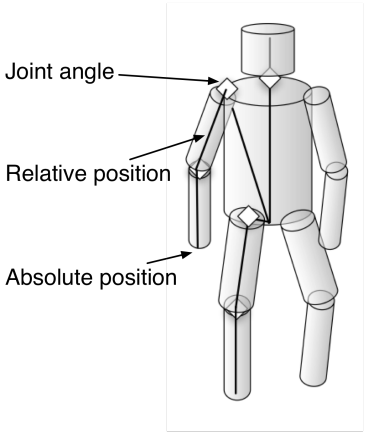


Figure 2.25.: Scheme of several body pose features, considered for activity classification. - [89]

advantages of DL and ontologies in general: inference is not very fast and designing a general purpose ontology including all aspects of a wide range of robot missions does not scale well.

Therefore, the knowledge base of the system discussed in this thesis is used in a different context as shown in Sections 3.7 and 4.10. Accordingly, its main application is highly specialized offline *generation* of planning models, using a special-purpose ontology.

2.7. Skill Components for Observation and Execution

The mission-level decision-making system and the corresponding demonstration-observation process discussed in this thesis utilize various skill components for perception and action execution. Most skill components employ ordinary, widely available methods and their integration will only be shortly explained in Section 3. However, two components have been developed in close cooperation with the system discussed in this thesis, and the component presented in Section 4.3 was developed to create a connection between these two components, which is necessary for an automated PbD process. Therefore, both methodologies will be explained in more detail in the following.

2.7.1. Classification of Observed Human Manipulation Activities

The CHARM system provides symbolic classification of whole body motion activities of humans. Typical activities could be *standing*, *waving* or *grasping a chair*. Classification is based on real-time coarse-grained marker-less perception and tracking of a human skeleton in dense depth sensor data with 25 Hz or more. Originally, the tracking system used was VooDoo [78], processing data from a Swissranger Time-of-Flight sensor [63]. It was later replaced by the Kinect Sensor and its corresponding tracking system [112]. These sensors are mounted on

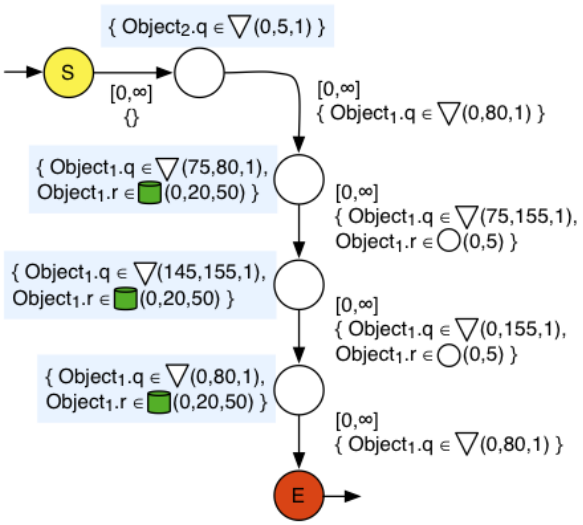


Figure 2.26.: Schematic view of a manipulation strategy graph. - [64]

service robot platforms, thus robot-based observation and classification of human activities is feasible. Optionally, wireless data gloves can be added for interpretation of hand configurations.

CHARM takes time series of vectors containing poses of all body parts as input. From this input, a large set of features is computed, composed of absolute and relative joint poses, joint angles and joint angle velocities, illustrated in Figure 2.25.

Because the total number of features is too high for capable classification methods like *Support Vector Machines* (SVMs) or Bayesian Networks *BNs*, selection of a small subset of only the most relevant features for a certain activity is the most crucial step of the process [166], [90], [91]. First, in a training stage, a user demonstrates a set of activities. These have to be manually segmented in turn. Subsequently, a statistical analysis determines important features for motions within a segment with a relevance above a certain, given threshold. Based on these features, classifier training takes place using manually labeled positive and negative sequences for each classifier.

Applying a certain set of classifiers on a tracking situation delivers probabilities for each symbolic label (classifier) at each tracking time step. This information can then be used to characterize demonstration situations or human-robot interaction with the robot during runtime.

Concerning manipulation activities, classifiers can be trained which represent a single object manipulation action on the abstract, POMDP mission level.

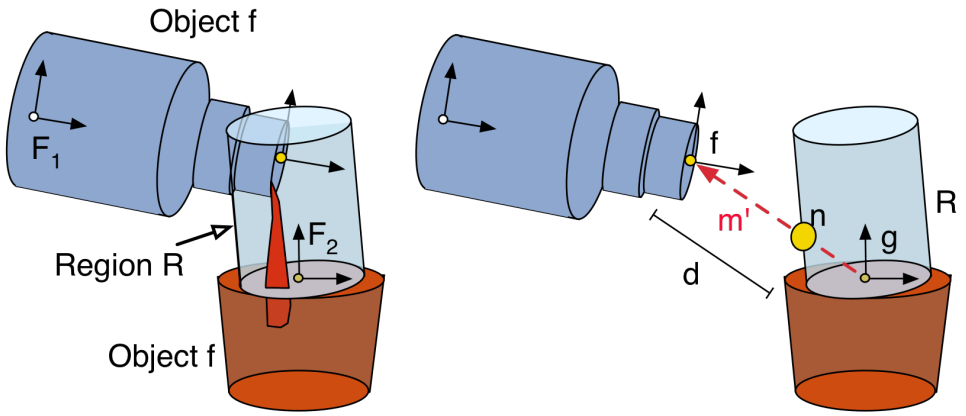


Figure 2.27.: Illustration of geometric constraints (right), based on regions relative to objects (left). - [65]

2.7.2. Manipulation Strategies Representing Robotic Skills

During execution time, abstract object manipulation actions have to be controlled by dedicated skill components. Typical skill control techniques employed for operations like "grasping a cup", "pouring a bottle" or "unscrewing a lid" are motion planning as mentioned in Section 2.1 and controllers generated by low-level imitation learning as discussed in Section 2.4.1. Both approaches have advantages as well as severe limitations alone and thus a system was developed that combines both the advantages of motion planning as well as motion-level PbD [64].

Its central element is a generalized representation, called *manipulation strategy* (MST), of all important aspects of an object manipulation in a way that it can be executed using constraint-based motion planning. This representation consists of a graph formulating a *temporal domain constraint-satisfaction problem* (TDCSP) [65]. Geometric (illustrated in Figure 2.27), force and temporal motion constraints are attached to each node and edge as illustrated in Figure 2.26. An element of an edge constraint could, for instance, describe that the TCP (hand) of the robot has to stay within a certain distance interval of an object coordinate frame while the edge is processed. Motion planning has to find a path which fulfills all constraints of an edge. Nodes represent momentary constraint sets, which are basically goals of parts of the motion and at the same time form the start of the next one.

A pouring motion, illustrated in Figure 2.27 can be represented by a starting node, where the opening of the bottle is within a cylinder above a cup while the axis of the bottle has to be parallel to the ground. The edge has again the cylindrical constraint, but also the constraint to keep the angle of the axis above parallel. The end node keeps the cylindrical constraint, while the axis has to be strongly tilted against the ground with the bottom pointing upwards. The motion planner then has to find a suitable path in a given object setup.

As with a POMDP mission model, a strategy graph for a dexterous object-manipulation action can grow very complex. Because it is then infeasible to define it manually, a set of methods has been devised to generate it by means of PbD. Constraint sets and strategy graphs are generated from analysis of fine-grained observation of human object manipulation demonstrations. The constraints are further refined by several learning mechanisms [67].

Overall, the workflow for manipulation strategy PbD is very similar to the workflow of the system discussed in Section 4. Both perform analysis of demonstrations, generalization and refinement, and finally generate a planning model. However, the level of abstraction, granularity of observation, skill domains, specific methods of generalization and refinement as well as the resulting planning models are totally different. Because of this, both systems are fully complementary and have been integrated, with manipulation strategies providing single, basic manipulation actions for POMDP decision making. Individual manipulation strategies (skills) and missions are demonstrated independently and within different setups. A mapping method discussed in Section 4.3 provides autonomous mapping of human manipulation actions within mission demonstrations onto known, previously learned manipulation strategies. During execution time, a selected manipulation strategy is autonomously executed within the scene, performing collision-free motion planning.

3. Modeling Probabilistic Decision Making by Service Robots with Multiple Skill Domains

Three main aspects have to be tackled for realizing a decision-making system for autonomous service robots:

1. Designing an execution-time information processing system providing autonomy

2. Modeling real-world settings with symbols and quantities

3. Developing a model acquisition process for the robotic system

The review in Section 2.1.3 shows that hierarchical three-layer execution architectures are a capable information processing layout. As argued in Sections 2.1 and 2.2, POMDPs are a powerful fundamental modeling technique for autonomous decision making. Finally, Programming by Demonstration (PbD) is a superb paradigm for model acquisition, as discussed in Section 2.4.

Thus, missions are modeled as POMDPs that are applied to execution time reasoning in a three-layer architecture and acquired using PbD. Modeling and execution-time system are closely coupled, while the PbD process is complementary: in theory, a model could be programmed manually - although in most cases that might lead to an infeasible amount of effort.

Therefore, a detailed description of modeling and the execution-time system is given in this Chapter. These insights establish a foundation for the requirements and actual algorithmic procedures of the PbD process, described in detail in Chapter 4.

First, details of the execution-time information-processing architecture are described and a concept to integrate POMDP decision making on the most abstract level is outlined in Section 3.1. Next, state grounding (Section 3.2), observation uncertainty (Section 3.3) and action scope (Section 3.4) definitions and realization are presented. Then, systematic design of general scenarios is discussed in Section 3.5, followed by a functional expression framework for model compilation in Section 3.6. Finally, a knowledge base overarching multiple mission models is presented in Section 3.7.

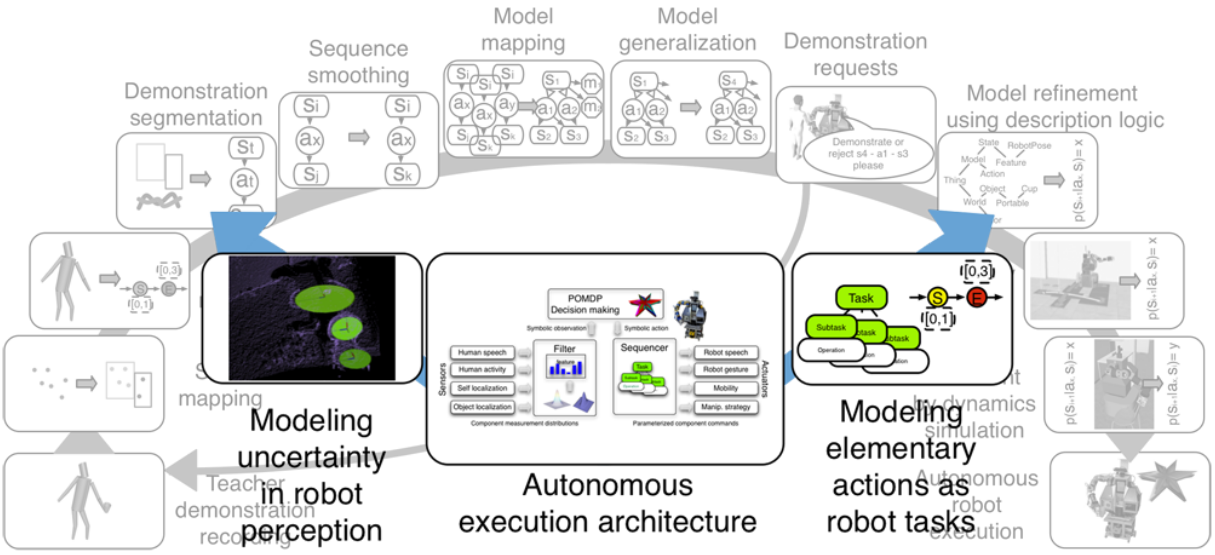


Figure 3.1.: Contents of the modeling Chapter highlighted in the overall scheme. - [125]

3.1. Autonomous Decision-making System Architecture

In the information processing system, raw sensor data is converted into representations used by abstract decision-making models. Selected abstract actions in turn are transformed into commands to physical actuators. A big challenge with robots integrating vastly different skill domains such as natural human-robot interaction (HRI) and object manipulation is to encapsulate sensor measurements in a way that the abstract perception representation in turn is homogeneous. The same has to apply to abstract decisions when converted into actuator commands. On the other hand, information about perception uncertainty has to be preserved to be taken into account by abstract decision making. Such requirements can be accomplished by a three-layer architecture with a filtering layer between sensor and abstract level.

3.1.1. Information Processing Architecture

The information processing system of a typical, autonomous service robot is connected to physical sensors and actuators by electronic and mechatronic components. Normally, a service robot with multiple skill domains has several personal computer type information processing systems on board which are connected to those electronic and mechatronic components [18], [103], [6]. In such a setup, sensors and actuators present themselves to the abstract coordination system as dedicated software components (see Section 2.1.3). Typically, highly specialized algorithms refine observation data and execution control. Coordination and planning thus deals with a highly refined information interface to the real world. Therefore, a robot - even with all physical com-

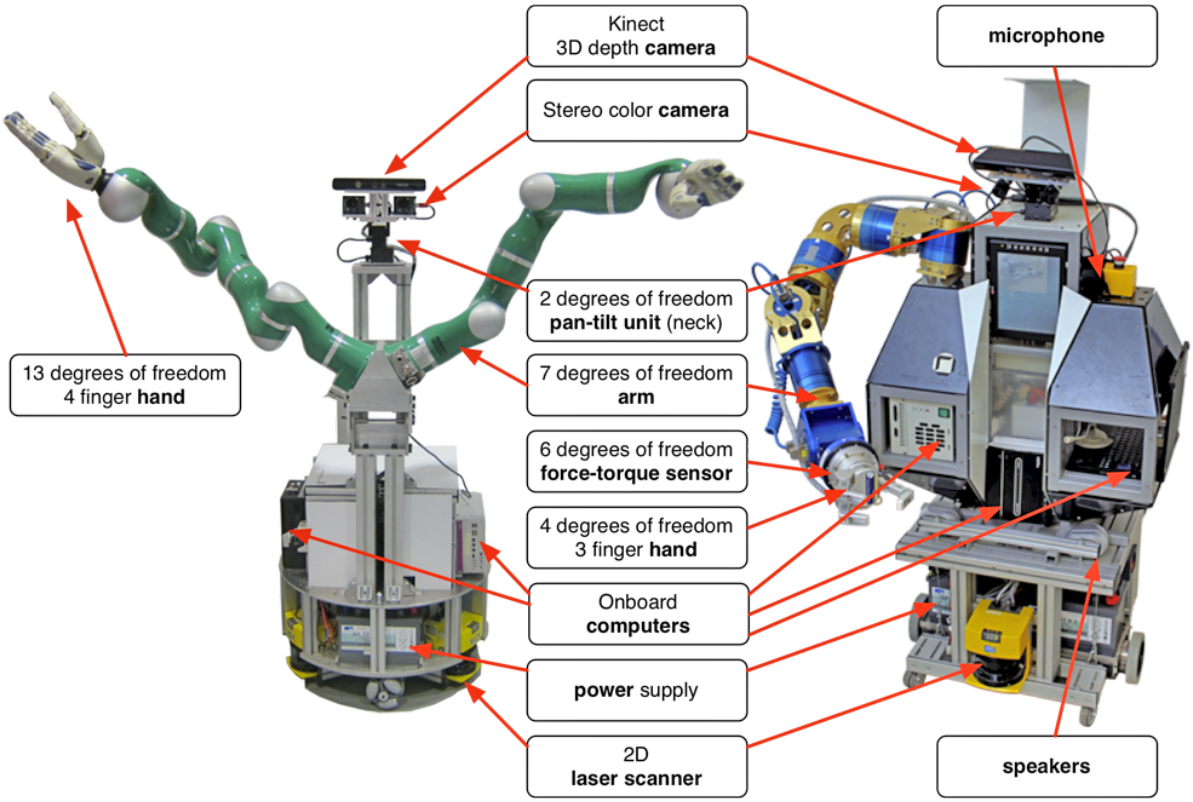


Figure 3.2.: Adero (Advanced DExterous RObot) on the left and Albert on the right with labeled components. - [125]

ponents tightly integrated - is only a loose collection of components and capabilities from the coordination systems point of view.

Albert and *Adero*, shown in Figure 3.2, are typical anthropomorphic service robots and examples of this concept. They cover multiple skills domains and were utilized for evaluation of the system presented in this thesis.

The following sensors are available on these robots as illustrated in Figure 3.2:
3D point-cloud camera; stereo color camera; 2D laser scanner; microphone; force sensors; odometry.

The following actuators are available on these robots as illustrated in Figure 3.2:
Wheel motors; arm/hand joint motors; camera mount motors – pan-tilt-unit (PTU): neck; speakers; screen.

Several onboard personal computers, connected by an onboard network, can access these devices [133]. Robot skills for both perception and actuation in certain skill domains is provided by dedicated, highly specialized algorithmic components which typically access one, or at most a few of these hardware devices. The following exemplary skill domains and components are available on Albert and Adero:

Mobility is provided by self-localization and navigation. Mobility provides perception in the form of a world relative (x, y, θ) robot pose value and actuation in turn moves the robot to a pose identified by such a notion:

Skill component	Input	Algorithm	Loop	Output	Ref
Robot self localization	Laser,, odometry, map	Kalman filter	yes	**World relative pose** (x, y, θ)**, uncertainty** **Cov**(x, y, θ)	
Navigation	**Target** (x, y, θ), map	Path planning	yes	Path trajectory	Sec. 4.11
Motor control	Trajectory	Control	yes	Motor current	

Object Manipulation is provided by object localization, based on cameras shown in Figure 3.3 and manipulation motion planning. Object localization provides world-relative pose information for known object geometries and types. On the other hand, strategy-based motion planning takes robot and object poses as well as a known manipulation strategy and translates it into a collision-free manipulator motion:

Skill component	Input	Algorithm	Loop	Output	Ref
Object localization	3D point cloud camera, stereo color camera	SIFT, Appearance, Shape, RANSAC, etc	no	**Camera relative object poses** (x, y, z, r, p, yaw), **Cov**(x, y, z, r, p, yaw), **confidence** $p(type)$	IVT [8] Furn [96] Sec. 3.3.4
Manipulation strategy motion planning	**Strategy graph**, **object poses** (x, y, z, r, p, yaw) **robot pose** (x, y, θ)	Constraint RRT, TDCSP, collision checking	yes	Manipulator trajectories	Sec. 2.7.2

Human-Robot Interaction is provided by visual human body activity recognition, gestures, speech recognition and speech synthesis. In the skill domain of HRI, most low-level perception components provide quite abstract, symbolic output while actuation components require symbolic input. Thus, abstraction is expressed more strongly here than in the other skill domains:

Figure 3.3.: Camera head of Albert with pan-tilt unit (neck), Kinect 3D depth camera and Guppy stereo color cameras. - [125]

Skill component	Input	Algorithm	Loop	Output	Ref
Human body activity recognition	3D point cloud camera	Tracking, SVM classifier	no	**Limb poses** (x, y, z, r, p, yaw), **activity** $p(\textbf{\textit{type}})$	NITE [112] Sec. 2.7.1
Gestures	**Gesture name**		no	Manipulator trajectory	
Human speech recognition	Microphone	Acoustic models, HMMs	no	**Text (string)**	Sphinx [160]
Speech synthesis	**Text (string)**		no	Sound	Festival [15]

In summary, data delivered by perception components and required by actuation components is too inhomogeneous and still not abstract enough concerning state space, actions as well as temporal aspects to be directly usable by POMDP decision making. Thus, an intermediate layer is necessary between strategic mission level POMDP decision making on the one hand and respective skill components on the other.

This insight leads to a typical three-layer architecture. Skill components provide an interface to the physical world. In the middle, a filter system harmonizes observation data into a unified, abstract situation representation, while selected abstract actions are broken down into actuation component commands. On top, abstract decision making selects abstract actions based on abstract observations and its mission model. This architecture is sketched in Figure 1.4.

There are different possible approaches to compute an abstract situation representation from a temporally and representationally inhomogeneous set of observation data. The technique

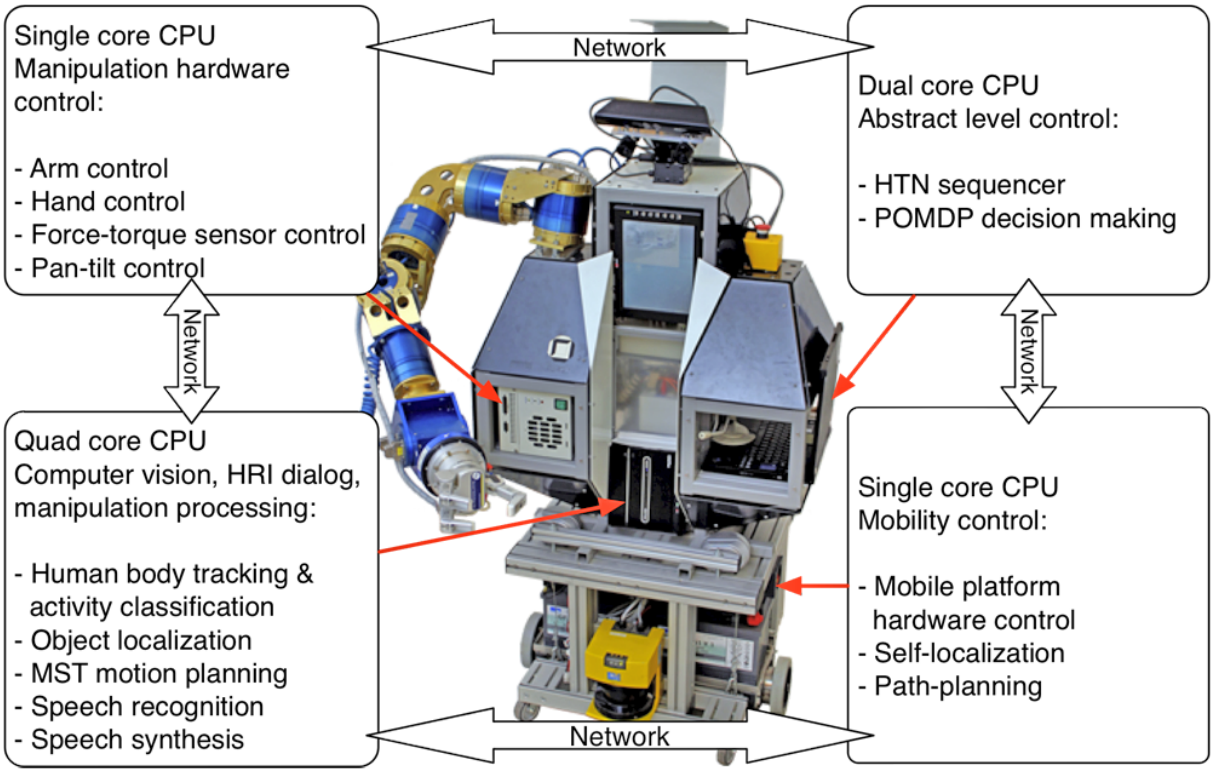


Figure 3.4.: Computers on Albert annotated with associated skill components. - [125]

utilized in this architecture is discussed in the next Section. Usage of *Flexible Programs* decomposing abstract actions into elementary component commands is described in Section 3.4.

In turn, abstract POMDP decision making is embedded into an abstract, virtual robot agent provided by perception and actuation interfaces of the middle layer. By these means, decision making can easily be simulated by providing simplified observations to the abstract agent which is beneficial for testing and evaluation purposes. In the present system, a decision-making policy is computed offline from a model learnt by PbD - or manually designed - using the SARSOP algorithm, presented in Section 2.2.5. That policy is in turn queried to select actions given a certain belief state during execution-time.

Further properties are relevant from a more practical perspective. Individual skill components, the middle layer and the abstract reasoning can communicate with each other in a network-transparent manner, distributed among several computers as illustrated in Figure 3.4. This is an important property since typical service robots have multiple onboard computers. Furthermore, the architecture is designed in a way that new skills can easily be added and the abstract layer can be ported to other robots with different skill components [133]. Therefore, extensibility is provided.

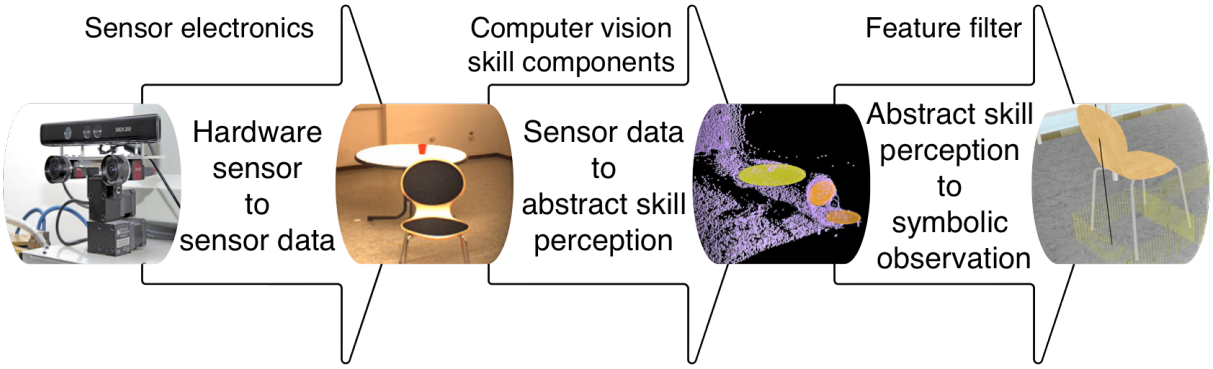


Figure 3.5.: Schematic view of $f_g(E_s) \rightarrow s_i$ based on robot perception. - [125]

3.2. Definition and Design of State Grounding

When reasoning about the world using abstract decision making, symbolic elements representing aspects of the world have to be clearly defined and rooted in the world. This is usually referred to as symbol grounding. Internally, these symbols may have arbitrary denotations, as long as there is a unique, distinct mapping from an environment aspect to such a symbol.

State grounding is basically a function f_g, that maps an arbitrary environment situation E_s to exactly one distinct, symbolic state $s_i \in S$, $s_i := f_g(E_s)$. Typically, aspects of the environment have to be described by continuous properties, thus f_g performs discretization. Different, similar situations E_s^i, E_s^j may map to the same symbolic state: $s_k = f_g(E_s^i) = f_g(E_s^j)$. So, a mapping f_g can be surjective, but not injective. A state therefore forms an abstract class of situations.

From the point of view of applying this approach to a rational agent in a partially observable environment, such as a service robot, both E_s and s_i are hidden variables, though. Yet, the robot is able to get information about E_s by measurements M as described in Section 2.2.3. Because perception frames the reality of the robot by these means, only aspects that can give hints about E_s through M are relevant to be modeled in s. Everything else has to be subsumed without a descriptive model as Bayesian probabilities in stochastic models T and O. Or in other words: the robot is agnostic about aspects in E_s that do not correlate in any way with any measurement $m \in M$.

Therefore, the mapping $s_i = f_g(E_s)$ has to be structured according to perception skills. Those encompass only relevant aspects of the world with which the robot interacts. For example. the specific indoor temperature is usually irrelevant and there is also no sensor for it on most robots.

According to this paradigm, perception skill components have a leading role in f_g and thus state grounding. Such a channel, depicted in Figure 3.5 can also be considered as a *perception*

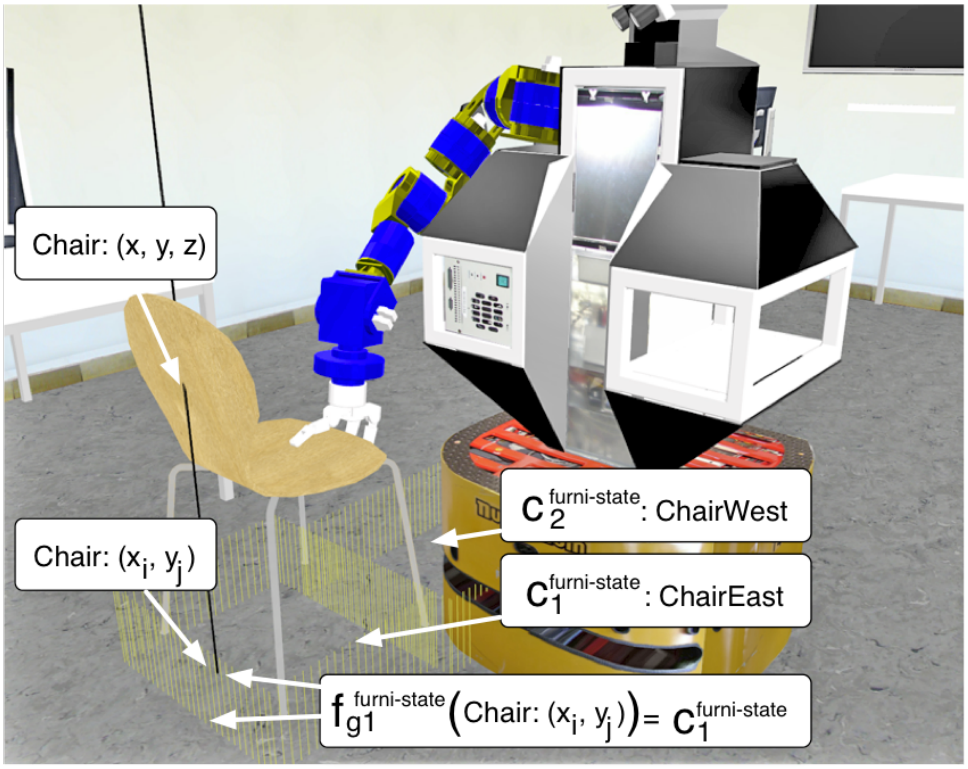


Figure 3.6.: *Illustrative example:* Pose of the chair is mapped to a topological, abstract region $c_{x_i}^{furni-state}$ by $f_{g1}^{furni-state}(E_s)$, illustrated by a yellow fence. $Chair_\theta$ is not shown. - [125]

skill domain from the robot's point of view. These domains form the largest building blocks of the mapping and can be sub-divided into aspects, *features* f_{feat_i} defined here as:

$$f_g : E_s \rightarrow f_1 \times \ldots \times f_{|feat|} \quad [3.1]$$

$$f_g(E_s) = (f_{feat_1}(E_s), \ldots, f_{feat_{|feat|}}(E_s)) \quad [3.2]$$

Hence, each feature can be seen as a component of a multidimensional mapping aspect, each of which may be multidimensional itself and continuous in nature. Two different feature examples can be human verbal utterances and world poses of furniture objects in the scene.

Finally, as the codomain S of the mapping f_g is discrete, individual features have to be discretized by the mapping. For each feature, a discretization function maps a set of situations onto one distinct *category*: $c_j^i = f_{feat_i}(E_s)$. A set of categories defines a feature state space: $F_i := \{c_1^i, \ldots, c_n^i\}$. A complete mapping is of surjective design with feature categories c_j^{feat} defined in such a way such that a mapping exists to exactly one category from each possible configuration of E_s. In turn, a state is defined as a tuple of categories: $s_i := (c_{x_1}^1, \ldots, c_{x_{|feat|}}^{|feat|})$. Accordingly, the overall mapping f_g is surjective with each E_s being mapped to exactly one s_i, yet different, typically similar E_s may be mapped to – and thus subsumed into – one unique s_i.

On a more practical level, perception-skill components are often heavily involved in simplification and discretization of environment property subsets of E_S, speech and dialog processing being a prominent example. Often, those components also account for E_S being a hidden variable (see Section 3.2.2 for a discussion of both aspects). Some components perform simplification and temporal filtering of the state description, but no discretization. In those cases, an additional discretization step has to be performed on the skill-component output as illustrated in Figure 3.6 (see Section 3.2.2 for examples, also discussing the filtering aspect).

In summary, the first step is mapping relevant and measurable aspects of the environment situation E_S onto discrete categories, leading to a feature-discretized state, a vector of categories: $f_{g1}(E_S) \rightarrow (c^1_{x_1}, \ldots, c^{|feat|}_{x_{|feat|}})$. In the second step, a feature-state mapping f_{g2}, (*FSMap*), assigns one unique state to each distinct category vector: $f_{g2}(c^1_{x_1}, \ldots, c^{|feat|}_{x_{|feat|}}) \rightarrow s_i$.

This notation holds only for the fully observable, non-probabilistic case. To transform this concept from a fully observable situation E_S to a partially observable one, probabilistic Bayesian filtering has to be included as discussed in the next Section.

3.2.1. Preserving Uncertainty for Multiple Skill Domains: filterPOMDP

As discussed in Section 2.2.3, in partially observable environments the true state of the world s_t is a hidden variable. An agent acting within such an environment has to derive a subjective belief-state probability distribution b_t, representing knowledge about the true, hidden state, by Bayesian filtering. It combines a probability distribution representing imperfect prediction with a probability distribution representing imperfect environment observation. In discrete POMDPs, both observation and prediction are discrete representations with discrete probability distributions representing uncertainty. In contrast, in many robotic skill domains, Bayesian filtering of continuous variables, such as Kalman filtering or particle filtering is performed utilizing parametric or non-parametric continuous probability distributions to represent measurement, prediction and belief. When working with continuous situation domains and discrete state spaces, this leads to two possible options:

1. Continuous observation and prediction models, discretizing belief distributions.

2. Discretizing observation distributions and applying discrete prediction models resulting in discrete belief distributions.

Discrete POMDP applications are typically of the second type, with a static state discretization mapped onto observation symbols and a static measurement uncertainty probability distribution $p(s|m)$ as well as transition effect probability distribution $p(s'|s,a)$. This model is used

both for policy computation as well as execution-time belief updates. As a consequence of its representation, these discrete distributions are only coarse approximations of real-world dynamics. While any Bayesian probability distributions representing measurement, effect and belief uncertainties are just approximations, highly specialized continuous distributions and corresponding sophisticated, even non-linear filters are better approximations for most skill domains as discussed in [155].

Accordingly, to utilize different, more specialized Bayesian filters, online time belief updates and planning-time belief updates for policy computation are split up in the presented system. The idea is to use specialized filters for individual features *feat* as defined previously, with high spatial and temporal resolution as well as continuous probability representations where applicable for online belief updates. Subsequently, these feature beliefs are discretized and fused into a single belief distribution. On the other hand, abstract-level POMDP observation and transition models, used for policy computation, are approximations of feature filter behavior. Additionally, they introduce cross-feature dependencies into the belief update during fusion of the single belief. By these means, characteristics arising from the sensor-level and algorithmic processing peculiarities leading to specific uncertainty properties can be preserved up to the most abstract belief state. A perception component investigated closely for the transfer of information about observation uncertainty from the sensor level up to the abstract level is discussed in Section 3.3.4. This leads to the definition of *filter*POMDP:

A feature belief may be continuous and updates may use linear or non-linear models. In the following example, x, y, θ are the 2D position and orientation of a mobile robot in a flat indoor environment and $b_{\text{robot-pose}}$ is a continuous belief distribution of such a pose:

$$b_{\text{feat}}(x_t) = \int p(x_t|a_t, x_{t-1}) b_{\text{feat}}(x_{t-1})\, dx_{t-1}, x \in E_s \quad [3.3]$$

$$\text{e.g. } b_{\text{robot-pose}}(x_t, y_t, \theta_t) = (x_t, y_t, \theta_t, \text{Cov}(x_t, y_t, \theta_t)) = \text{Kalman}((x_{t-1}, y_{t-1}, \theta_{t-1}, \text{Cov}(x_{t-1}, y_{t-1}, \theta_{t-1})), u_{t-1}) \quad [3.4]$$

Subsequently, a continuous belief b_{feat} is discretized by applying a mapping $f_{g1,feat}$, leading to discrete category belief probabilities $p(c_j^{\text{feat}})$:

$$b_{\text{feat-state}}(E_s) = f_{g1,feat}(b_{\text{feat}}(\vec{x})) \quad [3.5]$$

$$b_{\text{feat-state}}(E_s) = \{p(c_1^{\text{feat}}), \ldots, p(c_{|\text{categories}|}^{\text{feat}})\} \quad [3.6]$$

$$\sum p(c_i^{feat-state}) = 1 \quad [3.7]$$

There are several options concerning discretization technique f_{g1}, depending on the complexity of the probability distribution representing b_{feat}. In case of simple normal distributions,

discretization can be computed numerically applying a flexible grid as discussed in the next Section. More complex distributions must be approximated by Gaussian mixtures that are discretized, or samples must be drawn from the distribution and subsequently summed up. These more complex options have not been explored in this work.

The feature-state $c_{i \simeq feat_i, j} \rightarrow s_k$ mapping can then be used to compute the state belief:

$$b_{\text{filter}}(s_k) = \prod_{i=1}^{|feat|} \sum_{j, c_{i,j} \in s_k} p(c_{i,j}) \qquad [3.8]$$

Large scale predictive aspects, which are not considered by low-level filtering, typically arising from what is defined as *primary* stochastic effects in Section 3.4.2, can then be added using a transition model $T_{primary}$ by considering the belief b up to this point as an observation distribution. However, further prediction transition probabilities of an action a_t have to be conditionally independent from any predictive element in the low-level filter, otherwise it is included twice, distorting the result:

$$b_{\text{final}}(s_t) \;=\; \alpha \, b_{\text{filter}}(s_t) \, (\sum_{s_{t-1}} T_{\text{primary}}(s_t, a_{t-1}, s_{t-1}) \, b(s_{t-1})) \qquad [3.9]$$

By these means, the simple non-probabilistic feature-discretized model discussed in the previous Section is extended into a feature filter model, retaining grounding aspects while introducing preservation of low-level uncertainty information up to the abstract description [128].

Evaluation of the *filter*POMDP approach is discussed in Section 5.2. While Bayes filters are typically fixed for skill components, discretization is highly mission dependent. Hence, feature state grounding has to be defined specifically for each mission, to minimize the state space and distinguish abstract states where needed. Feature state grounding is therefore one of the properties that have to be learned using PbD for each mission, as discussed in detail in Section 4.2.

Another important aspect ist the temporal connection between low-level $b_{\text{feat}}(x_t)$ updates on the one hand and high-level $b_{\text{pomdp}}(s_t)$ updates on the other. Individual feature belief updates are highly asynchronous with differing frequencies. An abstract-level belief can be updated each time when a feature belief is updated. However, a belief is only relevant on the decision-making level when a previously selected abstract action has terminated.

An abstract level POMDP transition model, including cross-feature effects, thus has to model *overall* low-level effects within the time span of such abstract actions. The scope of such an abstract action is discussed in Section 3.4. Transition models learned by PbD and autonomous refinement as discussed in Chapter 4 consider this aspect.

In summary, filters reflect robot skill component domains and capabilities. They connect the aspects of the world relevant to the robot with abstract symbols, utilized by decision making while preserving information about uncertainty in regard to these aspects arising from incomplete observability. Filters can be re-used for all types of missions, while discretization and state grounding has to be generated for each mission model and policy. These filters define the world as the robot experiences it for decision making on an abstract level. However, new filters and features can be added easily within the scalable architecture.

3.2.2. Design of Filter Models

Design of features and corresponding filter models is driven by available input on the one hand and required feature state space output on the other. Input choice depends on hardware sensors and corresponding perception processing skill components on a robot. Output demands depend on environment aspects with which the robot has to interact with in certain missions. Feature design means to assess both ends and bring them together in a complete scheme:

1. Input: Perceivable numerical values and/or symbols, optionally with information about measurement uncertainty attached. Examples:
   - Object pose: 6D continuous values (+ variance), symbolic type (+ confidence), object geometry (+ uncertainty) - see Section 3.3.4 for an exemple
   - TCP force/moments: 6d vector of continuous values (+ variance)
   - Human utterance: symbolic text list (+ confidence probabilities)
2. Output: Discrete, small, for a mission-relevant meaningful feature state space.
3. Input combination: Either only one value from one skill, several values from one skill or several values from several skills.
4. Filter: Either an existing Bayes filter in a skill is sufficient, an extra Bayes filter is needed or no Bayes filter is used (e.g. predictive element gives no sufficient benefit).
5. Discretization scheme: Discretization either before or after the filter.
6. Actual discretization: Dependent on the aspects important for a mission.

In practice, these steps can be further grouped into three design stages:

1. *What?*: Determining the input data combination available, facilitating necessary output.

2. *How?*: Determining the way to combine input data to be transformed into an output set.

3. *How precisely?*: Determining the exact set and discretization of feature states.

Design steps *what* and *how* are complex, yet rarely performed, as resulting features and filter models are typically reusable for many different types of missions. A resulting feature is just an abstract-level perception routine available on the robot. Thus, automation has not been investigated within the scope of the presented system. On the other hand, deriving *precise* discretization boundaries and the exact resulting feature state space is complicated as well as error prone to be performed manually and is variable for each feature between missions. Thus, this design stage has to be performed for each mission. Hence, a technique was developed to learn feature state space discretization using PbD as discussed in Section 4.2.

In the robot decision-making system used for evaluation, features and corresponding filter models with diverse characteristics are applied. In the following, representative examples show different degrees of input diversity, filter usage and data conversion. Specific modeling of uncertainty regarding these features is discussed in Section 3.3.

Spoken dialog $f_{dialog-state}$ is an example of a feature using highly processed abstract input with attached information about perception uncertainty. Furthermore, application of an extra Bayes filter is suitable. Resulting feature states represent spoken dialog of a human interacting verbally with the robot in an abstract manner. Input to the filter is a normalized, discrete probability distribution over a set of symbolic human utterances m as delivered by speech recognition [160]: $P(m_t)$. Additionally, the last spoken utterance of the robot u determines the predictive effect model $T(s_t, u_{t-1}, s_{t-1})$. An observation model $O(m_t, s_t)$ correlates abstract dialog states s with spoken human utterances m. Processing applies a Bayes filter using these models and current online uncertainty as delivered by speech recognition:

$$b_{dialog}(s_t) \;=\; \alpha\Big(\sum_{m_t} P(m_t)\, O(m_t, s_t)\Big)\Big(\sum_{s_{t-1}} T(s_t, u_{t-1}, s_{t-1})\, b_{dialog}(s_{t-1})\Big) \qquad [3.10]$$

One-way dialog patterns can be modeled by *idle* utterances m_{idle} and u_{idle} [129]. Moreover, two levels of simplification may be applied: using a direct mapping between utterance and dialog state $m_t^i \sim s_t^i$, the observation model can be discarded. One step further, if the predictive model is not suitable for improving belief estimation in a setting, it may be discarded and the Bayes filter dropped altogether:

$$m_t^i \sim s_t^i,\, b_{dialog}(s_t^i) = p(m_t^i) \qquad [3.11]$$

In any case, the output feature state belief has a similar representation as the input: a discrete probability distribution over a set of symbols. The feature representing types of observed symbolic body activities of interacting humans $f_{human-act}$ (discussed in Section 2.7.1) is handled in the same manner.

Absolute pose of the robot in the world, $f_{robot-pose}$ is a feature linked to robot self-localization. On the robots presented in Section 3.1.1, a self-localization skill component performs Kalman filtering on the robot pose (x, y, θ), thus delivering a continuous belief (x_t, y_t, θ_t, Cov(x_t, y_t, θ_t)) in high frequency. No further filter is necessary, but discretization has to be performed. Discretization can be performed with or without orientation θ. In both cases, normal distributions have to be integrated over continuous regions of space, which is not possible analytically for multi-dimensional co-variance. An option implemented in the presented system discards orientation and integrates over small, non-overlapping rectangles over $r_i = (x_1, y_1, x_2, y_2)$, using a state-of-the-art numerical solution [50]. Categories of which the feature state space can be composed $c_j \in F_{robot-pose}$, are formed by several regions:

$$p(r_i) = \int_{x_{rp1}}^{x_{rp2}} \int_{y_{rp1}}^{y_{rp2}} N(x, y; \vec{\mu_{pos}}, \Sigma_{pos})\, dx\, dy \qquad [3.12]$$

$$p(c_j) = \sum_{r_i \in c_j} p(r_i) \qquad [3.13]$$

$$b_{\text{robot-pose}} = \{p(c_1), \ldots, p(c_{|robot-pose|})\} \qquad [3.14]$$

The output given is a discrete probability distribution over a set of symbolic locations, represented by categories c_j. Consequently, output describes symbolic topological robot locations, which is a description distinctively different from the input. Absolute pose of interacting humans, $f_{human-pose}$ is processed in the same manner.

State of furniture objects $f_{furni-state}$ processes input values from multiple, diverse perception skills, yet in most missions does not perform Bayes filtering because prediction model complexity makes its inclusion un-worthwhile in the light of powerful perception. Given input is a 6D pose of a furniture object (x, y, z, r, p, yaw) with uni-dimensional covariance, an object type confidence $p(\text{type})$ and optionally further object part size values (see Section 3.3.4 for a detailed description of the localization system). Additionally, hand finger angles and 6D heel of hand (TCP) forces and moments (x, y, z, r, p, yaw) are received.

The furniture 6D pose is reduced to a (x, y, θ) pose which is processed in the same manner as in the robot-pose feature. Additionally, hand-finger angles and TCP are processed together

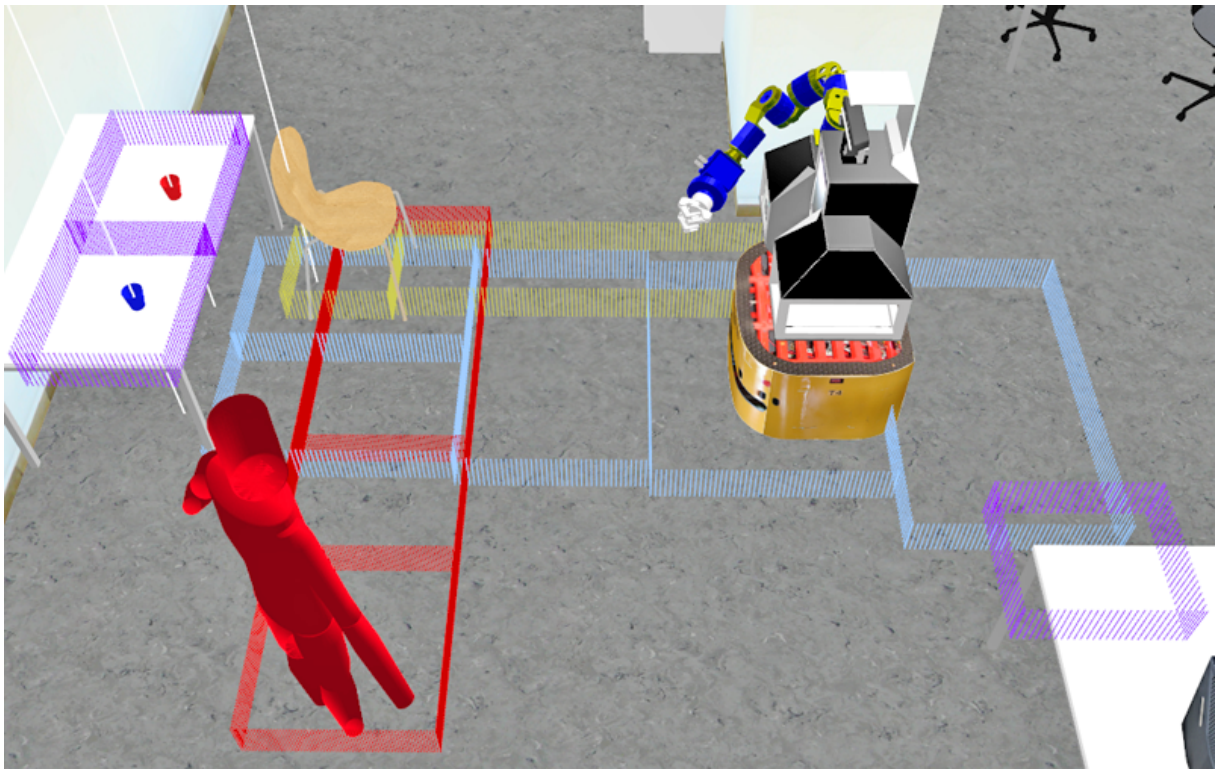


Figure 3.7.: *Illustrative example:* features with geometric input domains: $f_{robot-pose}$ in light blue, $f_{human-pose}$ in red, $f_{furni-state}$ in yellow and $f_{small-obj-state}$ in purple. - [125]

with manipulation action triggers to check for situations during object manipulation, e.g. *contact* or *in-hand*. By these means, rough object configurations relevant for manipulation action (strategy) selection are represented in the feature state space as illustrated in Figure 3.6. Feature states not related to poses alone are discussed in detail in Section 4.12, together with methods for deriving predictive models suitable for application in the filter. An exemplary category set is: *poseclass-A, poseclass-B, outside, jammed, not-present*. Using another localization method, small, portable objects are handled in a similar manner by the feature $f_{small-obj-state}$.

In evaluation settings, the following features have been applied as presented in Section 5.1.2:

1. $f_{robot-pose}$ (x, y indicated light blue in illustrative examples).
2. $f_{human-pose}$ (x, y indicated red in illustrative examples).
3. $f_{dialog-state}$ (abstract only).
4. $f_{human-act}$ (abstract only).
5. $f_{furni-state}$ (x, y indicated yellow in illustrative examples).

6. $f_{small-obj-state}$ (x, y indicated purple in illustrative examples).

Among these, the first relates primarily to mobility, the next three relate to HRI, and the latter two relate to object manipulation. Geometrical features are illustrated in Figure 3.7

In summary, feature filters perform directed reduction of a situation description while still preserving low-level measurement uncertainty for high-level probabilistic decision making. Specific observation uncertainty modeling and corresponding high-level POMDP observation models with the characteristics of skill component uncertainty are discussed next.

3.3. Discrete State-based Modeling of Observation Uncertainty

Observation uncertainty characteristics in feature-filter models have to be reflected, at least approximately, in POMDP observation model probabilities incorporated for policy computation. State-of-the-art POMDP robot applications (see Section 2.2.6) typically contain observation models that are just rough approximations of real-world correlations. To benefit from the strengths of abstract planning while utilizing observation models that are quite precise approximations of real-world dynamics is still mostly an unsolved problem, considering complex skill domains like natural HRI and autonomous manipulation. Hence, this Section contains a discussion of how to derive parameters (actual values) for abstract level POMDP observation model probability distributions from analysis of probabilities delivered by skill components.

First, perception skill components are discussed for which only a rough model of their internal operations is known. Empirically validating these internally unknown components is outside the scope of this thesis. However, to overcome this limitation, an exemplary, complex perception skill component was developed specifically for the presented decision-making system. This skill component models measurement uncertainties thoroughly from sensor data processing upwards, regarding further uncertainty introduced and altered by processing algorithms as explained in Section 3.3.4.

3.3.1. Observation Model Representing Human Utterance Uncertainty

Speech recognition - and the respective feature $f_{dialog-state}$ discussed in Section 3.2.2 - is based on a fixed grammar handled by the perception skill component. Consequently, any sound recorded by the robot is matched to the most similar known human utterance. Even with suitable microphones and state-of-the-art speech recognition, distant recognition without use of a headset is highly error prone. Two types of systematic uncertainty are prevalent:

1. Background noise matched as certain utterances frequently.

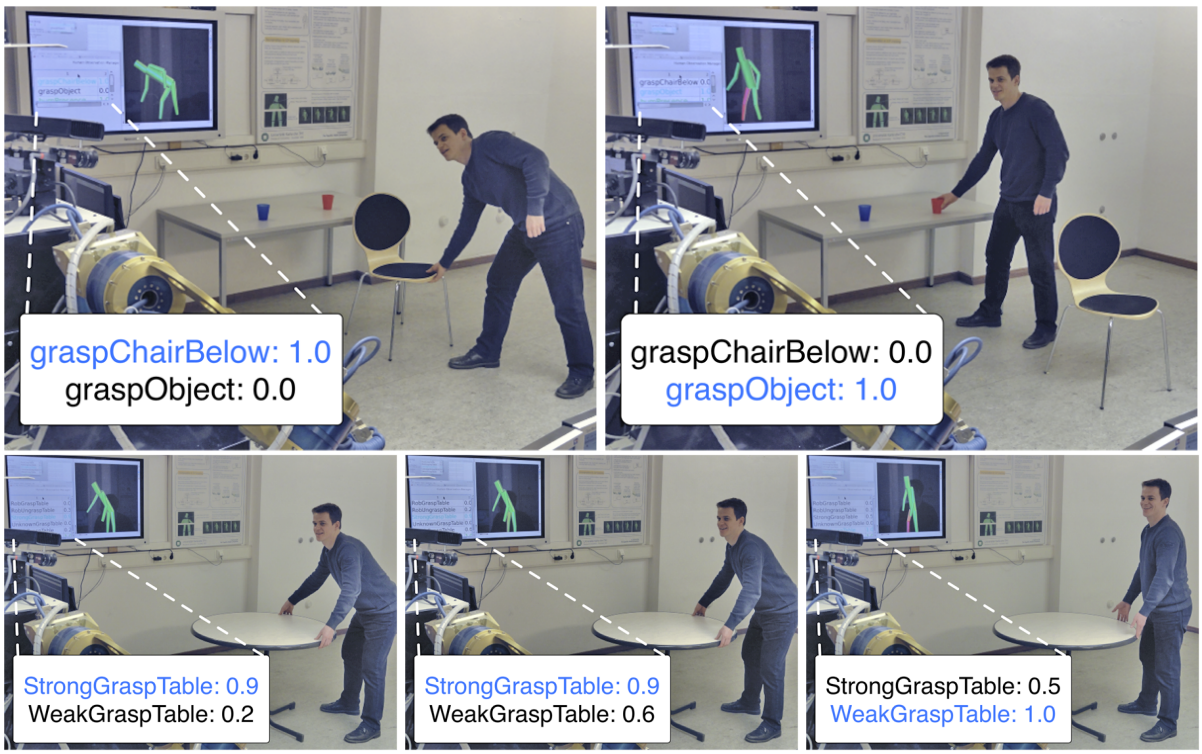


Figure 3.8.: Similar activities "GraspChairBelowFront" and "Grasp"[Small]"Object" are easily distinguished because of significant torso-to-floor angle difference (top). Below, "StrongGraspTable" and "WeakGraspTable" are more similar. - [125]

2. Utterance confusion: frequent mismatches between similar utterances.

While the first aspect can only be determined empirically which requires discussion beyond the scope of this thesis, the second can partly be tackled analytically. Confusion of utterances $m_{human-utterance}$ in technical speech recognition systems is subject to the same characteristics as human speech recognition in noisy environments: acoustic similarities, based on syllable and word sound similarity and overall length similarity define confusion probabilities. Accordingly, an utterance-similarity metric can be created, based on acoustic single-word similarities as shown in [113]. With such an utterance similarity metric $\langle m_i, m_j \rangle_{human-utterance}$, confusion probabilities in the observation model O can be inferred [130]:

$$O_{dialog-state}(i,j) = \frac{\langle m_i, m_j \rangle_{human-utterance}}{\sum_j \langle m_i, m_j \rangle_{human-utterance}}$$

By these means, the POMDP observation probability distribution can approximate speech-recognition feature filter output distributions.

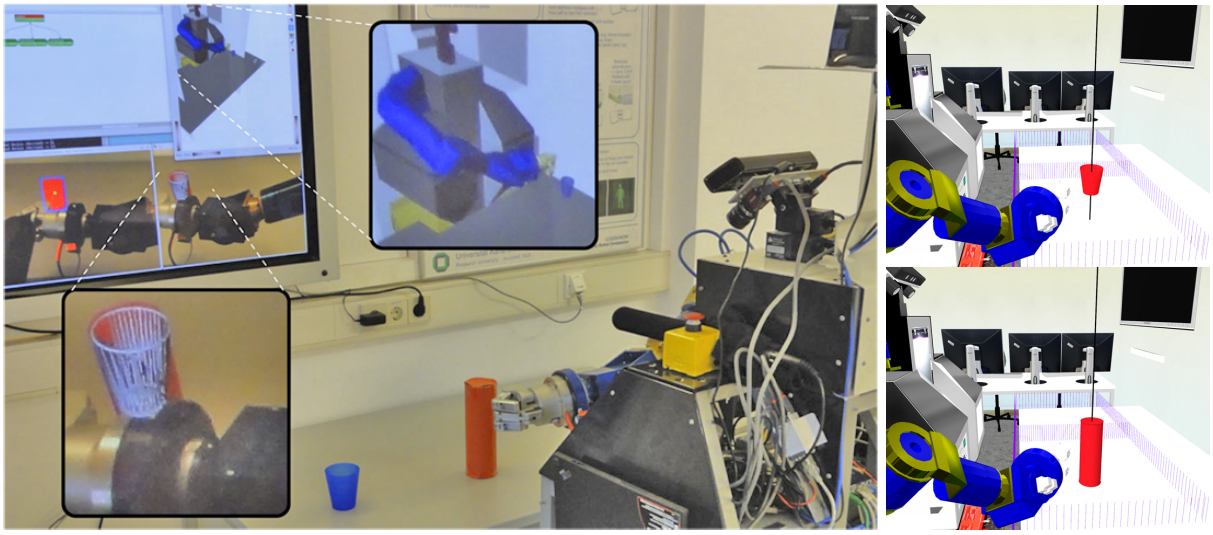


Figure 3.9.: Object confusion error. Occlusions lead to one object being detected as another (left) with the observed state (top right) being distinct from the intrinsic one (bottom right). Note: the red cup is also shown blue in the motion planning visualization. - [125]

3.3.2. Handling Human Body Activity Uncertainty

Recognition of symbolic human body activity types is similar to human speech recognition. A fixed set of activities can be recognized and probabilities are computed, reflecting likelihoods of intrinsic activities. However, there is one profound difference concerning the skill component discussed in Section 2.7.1: probabilities do not add up to one: instead several activities may be recognized in parallel. Therefore, background noise (e.g. from arbitrary movements) is not always matched to activities and probabilities have to be normalized for a feature belief. Therefore, a dummy *idle* activity has to be introduced. When normalizing independent activity probabilities, confusion errors can occur as with speech recognition. Hence, application of a similarity metric among activities is suitable. In this case, body configuration similarities take the place of acoustic similarities. The metric is defined over the attribute space of an activity $Att(a)$, which encompasses limb poses, joint angles and joint angle velocities as shown in Figure 2.25. Such a metric has to reflect similar motions that are easily confused by activity recognition [91]. Figure 3.8 illustrates similarity differences on activities a_i, a_j used in experiments, discussed in Section 5.1.2. Actual values for a metric can be computed when performing recognition training. During classification training, the set Att_a of important attributes of an activity and its SVM margin $\langle a_i, a_j \rangle_{svm}$ between activity SVM classifications a can be computed. This leads to an α-scaled metric:

$$\langle m_i, m_j \rangle_{body-activity} = |Att_{a_i} \bigcap Att_{a_j}| * \alpha \langle a_i, a_j \rangle_{svm}$$

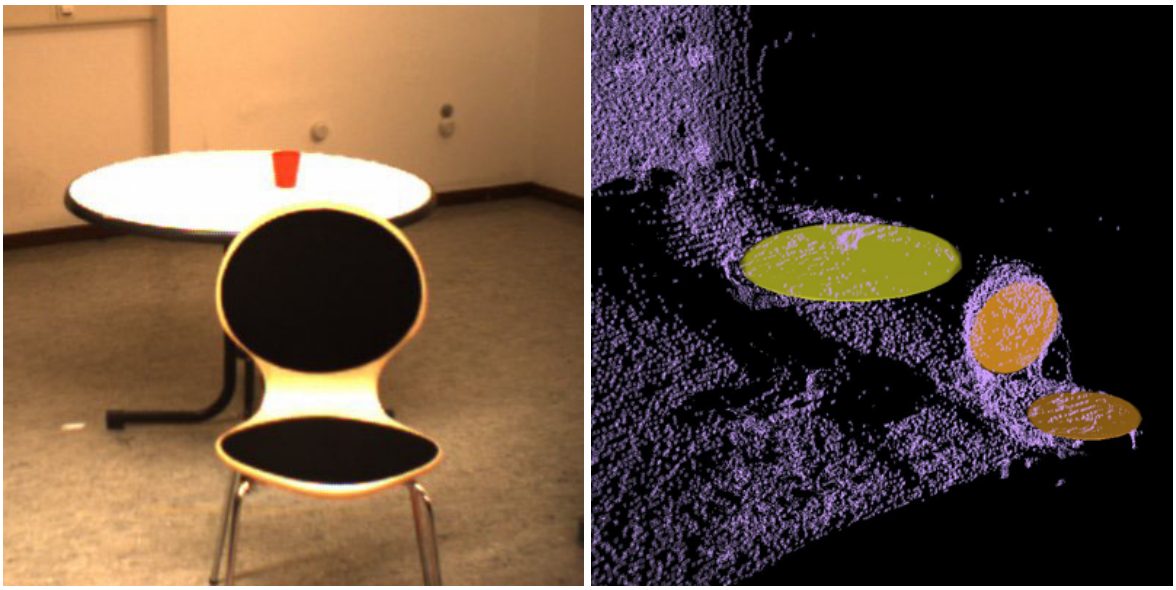


Figure 3.10.: Illustration of furniture localization with left stereo camera view as seen by the robot (left) and resulting detected object poses visualized in the 3D depth camera point cloud, in this case a Swissranger-4000 (right). - [97]

This metric can be used to generate observation model probabilities [130].

3.3.3. Small Object Localization Uncertainty

Relevant small object localization [8] uncertainty classes are *Existence* (false positives and false negatives) and *Object type* (confusion). A commonly occurring existence false negative example is shown in Figure C.1 while typical object type confusion is illustrated in Figure 3.9. In contrast to furniture localization, which was a dedicated development to support the presented system, small-object localization confusion and existence uncertainty was only modeled empirically. Respective observation model entries are consequently introduced into the background knowledge to be applicable to mission model generation as mentioned in Section 4.10.

3.3.4. Furniture Localization: an Example of Grounded Uncertainty

A complex observation skill, localization of furniture objects with elliptic parts, depicted in Figure 3.10, was developed as part of the presented system to explore consistent and detailed transfer of observation uncertainties from sensor to abstract level. As 2D and 3D image processing, heavily used in this skill component, is not in the focus of this thesis, processing elements will only be shortly referenced in the following with focus on how they are correlated with or transform uncertainty. Details of the image processing operations are discussed in [98], [97]. In

the given skill component, furniture localization consists of the following main process steps, illustrated in Figures 3.11 and 3.12:

1. A scene is captured by the robot head in its direction of view by the 3D point cloud and one of the 2D color cameras mentioned in Section 3.1.1.
2. After filter-based processing of 2D and 3D disparity images and fusion of both images, edges which are both depth and intensity edges can be extracted.
3. Based on these edges, a generalized Hough transformation extracts incomplete and imperfect 2D ellipses robustly.
4. By projecting ellipses back onto the 3D point cloud, points of interests in the point cloud, potentially belonging to objects, are selected (stamped out).
5. On a point cloud reduced in this way, planes are computed by the RANSAC algorithm.
6. 2D Ellipses are projected onto these computed planes, leading to 3D ellipses that represent potential object parts.
7. Known object types like chairs and tables are classified based on parameters of their parts like poses relative to each other and the world as well as extent of their parts.
8. Output of the skill consists of class (type), pose and extent of each object as well as confidence values for each of these values.

Uncertainty first arises from physical properties of both sensor types. In the following process stages, it is altered and amplified by a wide range of algorithmic operations and assumptions. By computing confidence values for certain properties along the way and including it in the output, the uncertainty can be preserved to be used in the filter system for belief computation. Furthermore, by empirical evaluation of these uncertainties, prior likelihood observation models can be composed. On the output level, several different properties contain uncertainty in the form of confidence probabilities:

1. Existence of an object of any known type in the scene at a certain location
2. Object type among the set of known objects for a given object
3. Pose of a given object
4. Proportions of a given object

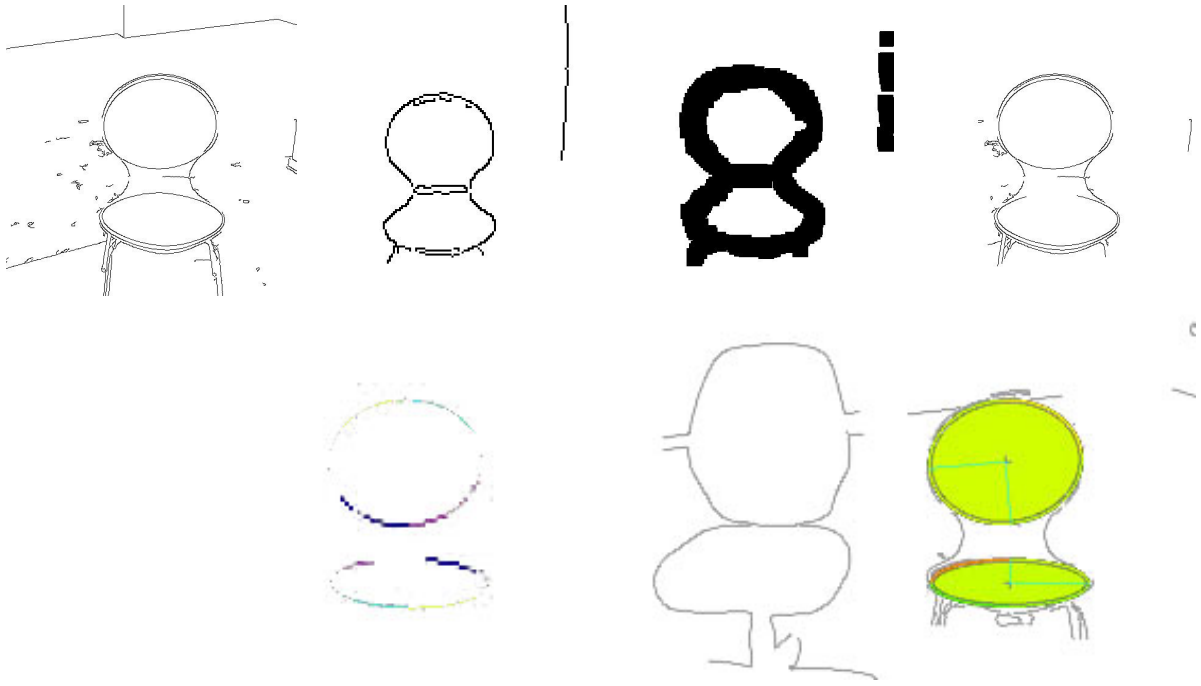


Figure 3.11.: Processing stages in furniture localization. Top from left to right: 1) edges in the intensity image, 2) edges in the depth image, 3) dilated depth edges, 4) combined (AND) edges. Bottom: 1) ellipse pixel voting, 2) ellipse (quadrant) confidences. - [97]

Existence describes a binary value with false positives meaning an object was detected based on algorithmic flaws and scene characteristics where there is no object at all. False negatives occur when the algorithms are not able to detect a present object of known type. Both error types are common with object localization.

Confusing the *object type* means wrong classification of a detected object, occurring most frequently with objects similar in shape and size.

Deviations between true *pose* of a given object and the pose delivered by the perception skill always occur in practice. The amount of deviation depends on sensor and algorithmic characteristics.

Similarly as for pose, deviations in *object proportions* occur if the extent of objects is not given by a fixed and previously known geometric model.

Output confidence values contain error likelihood estimates of these parameters.

Hough ellipse fitting confidence In the first major algorithmic step, generalized Hough transformation [25] fitting ellipses in fused color and depth edge pixels Px_{ef}, aspects of all major object properties are determined. Object existence depends on ellipses being detected at all. Pose is related to ellipse center points. Proportions depend on the aspect ratio of the ellipse,

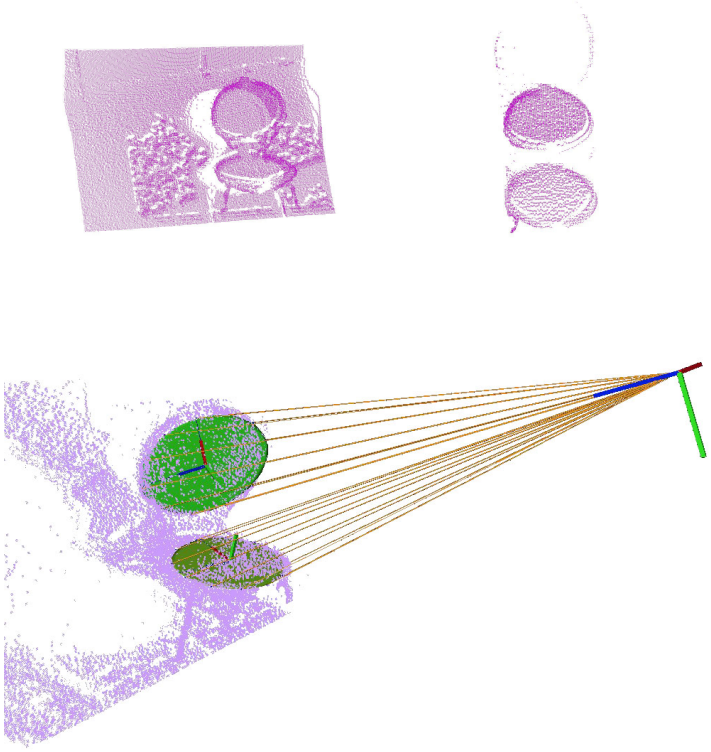


Figure 3.12.: Furniture localization depth point processing stages. Top: 1) raw points, 2) ellipse-correlating selected 3D points. Bottom: back-projection and fitting after RANSAC on selected points with camera coordinate system shown. - [97]

and types on all parameters. Therefore, uncertainties arise from deviations of fitted ellipses from an ideal fit in the image that reflects real object properties best.

A confidence value can be computed both for the overall pose and extent. A 2D ellipse is described by the following parameters center (o_x, o_y), major and minor axes α, β and spatial angle θ. In the general Hough transformation, it is determined how many edge pixels Px_{ef} correspond with a certain parameter set: $H^{Px_{ef}}_{total}(o_x, o_y, \alpha, \beta, \theta)$. This value is normalized by ellipse circumference. It is also applied to the four ellipse quadrants $H^{Px_{ef}}_{quadrant_i}(o_x, o_y, \alpha, \beta, \theta)$. Such a total confidence reflects that with more pixels voting for a certain parameter set, both center pose and extent are more likely fitting. To obtain confidences, normalized voting has to be adjusted to image pixel density by a function
$\eta(H) : p_{Hough}(o_x, o_y, \alpha, \beta, \theta) = \eta(H^{Px_{ef}}_{total}(o_x, o_y, \alpha, \beta, \theta))$. This probability represents an estimate that the given ellipse parameters correctly match a corresponding real object as illustrated in Figure 3.11 at the bottom-right.

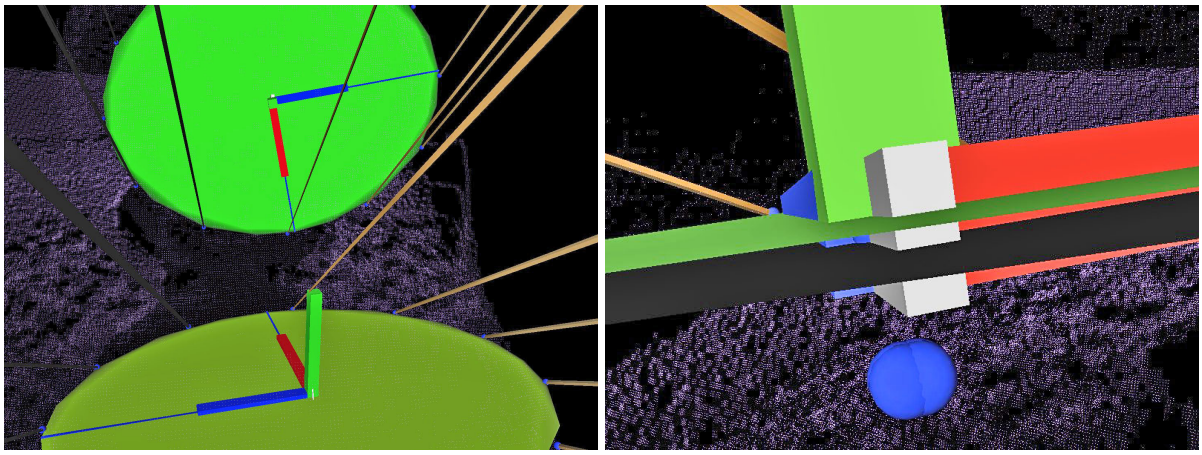


Figure 3.13.: $q_{loc}^{e^{pt}}$ of the 3D ellipse shown as gray box (right), for the chair seat (left). - [97]

RANSAC plane fitting confidence Plane-fitting variance on regions of interest in the 3D point cloud $\{pt_i\}$ depends on the squared average distance of points in the vicinity of a plane as computed by the RANSAC algorithm [46]. RANSAC computes model poses m_k for a maximum of n plane candidates from a subset of points $\{pt_l\} \subseteq \{pt_i\}$ within a certain maximum distance from the model plane $\|m_k - pt_l\| < \varepsilon$ and a minimum number of fitting points $|\{pt_l\}| \geq c_{min}$: $RANSAC(\{pt_i\}, \varepsilon, c_{min}, n) \rightarrow \{m_k\}$. Plane variance is increased by outliers which lead to worse fits. Based on parameters ε, c_{min} which define not considered outliers, a sample variance q_{m_k} of a plane model m_k with respect to its defining points $\{pt_l\}$ can be computed. Based on such variance q_{m_k} and empirically determined best and worst variance thresholds ω_b, ω_w, a plane confidence value $p_{Ransac}(m_k) \rightarrow [0,1]$ can be defined:

$$q_{m_k} = \frac{1}{|\{pt_l\}| - 1} \sum_{pt_l} \|m_k - pt_l\|^2 \qquad [3.15]$$

$$p_{Ransac}(m_k) = \begin{cases} 1 & if\, q_{m_k} < \omega_b \\ \frac{q_{m_k} - \omega_b}{\omega_b - \omega_w} + 1 & if\, \omega_b \leq q_{m_k} \leq \omega_w \\ 0 & if\, q_{m_k} > \omega_w \end{cases} \qquad [3.16]$$

RANSAC ε and ω_w have to guarantee no points $> \omega_w$ being in $\{pt_l\}$. This approach considers noise and data sparsity, but cannot reflect systematic distortions shown by some depth sensors.

Confidence of ellipse onto plane projection The previous two computed confidence aspects are only marginally conditionally dependent, given large ellipses performing preselection of 3D points into which planes are fitted later on. However, when projecting ellipses onto determined planes, both with their own confidence values, those confidences have to be combined because there are significant interdependencies. Projecting a 2D ellipse via a non-orthogonal

angle onto a 3D plane generally does not lead to an ellipse. Therefore, this process projects sampled points on the 2D ellipse individually onto the 3D plane. It then selects 3D points $\{pt_l^e\}$ belonging to the plane and being closest to projected points. Finally, a 3D ellipse e^{pt} is fitted into $\{pt_l^e\}$, which can be considered derived from $(m_k), (o_x, o_y, \alpha, \beta, \theta)$.

Hence, the variance of the 3D ellipse e^{pt} can be defined in terms of the variance of $\{pt_l^e\}$ relative to e^{pt}. To account for variances in ellipse detection and plane fitting, the variance along the plane normal e_z^{pt} is computed from q_{m_k}. In contrast, variance within the plane e_x^{pt}, e_y^{pt}, similar to plane confidence, is a sample variance of $\{pt_l^e\}$ relative to e_x^{pt}, e_y^{pt}. To compute sample variance, pairs of sensor 3D points closest to the ellipse $pt_{l,i}^e$ and corresponding closest (not measured, but computed) points $pt_{ell,i}^e$ *on* the ellipse are determined [94]:

$$pt_{ell,i}^e = argmin(\|pt_{ell,i}^e - pt_{l,i}^e\|), pt_{ell,i}^e \in e^{pt} \quad [3.17]$$

Applying pairs $(pt_{l,i}^e, pt_{ell,i}^e)$, sample variance $q_{e_x^{pt}}$ and in the same way $q_{e_y^{pt}}$ can be computed, isolated for each dimension. Then, individual dimension variances for a 3D ellipse are composed into $q_{loc}^{e^{pt}}(e)$:

$$q_{e_x^{pt}} = \frac{1}{|\{pt_l^e\}| - 1} \sum_{i=1}^{|\{pt_l^e\}|} \|x(pt_{ell,i}^e) - x(pt_{l,i}^e)\|^2 \Rightarrow q_{loc}^{e^{pt}}(e) = \begin{pmatrix} q_{e_x^{pt}} \\ q_{e_y^{pt}} \\ q_{m_k} \end{pmatrix} \quad [3.18]$$

Accordingly, $q_{loc}^{e^{pt}}$ is a vector of variances - without covariance - in the base of the plane. The box shown in Figure 3.13 is an example. It can be transformed into the world-relative coordinates by applying a boundary box, which represents a variance upper bound estimator $B(q) : q_{loc}^{world}(e) = B(q_{loc}^{e^{pt}}(e))$. In turn, such a variance q_{loc}^{world} can then be used as an estimate for Σ in a normal distribution, denoting the pose of the ellipse, and μ in the normal distribution derived as follows: $\mu_x = e_x^{pt}, \mu_y = e_y^{pt}, \mu_z = e_z^{pt}$. Finally, the probability of the object part pose being inside a certain pixel or voxel, reflecting certain feature states $c_i^{furni-state}$ can be computed applying that normal distribution as described in Section 3.2.2.

Type classification confidence In this framework, the type of an object Obj is described as being composed of a set of parts $\{Part_i\}$, each described by a set of parameters w_j lying inside intervals $I_{Obj}(w_j)$:

$$I_{Obj}(w_j) := \begin{cases} 1 & \text{if } w_{min}^i \leq w_j \leq w_{max}^i \\ 0 & \text{else} \end{cases} \quad [3.19]$$

$$Part_i := \{I_{Obj}(w_1), \ldots, I_{Obj}(w_m)\} \quad [3.20]$$

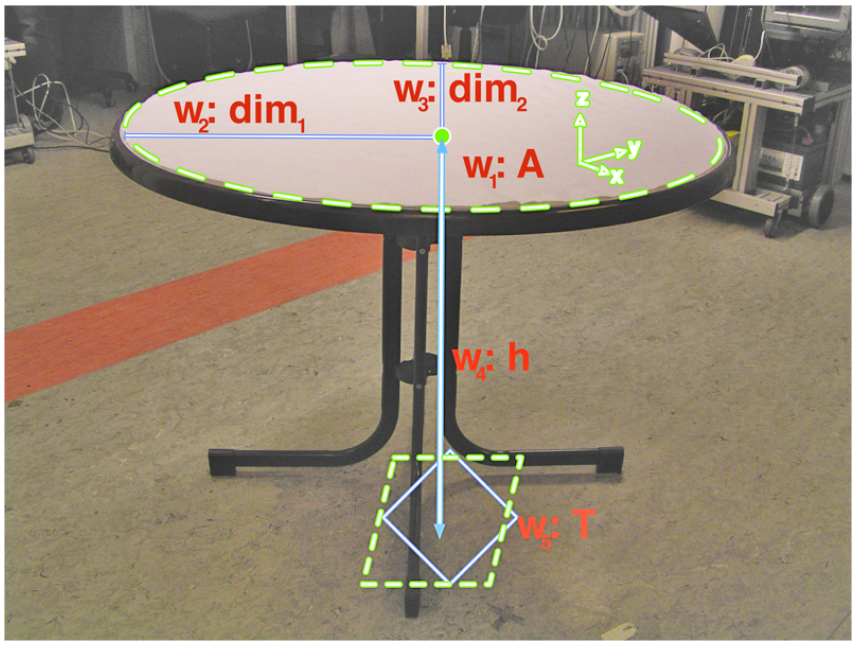


Figure 3.14.: Illustration of table classification parameters. - [97]

As an example an *elliptical table for manipulation* can be described by a single part, a 3D plane area that can be detected as explained above. To be classified by type, some properties w_j of a planar ellipse, as shown in Figure 3.14, have to be inside defined intervals $I_{Obj}(w_j)$:

1. Surface area w_1.

2. Allowed ranges of ellipse semi-axes extends $w_2 \sim \alpha_{3D}$, $w_3 \sim \beta_{3D}$.

3. Height above ground w_4.

4. Angle of the plane normal against the ground normal w_5.

Intervals can be learned from examples or defined manually. Variances q of parameters w_1 to w_4 have been computed during detection, reflecting the probability of the true value $p(x_j)$ of a parameter. Thus, the total probability inside the parameter interval can be computed by integration:

$$p(x_j) \Rightarrow p(I_{Obj}(x_j) = 1) \qquad [3.21]$$

Furthermore, an existence confidence for ellipses $p_{exists}(Part_i) = p_{Hough}(Part_i)$ has to be computed. Without considering conditional interdependences, object-type confidence can be computed from confidences about parts:

$$p_{belongstotype}(Obj, Part_i) = \prod_{j}^{m} p(I_{Obj}(x_j^{Part_i}) = 1) \quad [3.22]$$

$$p_{parttype}(Obj, Part_i) = p_{exists}(Part_i) * p_{belongstotype}(Obj, Part_i) \quad [3.23]$$

$$p_{isobject}(Obj) = \forall Part_i \in Obj : \frac{\sum_{Part_i} p_{parttype}(Obj, Part_i)}{n} \quad [3.24]$$

Objects with multiple parts, such as a chair with seat and back, may be sufficiently detected with only some parts being present. Thus the average probability is a better, because more aggressive, estimate than the product.

Conclusions While such confidence and variance values are still imperfect and cannot be interpreted as an optimal approximation of uncertainty in a frequentist sense, such grounded Bayesian uncertainty likelihoods are a far more precise measure than crude, manually given approximations as typically applied in the probabilistic decision-making literature. It also shows that modeling and computing meaningful confidence likelihoods is both non-trivial and can never be accurate in a frequentist sense for complex real-world robot perception skills. It still helps a robot to make more robust decisions when facing real-world uncertainty. More details are discussed in [97] and a thorough evaluation can be found in [98]. In experiments described in Section 5, extents of furniture objects were discarded and instead certain types mapped onto fixed, known mesh models for motion planning and visualization. Thus, only the parameters *existence*, *object type* and *pose*, but not *proportions* were actually used in the scope of the presented system.

3.4. Modeling Tasks Reflecting Elementary Actions in Decision Making

Given state and measurement set definitions S, M, the POMDP model domain definition is completed by the action set A. Similarly as for S, role, scope and grounding of abstract actions A have to be determined. With diverse skill domains such as mobility, natural HRI and object manipulation, such grounding is challenging. It can be specified from two main perspectives:

1. Modeling abstract actions as subtasks incorporating several different actuator commands and control loops: the architectural and abstraction perspective as discussed in Section 2.1.3.

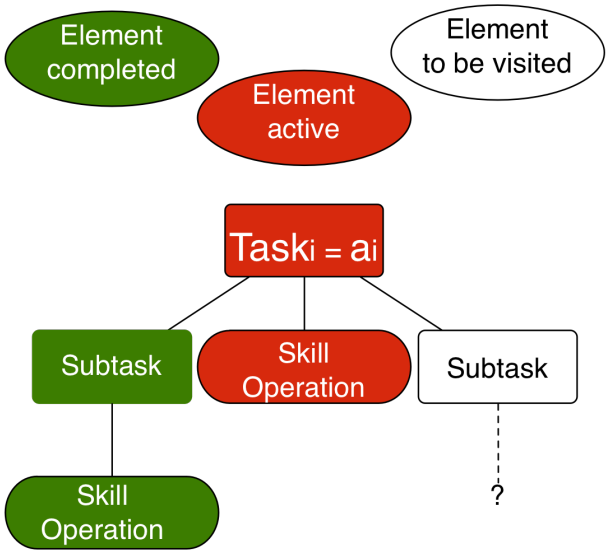


Figure 3.15.: Schematic view of flexible programs (FPs), a HTN type. - [77], [125]

2. Modeling abstract actions as sets of actuation elements, each contributing stochastic effect correlations: the POMDP transition model perspective as discussed in Section 2.2.6.

3.4.1. Designing Abstract Actions as Complex Subtasks

Abstract action symbols have to be transferred into a compound of, potentially continuous, actuator commands and control loops. Such a transfer is in fact a subtask decomposition, reflected by a *Hierarchical Task Network* (HTN) subtask representation in the presented system.

Actuation control skill component interfaces as described in Section 3.1.1 are addressed in leaves in this HTN implementation, called *Flexible Programs* (FPs) [77]. Actuation-skill parameterization is encapsulated in leaf commands as illustrated in Figure 3.15. Execution of leaves may take place in parallel. While large FPs can simulate complete missions as complex finite-state machines by incorporating conditional branching and conditional loops, as used in evalution discussed in Section 5.2, tasks representing decomposition of POMDP actions are compact and without loops or branches, except conditional termination. Skills like navigation commands, robot utterances and complex manipulation strategies are adressed through single leaves. Subtasks can differ in complexity: for instance, a robot utterance may encompass only one leaf, grasping a chair may encompass moving the arm into a suitable starting position, navigating a little closer to the chair and then executing a certain grasp strategy. The abstract action has finished, when the FP subtask finished, regardless of the termination cause.

For navigation and utterance actions, FP templates can provide a generic shell as illustrated in Figure 3.16. More complex subtasks, like object-manipulation compounds, including corre-

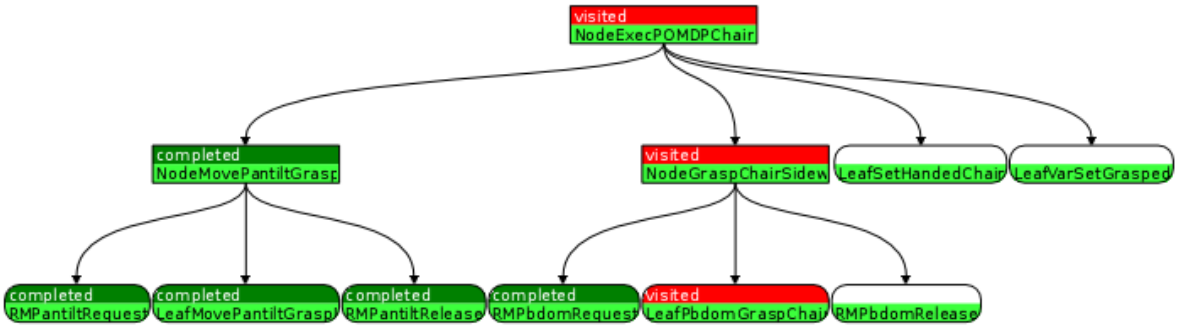


Figure 3.16.: Screenshots of a visualization of flexible programs (FPs), executed in POMDP elementary actions in mission CESM-1 (see Section 5.1.2). "GraspChair" at the top includes setting the neck to look into the direction of the grasp. More examples can be found in Figure C.2. - [125]

sponding manipulation strategies, can be stored and shared for all missions. Both FPs [76] and manipulation strategies (see Section 2.7.2) can be acquired through skill- and subtask-level PbD. This provides comfortable acquisition of skill and subtask representations, especially when also learning mission models using PbD as discussed in Chapter 4.

3.4.2. Transition Models of Abstract Actions

Taking a different perspective, abstract actions transfer an abstract world state into another. In POMDPs, with underlying stochastic action effects, an action can transfer a state into several others with certain probabilities, implied by the transition model $T(s',a,s) := p(s'|s,a)$. With an abstract action, many distinct, potentially independent, correlations may contribute to a single transition probability. Transition-probability sources can be roughly classified as two types:

1. Actuation-skill effects and interactions with the environment which cannot be modeled in a deterministic way. Actuation skill effects may arise from each leaf operation in a subtask in turn.

2. Dynamic events happening in the environment, quite unrelated to the action itself, which correlate just with the origin state and the temporal duration of the action.

Subtask operations are typically conditionally dependent on operations executed previously or in parallel. Dynamic events are mostly conditionally independent of subtask operations, as they are classified this way. Consequently, to model abstract transition probabilities for origin-state-action pairs (s,a) synthetically from its parts, conditional dependence of all aspects has to be determined, conditionally dependent probabilities must be inferred and finally merged with conditionally independent probabilities. This basically leads to each transition model row

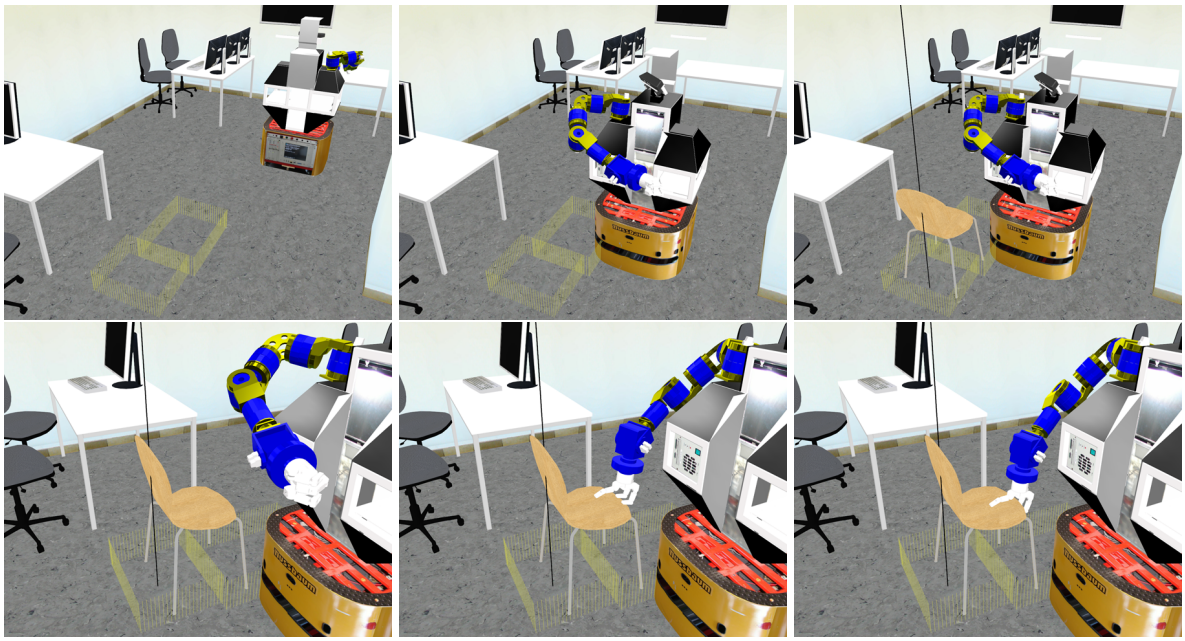


Figure 3.17.: *Illustrative example:* Primary stochastic action effects at the top: origin state (left), potential effects (center, right). Here, probabilities may depend on behavior of humans or other robots. Secondary stochastic action effects at the bottom: origin state (left) and potential effects at the (center, right). Here, probabilities depend on robot grasping peculiarities. - [125]

$T(s', a_k, s_i) : (s_i, a_k) \rightarrow (p(s'_1), \ldots, p(s'_n))$ being the result on an inference in a Bayesian Network.

Manual, correct modeling of $s \times a$ BNs is infeasible. Instead, overall subtask transition probabilities have to be determined by learning instead for conditionally dependent operations with only mostly conditionally independent probabilities merged. That approach is taken in POMDP mission PbD discussed in Chapter 4. Examples are shown in Tables C.9 and C.10.

Furthermore, stochastic transition effects can be distinguished along another characteristic which becomes obvious with PbD: *primary transition effects* are agent invariant and define a mission (such as dynamic events, human behavior or stochastic object properties) while *secondary transition effects* are dependent on skills of a specific robot. While the former type can be learned from human demonstrations, the latter type can only be acquired from explorative robot learning. An illustration with examples of both effect types is shown in Figure 3.17.

3.5. Robot Mission Modeling

Practical mission models providing service robot decision making in real-world settings have to contain precisely crafted sets and values as discussed previously. Generating mission models

thus encompasses determining state S, action A and measurement M sets as well as transition model T and observation model O probabilities plus reward model values R in a manner leading to good robot mission performance. *Good* is specified here as a POMDP model representation and decision-making system leading to superior performance compared to non-probabilistic (FSM, logic-based planning) or MDP based action selection as discussed in Section 5.2.

In the literature, typically one very specific mission is handcrafted as discussed in Section 2.2.6, which means a human engineer determining at least most, if not all, crucial parts of the model S, A, M, T, O, R. Such an approach can be interpreted as explicitly programming a service robot mission model in a POMDP representation. For a generic, abstract level decision-making system of highly versatile anthropomorphic service robots covering diverse skill domains, potentially performing a wide range of missions, such an approach is cumbersome as demonstrated next.

3.5.1. Systematic Design of Service Robot Missions

Given clear, written specifications or observations of humans conducting a mission, an engineer first has to segment a mission conceptually, spatially and temporally.

A) Segmentation determines skill domains involved in various portions of a mission.

1. Skill domains are analyzed for the minimal set of features $\{f_1, \ldots, f_n\}$ providing sufficient interaction with the world to perform the mission successfully. Mission model grounding concerning state and observation space is provided by this set of features.

2. For each feature f_i, filter discretization $f_{g1:i}$ has to be determined. Basically, major parts of the resulting state space S_p are designed in this step. Discretization is closely tied to fine grained temporal and spatial segmentation of typical courses of events in a mission. In features which do not have a spatial aspect, such as interacting human intention, relevant distinctions are only conceptual and temporal.

3. Segmentation of contiguous courses of events can be performed considering three aspects:
   - Available self-contained subtasks in the action library
   - Events which may transfer one discretized feature state c_{x_i} into another c_{x_j}
   - Points of event branching in a mission, controlled by explicit decision making of the acting agent

   Resulting action choices A_p have to be optimized, until a minimum set is found that can compose all relevant courses of events in the mission.

4. As a result, the preliminary primary state and action spaces S_p, A_p can be compiled.

5. Finally, as observations are closely tied to states in the presented modeling concept, a preliminary observation set M_p can be added.

B) Dynamics model portions are highly specific to the POMDP formalism. Determining appropriate values for T, O, R is even more elusive than S, A, M when modeled manually.

1. Compilation and probabilistic representation of primary stochastic effects has to be performed first. Each state-action pair $(s_i, a_k) \in (S_p \times A_p)$ is analyzed for relevant effects, when performed by a generic humanoid agent (human or robot). Relative frequencies of different stochastic outcomes of a single pair $(s_i, a_k) \rightarrow (p(s'_1|s_i, a_k), \ldots, p(s'_n|s_i, a_k))$ are either derived from explicit mission specifications or human user trials. With $|s \times s \times a|$ probabilities in a transition model, manual design would be infeasible even for the most simple missions if a) in typical missions the vast majority of probabilities would not be 0 and b) items $p(s'_j|s_i, a_k)$ could not be grouped into sets of i, j, k. Sparsity arises simply from the fact that execution of an action a_k in a certain state s_i can and will causally not lead to most other states $s'_r := p(s'_r|s_i, a_k) > 0$: $|r| << n$.

   Grouping means assigning several transition probabilities while considering a single effect. Typically, grouping along features, thus subsets of $S = F_{feat^1} \times \ldots \times F_{feat^m}$, is possible as most actions are invariant outside their primary feature-state space. For example, expressing an utterance will never change the location of the agent himself. A toolset for grouping along this and other characteristics is discussed in Section 3.6. A classification scheme for primary stochastic action effects in typical household tasks is given in [87].

2. Determining secondary stochastic effects requires an extensive model and understanding of robot skill behavior characteristics. These may either be available from in-depth analytical investigations for simple skillls or be acquired from empirical studies. Secondary effect probabilities are only valid for a specific type of robot and skill component. Those effects may, for instance, arise from imperfect planning, actuation variance, hardware limitations (such as bad hand compliance or little friction) but also perception feedback which is directly processed in low-level control such as navigation. Grouping is also applicable in most cases, here. Examples of such secondary effects are shown in Figure 3.17. As a result, error states S_e have to be introduced (see Section 4.12 for examples).

3. Adding error states resulting from secondary effects leads to the state space $S := S_p \cup S_E$.

4. Determining additional error-state recovery actions A_E and information gain actions A_I, extends A_p to the full set $A := A_p \cup A_E \cup A_I$.

5. To finalize the transition model, including S_E, A_E and A_I, transition probabilities have to be normalized as well as primary and secondary effect probabilities merged. Furthermore, transition models for A_E and A_I have to be determined.

6. Next, the observation model can be generated using techniques outlined in Section 3.3.

7. Reward model costs (negative rewards) of each action are added to R, based on duration, effort and annoyance potential to interacting humans.

8. Finally, reward-model goals (positive rewards) are added to R, reflecting desired intermediate and end states of potential courses of events.

To verify a model, policy computation can be followed by policy visualization and analysis as discussed in the next Section. Furthermore, policy simulation can give additional insight, leading to model parameter refinement before testing decision making on the real robot. In summary, the manual process is cumbersome, error-prone, imprecisely specified - and completely infeasible for more complex missions. Thus, development of an automated process is mandatory, with stages inspired by the manual process. It can be accomplished by means of automated learning from recorded human demonstrations as discussed in detail in Chapter 4.

3.5.2. Mission Design Analysis by Policy Visualization

Policy computation from an explicit model by means of approximate value iteration, discussed in Section 2.2.5, provides a value function Γ, which is a set of linear functions α defined over the whole belief state space simplex. The linear function α_i with the highest expected future utility value at a given belief (point) b denotes the ideal action choice at that belief: $\alpha_{max}(b) = argmax_i(\alpha_i * b)$.

Visual analysis of a value function can give insight into action selection characteristics as boundaries in the belief space indicating different optimal actions at two distinct beliefs can be shown. However, it is difficult to achieve because of high dimensional belief states. Value functions can be plotted directly over the $|S|-1$ dimensional belief simplex with their utility value utility$(b(s_1),\ldots,b(s_{|S|})) = \text{valf}(b(s_1),\ldots,b(s_{|S|}))$. Naturally, only $|S| = 2$ and $|S| = 3$ can be plotted easily with a straightforward scheme, as shown in Figures 2.12, 2.13, 3.19.

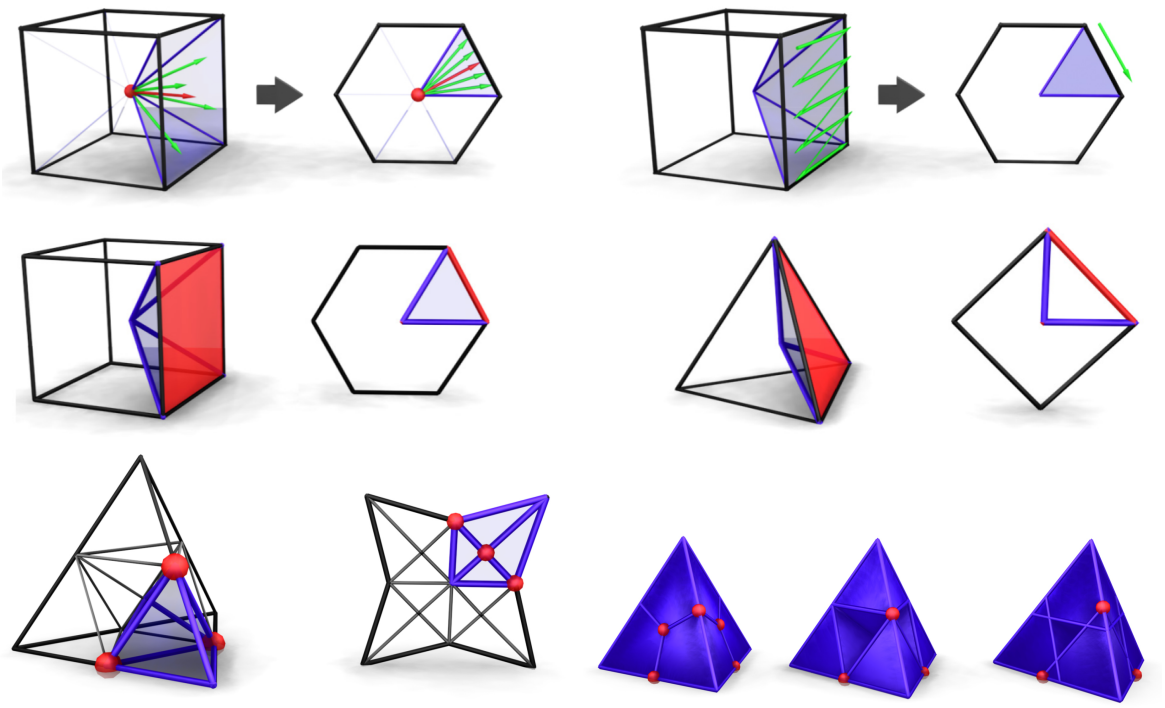


Figure 3.18.: Adaptations on the PolarEyez concept for StarViz. Plotting points in n-D (top), adaptation to the shape of the belief simplex (center). Mapping of corners of the belief simplex onto star rays (bottom left) and domain space lost to simplification (bottom right). - [105]

Colors denote actions represented by each linear function. The color with α_{max} with the highest utility at $\vec{b_t}$ represents the optimal action choice: $\forall \alpha_j : \alpha_{max} * \vec{b_t} \geq \alpha_j * \vec{b_t}$.

Such a policy visualisation can be used to determine near which beliefs a change in action selection occurs: $|\vec{b_1} - \vec{b_2}| < \varepsilon : a_{max}(b_1) \neq a_{max}(b_2)$. It also shows which actions are chosen in the corners of the simplex, representing beliefs with full state confidence, thus basically the fully observable MDP aspect of the POMDP. Furthermore, it can be noticed if certain actions are a_{max} only in very small regions of the belief space and thus rarely selected. Such information is important when assessing if a mission might be executed roughly as intended. While such a visualization can also be used for $|S| > 3$, only $|S| = 3$ subspaces of the simplex can be visualized with such an approach, losing all other gradient information in non-visualized dimensions. However, with simple beliefs with relevant belief probability values in just 1-3 states, $p_b(s_i) >> 0 : |i| \leq 3$ and relevant α gradients also restricted to these dimensions, subspace visualization can give critical insight into action selection behavior.

When dealing with more evenly distributed beliefs and more complex α gradients such straightforward visualization is insufficient and on the contrary may give a false sense of insight. Therefore, a visualization technique *StarViz*, able to handle more dimensions by exploiting regularities of linear functions and its convex maximum was developed [105]. It is based on the

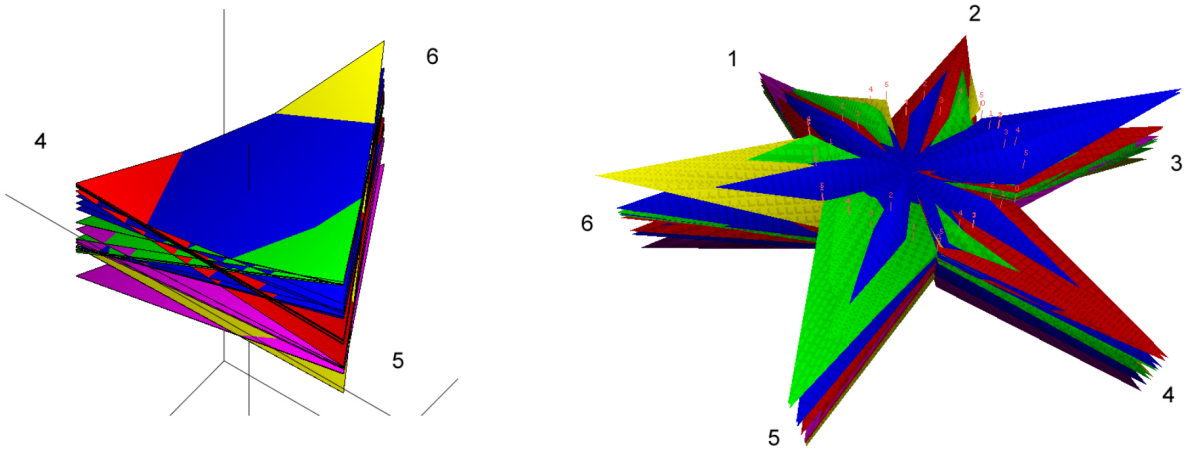


Figure 3.19.: Illustration of a 3D slice of a POMDP policy with 6 states in the original 3D policy visualization (left) and in StarViz (right). StarViz contains more relevant information. - [105]

PolarEyez concept [69], reducing n-dimensional function plotting into a plane by sampling the high-dimensional domain from a focal point outwards and reducing these values onto a sector of a 2D polygon with $2n$ corners.

As the belief space simplex has a distinct form because of $\sum_i p_b(s_i) = 1$, the PolarEyez projection scheme has to be altered as shown in Figure 3.18. However, complications arising from this are balanced by the simplification introduced by all function elements being linear. Finally, instead of colors representing function values (utility), utility values are plotted on the z-axis and instead corresponding actions are denoted by color as in the simple schemes as illustrated in Figure 3.19. Automated filtering of dimensions with least belief probability reduces visualization to a given number of dimensions (typically $n \leq 12$), and the shown dimensions are chosen automatically.

A chosen belief point forms the focal point of the diagram and gradients of the belief towards corners of the simplex for the most relevant dimensions (states) form rays in this star-like diagram. Hence, the most important aspect, changes in action choice around a belief, for instance becoming more certain about the world being in one or another state, can be seen in this diagram in a high-dimensional context. Nonetheless, as can be seen in 3.20, policies can be so complex that they are still difficult to read in this representation even by an expert. Yet, even such visualization is still a simplification of high-dimensional gradients of a large number of value function components (the set of α). Those typically range in the hundreds or thousands even with highly pruning approximate value iteration algorithms for realistic missions as discussed in Section 5.

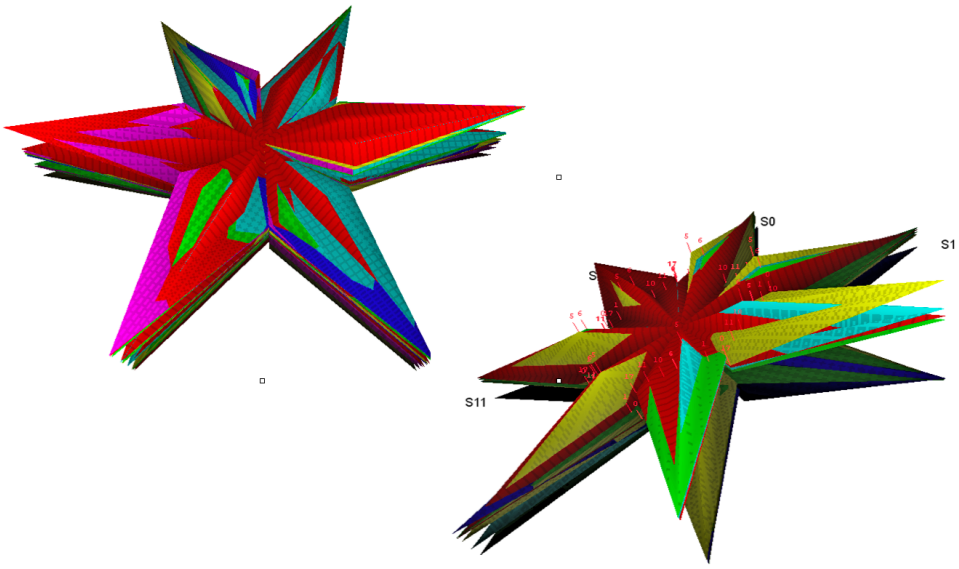


Figure 3.20.: 6D StarViz visualisation of policies of models with more than 20 states. -

Consequently, fully manual design of models and resulting policies is infeasible - the whole process has to be automated by a more natural (to the human) way of model and policy acquisition by the artificial system.

3.6. Functional Expressions Assisting Model Compilation

For handling abstract knowledge and learning from demonstrations, an intermediate functional expression processing system is presented. It mediates model value assignments by exploiting *grouping*, mentioned in Section 3.5.1. It reflects the characteristic of individual model aspects not only concerning a single 3d-tensor (T) or matrix (O,R) entry, but whole rows, columns, diagonals or even blocks. State spaces S being composed of feature state spaces F_i pronounces this tendency. A simple example in Table C.1 illustrates such a characteristic.

Consequently, there is a need for a toolbox of expressions handling such interdependencies, focussing model compilation on transition, observation and reward characteristics alone [37]. Processing flow in this toolbox is depicted schematically in Figure B.1. Basic expression processing is aligned along specific POMDP structure properties introduced by the way state and observation grounding (discussed in Sections 3.2, 3.3) are handled in the system:

- $feat_1, \ldots, feat_n$, a set of utilized skill domain *features*.
- $F_i := \{c_1, \ldots, c_{|feat_i|}\}, c_j = f_{feat_i}(E_s)$, a feature with instantiated discretization and category set (feature state space).
- $FS := F_1 \times \ldots \times F_n$, the set of feature state tuples $(c_{x_1} \in F_1, \ldots, c_{x_n} \in F_n)$.

- $S := \Psi(FS)$, a surjective mapping $FS \rightarrow S$, with some feature state tuples $fs_{b_1}, \ldots, fs_{b_y}$ combined onto a single state $s := \{\forall fs_{b_x} \in FS | \, \psi(fs_{b_x}, fs_{b_1})\} \Rightarrow fs_{b_x} \in s$.
- $A := a_1, \ldots, a_k$, a set of abstract actions.
- $MF_i \sim F_i$, a set of feature state measurements. Measurements are directly related to states by means of features: feature categories define a state and can be measured.
- $FM := (MF_1 \times \ldots \times MF_n)$, the set of feature state measurement tuples.
- $M := \Psi(FM)$ corresponding to $\Psi(FS)$.
- $O := \forall M, \forall S : \, p(m|s)$, observation-state correlation probabilities are conditionally independent of actions.
- $R := \forall A, \forall S : \, r(a,s)$, rewards are independent of measurements.
- $\pi_{F_i} : FS \rightarrow F_i$, $\pi_{F_i}(fs) := c_{x_i}$, a feature state selection in a feature state tuple.
- $\tilde{\pi}_{F_i} : S \rightarrow \mathscr{P}(F_i)$, $\tilde{\pi}_{F_i}(s) := \{\pi_{F_i}(fs) | fs \in s\}$, selection of all feature states within one feature assigned to a certain state.
- $\lambda_v(f_i)$, value entry corresponding to feature state $f_i \in F_i$ in the respective sub-model with $f_i = c_{x_i}$.
- Mat is a $|row| \times |col|$ matrix, with
  - $Mat(row, col)$, as an entry.
  - $Mat(row) := Mat(Mat(row, col_1), \ldots, Mat(row, col_m))$, a row-vector.
  - $\| \, Mat \, \|_{row}$, a function to normalize sums of rows in Mat with
    $$\| \, Mat \, \|_{row} \, (row, col) := \begin{cases} \frac{Mat(row,col)}{\|Mat\|_1} & for \, \| \, Mat \, \|_1 \neq 0, \\ \frac{1}{m} & else \end{cases}$$
    and $\forall row, \forall col : Mat(row, col) \geq 0$.

As shown in Figure B.1, *for each feature*, there are $|A|$ different $|FS| \times |FS|$ matrices, one $|FM| \times |FS|$ and one $|A| \times |FS|$ matrix Mat for internal processing and computation of expressions. All matrices are initialized with 0 before any expression is processed as described further below. After expression processing has generated all values in these internal matrices Mat, final POMDP model components T, O, R are computed in the following manner.

Transition model $T(s,a,s')$ is represented by a distinct $Mat\ T^a_{F_i}$ for each feature F_i and each action $a \in A$. Rows represent origin states $fs \in FS$ in the set of all feature state tuples while columns represent effect states of a transition $fs' \in FS$. Transition probabilities in the resulting POMDP considering the final state space S are computed by multiplying row-normalized transition frequency value entries in each feature transition model T_{F_i}. In the typical case, Ψ is the identity $FS = S$, and computing transition probabilities (Formula 3.30) is straightforward:

$$T^a(s,s') = \prod_{i=1}^{n} \parallel T^a_{F_i}(fs,fs') \parallel_{row} \quad [3.25]$$

$$P(s'|s,a) = \parallel T^a(s,s') \parallel_{row} \quad [3.26]$$

In contrast, when Ψ is not the identity, $|FS| > |S|$, computing transition probabilities is more complicated. The role of Ψ is to combine states, resulting from the product of feature state spaces $F_1 \times \ldots \times F_n$, which do not have to be distinct in a mission.

Reducing the number of effective states leads to faster computation of policies as discussed in Section 2.2.5 and also faster online belief computation and policy queries. In case of more complex missions, policy computation even with the most efficient algorithms may only be feasible after reducing the effective state space. While a resulting transition model requires probabilities defined on the reduced state space S, in the T_{F_i}, transition frequencies are defined only on the full state space FS. with respect to a transition model this means that transition values in rows, reflecting origin states $\pi_{F_i}(fs), fs \in s$ and columns reflecting effect states $\pi_{F_i}(fs')|fs' \in s'$ have to be combined. Combining effect states fs' (in columns) is trivial, as it is well defined: all resulting transition frequencies (probabilities) can be added up for all $fs' \in s'$:

$$P(\bigcup_{fs' \in s'} fs'|fs) = \sum_{fs' \in s'} P(fs'|fs) \quad [3.27]$$

Combining origin states is not straightforward, as their conditional dependencies $P(fs,a)$ are not known from the existing model. This implies that their weight ω_{fs} concerning a set of transition rows is unknown and thus probability distributions reflecting rows cannot be merged with a clear quantification of the share of each row. A solution is to derive the weight ω_{fs}

for merging rows from an additional parameter λ_{com} which is assigned to each $c_x \in F_i$ in the model. With this additional parameter, a final transition model can be computed:

$$com_{F_i}(fs) = \begin{cases} \lambda_{com}(\pi_{F_i}(fs)) & for \|\tilde{\pi}_{F_i}(s)\| > 1 \\ 1 & else \end{cases} \quad [3.28]$$

$$\omega_{fs} = \prod_{i=1}^{n} com_{F_i}(fs) \quad [3.29]$$

$$T^a(s,s') = \sum_{fs \in s} \left(\omega_{fs} \sum_{fs' \in s'} \left(\prod_{i=1}^{n} \| T^a_{F_i}(fs, fs') \|_{row} \right) \right) \quad [3.30]$$

$$P(s'|s,a) = \| T^a(s,s') \|_{row} \quad [3.31]$$

Determining well-defined λ_{com} manually is difficult except for very simple Ψ (e.g. combining only two fs) as it correlates to implicit model complexity reduction. Therefore, combining fs is mostly suitable to automated model generation with integrated checks for conditional dependencies. All mission models discussed in the context of experiments in Section 5 use a Ψ that is the identity unless explicitly stated otherwise explicitly.

Observation model $O(s,m)$ is represented by a distinct $Mat\ O_{F_i}$ for each feature F_i. Probabilities are computed equivalent to T. With Ψ being the identity final probabilities can be computed directly:

$$O(s,m) = \prod_{i=1}^{n} \| O_{F_i}(fs, fm) \|_{row} \quad [3.32]$$

$$P(m|s) = \| O(s,m) \|_{row} \quad [3.33]$$

With non-identity Ψ, weighting factors ω_{fs} also utilized for transitions have to be applied:

$$T(s,m) = \sum_{fs \in s} \left(\omega_{fs} \sum_{fm' \in m'} \left(\prod_{i=1}^{n} \| O_{F_i}(fs, fm) \|_{row} \right) \right) \quad [3.34]$$

Reward model $R(s,a)$ is represented by a distinct $|FS| \times |A|$ - $Mat\ R_{F_i}$ for each feature F_i. The reward model can be computed with an identity Ψ:

$$R(s,a) = \prod_{i=1}^{n} R_{F_i}(fs,a) \quad [3.35]$$

or with combined states:

$$R(s,a) = \frac{1}{|s|} \sum_{fs \in s} \left(\prod_{i=1}^{n} R_{F_i}(fs,a) \right) \quad [3.36]$$

Basic expression processing provides a compact input interface to manipulate entries in the internal model matrices Mat. By these means, grouped characteristics arising from the feature-derived structure of the state space can be exploited. A rule is defined by an *access mask* indicating table group G, feature table F_i, state masks mk_{FS}, $mk_{FS'}$, action mask mk_A and observation mask mk_M, a *mode* and an arithmetic *expression*. The access mask selects matrices and their matrix elements to which the expression is applied. One or several table groups G are selected, each of which is composed of a set of tables, one for each feature. Either a single table or all within a group can be selected. Next, rows and columns in each selected table are picked by state, action or observation masks. A mask can select all, none or all except one feature state/measurement rows or columns corresponding to a feature, for example:

$$mk_{FS} = \{fs \in FS \,|\, \pi_{F_2}(fs) = c_4 \in F_2 \wedge \pi_{F_4}(fs) \neq c_2 \in F_4\} \quad [3.37]$$

This leads to all feature state tuples being selected with:

$$(c_1 \vee \ldots \vee c_{|F_1|} \in F_1) \wedge (c_4 \in F_2) \wedge (c_1 \vee \ldots \vee c_{|F_3|} \in F_3) \wedge (c_1 \vee c_3 \vee \ldots \vee c_{|F_4|} \in F_4) \quad [3.38]$$

By these means, expressions to manipulate transition effect frequencies can be adressed:

$$rule(G, F_i, mk_{FS}, mk_{FS'}, mode, exp), G = a \text{ (single action) or } G = A \text{ (all actions)} \quad [3.39]$$

The internal observation and reward model values can be addressed accordingly:

$$rule(G = O, F_i, mk_M, mk_{FS}, mode, exp) \quad [3.40]$$

$$rule(G = R, F_i, mk_{FS}, mk_A, mode, exp) \quad [3.41]$$

Convenience operations like clearing all corresponding columns or all corresponding entries in feature tables Mat_{F_i} except those currently addressed, can be flagged in the mode. Expressions that manipulate entries can either be numerical values directly written into entries or different kinds of arithmetic expressions. The latter include standard arithmetic expressions like addition and multiplication, but also more complex operations like computing discrete Gaussians over rows and columns. A detailed list of available expression can be found in [37] and [127].

By means of this expression-processing system, complex mission models can be composed from a limited set of instructions, which in turn are either hand modeled, generated by inference on background knowledge as discussed in the next Section or generated in the PbD process discussed in Section 4. Manual mission modeling for stand-alone evaluation of the execution-time decision-making architecture as discussed in Section 5.2 is only feasible with such a processing system. In automated mission model generation, both the analysis of human demonstrations stages (see Section 4.6) as well as complementary background knowledge inference (see Section 4.10) generate these intermediate rules to compile a final POMDP mission model.

3.7. Description Logic Based Background Knowledge

In contrast to settings with specific, one-time handcrafted missions as discussed in Section 2.2.6, more generic household settings require large numbers of different missions. Those typically include overlapping aspects, hence it is necessary to provide a representation and storage concept for characteristics covering several missions and thus multiple mission models.

This Section outlines a system organizing such long-term mission knowledge for service robots. The system supports model PbD, discussed in Section 4.10.

3.7.1. Scope of Background Knowledge for Mission Model Generation

In contrast to most typical background-knowledge systems, discussed in Section 2.6, incorporating inferred information during execution time, inference is performed offline during mission model generation in the presented system. Therefore, environment characteristics invariant between different missions, as outlined in Section 3.5, are most suitable to be exploited. There are three basic aspects to consider:

1. POMDP model component S, A, M, T, O or R
2. Environment invariance
3. Agent invariance

Some POMDP model components are more fundamental than others. As outlined in the previous and this Chapter, states S and actions A are the most basic model properties, grounding the model in the world. Observations M, transition model T and reward model R are dependent on states and actions. The observation model O can be interpreted as another layer of dependency. Accordingly, properties related to S and A alone typically have the highest invariance.

Environment invariance applies to aspects valid in any setting, for example when a robot manipulates any object together with a human, a situation may result it has never encountered previously, so in all cases additional stochasticity has to be added. In contrast, for instance, any local manipulation action will not directly influence a remote object, which reduces possible target states.

Agent invariance can be considered in two ways: a) characteristics independent of the agent (human or type of robot) and b) characteristics specific to an agent but valid in many similar situations (for instance, a service robot with limited perception and non-compliant manipulators is always at risk to jam itself when manipulating furniture).

An exhaustive analysis and derived taxonomy of invariance is beyond the scope of this thesis, but some specific properties can be determined based on these insights. Those types of mission-invariant aspects have a relationship to some mission-specific properties. The following list of aspects is considered in the presented system, with the former properties being mission-specific and the latter correspondingly derived from background knowledge [124]:

1. Related to A: Action cost negative rewards R_C.
2. Related to A: Robot-capability-specific error effect states S_E.
3. Related to A, S: Robot-capability-specific error effect probabilities T_E.
4. Related to A, S: Generic error recovery actions A_E.
5. Related to S, M: Information gain actions A_I.
6. Related to A, S: Information-gain action effect probabilities T_I.
7. Related to S, M: Skill component perception characteristics O.

All these aspects are specific to an agent but similar across different missions. Furthermore, most characteristics are environment invariant to a large extent and exploit fundamental model component dependencies. As discussed in Section 4.10, these robot-specific model properties cannot be learnt from analysis of human demonstrations, therefore robot scene analysis and trial learning has to be used to acquire these properties. Subsequently, preliminary models acquired from analysis of human demonstrations can be completed using that knowledge.

3.7.2. Description Logic Based Knowledge Processing

Background knowledge spanning multiple missions is represented by description logic (DL) based OWL [4] axioms, horn clauses in the form of SWRL rules [57], [55], [58], and additional OWL preprocessing rules expressed in OPPL [62]. Knowledge represented by axioms, clauses and rules can be stored persistently. It is used by inference and rule-processing engines during learning as well as POMDP model compilation [124].

Given an existing, persistent knowledge base and a preliminary POMDP model as input, DL inference, SWRL and OPPL rule processing generate properties completing the model. The persistent knowledge base represents a DL ontology containing abstract concepts of entities in POMDP model components. Such entities may be states s, actions a, observations m, transitions t or rewards r. After a preliminary POMDP model is supplied to the inference engine, new model component entity instances are generated in the ontology.

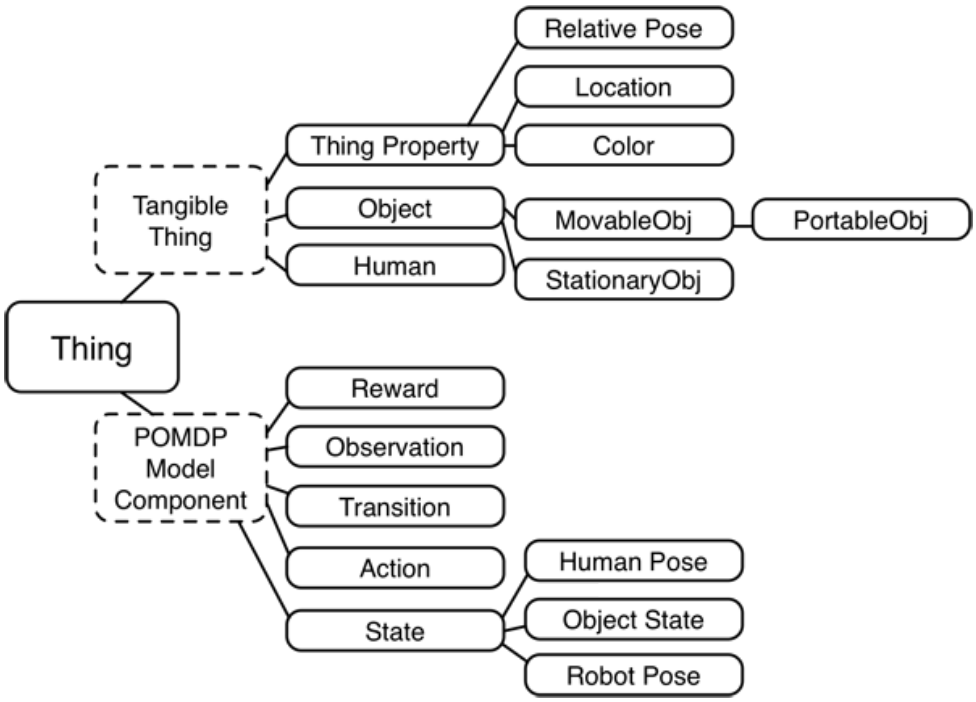


Figure 3.21.: Schematic view of the upper hierarchy levels of the ontology. - [125]

When confronted with a new preliminary model, entities s, a, m, t, r representing components in this model are realized and classified in the persistent ontology. Realization means that each inserted model entity is matched with a concept entity in the persistent ontology. Matching is performed by analyzing model entity description tags as described in Section 4.9. Classification generates new concept entities representing whole model feature state spaces.

Next, OPPL rules are processed to generate additional entities. Hence, after realization, an ontology is analyzed for entities from which related new entities can be derived. For example, given a set of action entities A, new error state entities S_E can be added.

Subsequently, after adding new entities, the DL reasoner Pellet [140] is able to infer new related entity properties automatically, for instance by linking existing transition entities t with new state s and action entities a. In this inference stage, both OWL semantics as well as SWRL rules are applied by Pellet. Afterwards, new POMDP model knowledge exists in the extended, non-persistent ontology. All relevant instance entities directly correspond to the by now enhanced POMDP model. These instances can then be assembled and exported into a final POMDP model. Next, the knowledge ontology hierarchy is outlined, followed by examples describing axioms and rules for concepts as well as entities.

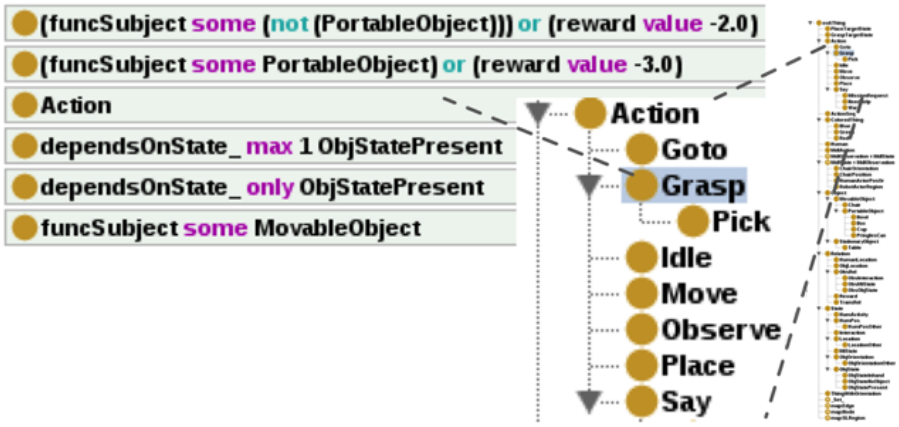


Figure 3.22.: Excerpt of persistent TBox knowledge in an OWL ontology editor. - [124]

3.7.3. Service Robot Knowledge Ontology Hierarchy

As always in DL, the root of an ontology – the universal concept – is $\top$. Below the root, concepts describing the tangible world and concepts describing POMDP model components are distinguished as sketched in Figure 3.21. Instances and their respective properties are expressed within that hierarchy as highlighted by the following examples. A *red cup* instance entity is described by:

$$Cup \sqsubseteq PortableObj \sqsubseteq MovableObj \sqsubseteq Object \sqsubseteq TangibleThing \sqsubseteq \top \quad [3.42]$$

$$Red \sqsubseteq Color \sqsubseteq ThingProperty \sqsubseteq TangibleThing \sqsubseteq \top \quad [3.43]$$

$$Cup(CupRed), Red(CupRed) \quad [3.44]$$

In this case, Expression 3.42, a DL axiom describes the concept *Cup* being an *Object* while Expression 3.43 describes *Red* being a *Color*. Consequently, instance *CupRed* is both an *Object* and a *Color* as given by Expression 3.44. A *goto PullStartPos* is an instance described by:

$$Goto \sqsubseteq Action \sqsubseteq POMDPModelComponent \sqsubseteq \top \quad [3.45]$$

$$Goto \sqsubseteq (\forall destinationRegion.Location) \quad [3.46]$$

$$Goto \sqsubseteq (\exists reward.\{-1.0\}) \quad [3.47]$$

$$Goto(GotoPullStartPos) \quad [3.48]$$

$$destinationRegion(GotoPullStartPos, PullStartPos) \quad [3.49]$$

Goto is an *Action* as described by the DL axiom given by Expression 3.45. A certain target *Location* is linked as given by Expression 3.46. Every *Goto* has a corresponding reward value defined by the anonymous concept stated in Expression 3.47. The instance is formed by the DL fact stated in Expression 3.48 and location instantiated by Expression 3.49.

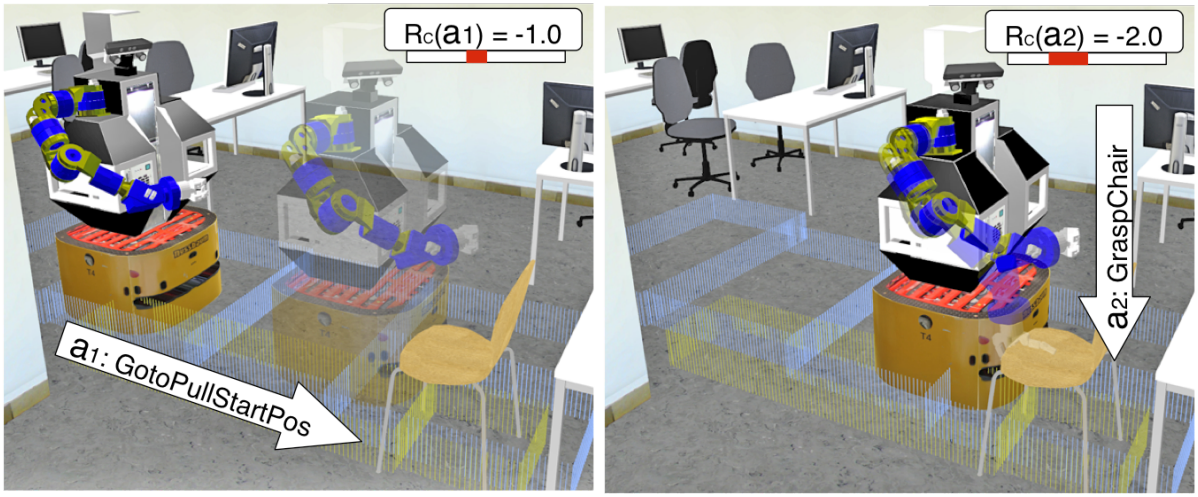


Figure 3.23.: *Illustrative example*: Reward cost rules for actions involving different effort. - [125]

3.7.4. Modeling Service Robot Mission Knowledge

No state or action instance entities exist in persistent knowledge. Instances are realized on request. Thus, domain characteristics are defined by *POMDP Model Component* concepts. Hence, knowledge valid in a skill domain for multiple missions is encoded as DL axioms.

POMDP Model Component Concept Examples show various axiom uses.
An example for a feature state space F concept is $ObjectState \sim F_{object-state}$, (Figure 3.22):

$$ObjectState \sqsubseteq State \sqsubseteq POMDPModelComponent \sqsubseteq \top \quad [3.50]$$

An example for an action type concept A is $Grasp$:

$$Grasp \sqsubseteq Action \sqsubseteq POMDPModelComponent \sqsubseteq \top \quad [3.51]$$

$$Grasp \sqsubseteq (\exists relatedObject.MovableObj) \quad [3.52]$$

For DL axioms describing transitions concepts T, the DL modeling technqiue *Concept Product* [119] $\circ$ is necessary as outlined by the following axioms and facts:

$$RobotPose \sqsubseteq (\exists inSet.\{RobotPoseSet\}) \quad [3.53]$$

$$cannotModifyAny \circ inSet^{-} \sqsubseteq cannotModifyInferred \quad [3.54]$$

$$Set(RobotPoseSet) \quad [3.55]$$

Given these axioms and facts, a transition concept for $Grasp$ can be defined:

$$Grasp \sqsubseteq (\exists cannotModifyAny.\{RobotPoseSet\}) \quad [3.56]$$

This axiom expresses that a *Grasp* action will never change the robot location concerning mobility. Modeling transition probabilities is discussed in Section 4.10.
A reward concept R can be expressed by:

$$Grasp \sqsubseteq (((\exists relatedObject.PortableObj) \sqcap (\exists reward.\{-1\})) \quad [3.57]$$
$$\sqcup((\exists relatedObject.\neg PortableObj) \sqcap (\exists reward.\{-2\}))) \quad [3.58]$$

This DL axiom states that costs of grasping non-portable objects are higher than those of grasping portable objects. Further illustrative examples are depicted in Figure 3.23.

Instance generating OPPL rules are applied after preliminary model input. Table C.2 shows an example of such an OPPL rule.

Knowledge acquisition can be differentiated into three different types:

1. Initial bootstrapping of central concepts in the upper layers of the ontology
2. Learning persistently stored concepts and corresponding parameters incrementally
3. Learning non-persistent knowledge for missions on-the-fly

Because initial bootstrapping has to be performed only once, with a resulting ontology remaining a valid skeleton for all missions, design by human experts is feasible. However, automatic completion of axioms based on those created by an expert can be performed by a bootstrapping process, as discussed in [124]. On the other hand, more specific persistent knowledge should be added incrementally to such an initial skeleton and based on experience as discussed in Section 4.10.3. Non-persistent knowledge for specific missions, generated by refinement stages during the PbD process, is added on the fly. Based on such knowledge, preliminary POMDP mission models, acquired by PbD can be completed as outlined in Section 4.10.

3.8. Modeling Conclusions and Discussion

This Chapter has presented a concept of how to model abstract level, strategic action selection on a service robot with multiple, highly diverse skill domains. Main points addressed are modeling states s representing classes of similar situations E_s, deriving observation uncertainty $p(m|s)$ from perception peculiarities as well as modeling abstract actions a as complex tasks and determining their effects $p(s'|s,a)$.

Two major challenges remain, however: model complexity and model parameter acquisition. The issue of model complexity could be solved by increasingly crude approximations of model

properties in a hierarchical manner. Such an approach could make larger, complex missions feasible. While such an investigation is beyond the scope of this thesis, current research in the community is promising for tackling that challenge.

On the other hand, the issue of model parameter acquisition is more immediate even for simple missions. Comfortably generating a specific set of S, A, M, T, O, R sufficiently reflecting a certain robot mission, is non-trivial but necessary to enable abstract-level autonomous decision making in practice. Hence, a PbD solution tackling that challenge is presented in the next Chapter. In summary, this Chapter is both a presentation of the grounding and the resulting model output of the PbD process, and a motivation why such a learning process is indispensable.

4. Programming by Demonstation of Probabilistic Decision Making

Manually and explicitly programming complex probabilistic decision-making models for service robot missions covering multiple skill domains in everyday settings is impractical, as explained in the previous Chapter. Instead, recording and analysis of natural human demonstrations of these missions can be conducted in order to generate decision-making mission models automatically.

The aim is to derive a model covering mission characteristics and symbol grounding as introduced previously, which are in turn suitable to be executed by the online decision-making system. Accordingly, the execution time rational agent cycle *perceive – decide – act* is complemented by a learning stage process *record – process – model*.

While real execution-time learning is not considered within the framework of this system, model refinement by assessment of situations encountered during mission execution could be included. Thus, the conceptual architecture does not prevent execution-time learning and potential execution-time model updates with online policy recomputation. In the following, the discussion distinguishes clearly between learning and execution stages, only for matters of focussed investigation.

Mission model PbD is a multi-stage process incorporating algorithmic procedures that use conceptual definitions presented in the previous Chapter. Three main groups of stages can be distinguished:

1. Recording, segmentation and abstraction
2. Model generation and model space exploration
3. Autonomous model refinement

The first portion of the process deals with transforming demonstration data into a representation compatible with mission model level of abstraction. In the next portion, autonomous model analysis and a resulting interactive dialog seek to acquire missing information from additional demonstrations. Finally, autonomous model refinement utilizes highly diverse methods to complement the model with information that is robot-specific and cannot be learnt from human teachers.

4.1. Recording Natural Human Demonstrations of Service Missions

Human demonstrations of service missions in the context of the presented system involve an indoor scene, a number of manipulatable objects and at least one, at most two humans. One human teacher takes the intended role of the robot, thus acting as the *robot role demonstrating actor* (*RR*) while a second, optional human can take the role of humans interacting with the robot in the mission, thus being the *human interaction role demonstrating actor* (*HR*).

By observing and recording demonstrations with perception components available during mission execution, the robot is able to relate to the same aspects of the world while learning as during execution (outlined in Section 3.2). By these means, state and action grounding is consistent. Additionally, the setup becomes flexible and natural: a robot observes a demonstration scene actively, following demonstrating humans with cameras on its neck and potentially its mobile base. To humans it appears as if a studying, physical human-like agent actively observes a teacher, thus making it easier to understand on which spatial parts of the demonstration observation attention is focussed.

Compared to dedicated recording centers and smart rooms, the main disadvantage is a lack of observation precision. Yet, on the one hand, mission observation is coarse-grained compared to skill PbD. Hence, there is less need for highly precise measurements. On the other hand, additional external sensors can be integrated into the presented concept as the robot is modeled just as a collection of perception skill components. Therefore, integration of further sensors, like headsets for speech recognition – as used in the presented system – wireless data-gloves retrieving human finger-joint angles, force sensors and external cameras for object localization is suitable, leading to a mixed robot and smart room based observation setup.

In the setup utilized in the presented system and shown in Figure 4.1, the following scene properties are observed with perception-skill components presented in Section 3.1.1:

- RR world-relative spatial pose $RR.pose$
- RR symbolic body motion activity $RR.act$
- RR spoken utterance $RR.utt$
- HR world-relative spatial pose $HR.pose$
- HR symbolic body motion activity $HR.act$
- HR spoken utterance $HR.utt$
- List of world-relative known object poses $Objs.pose$

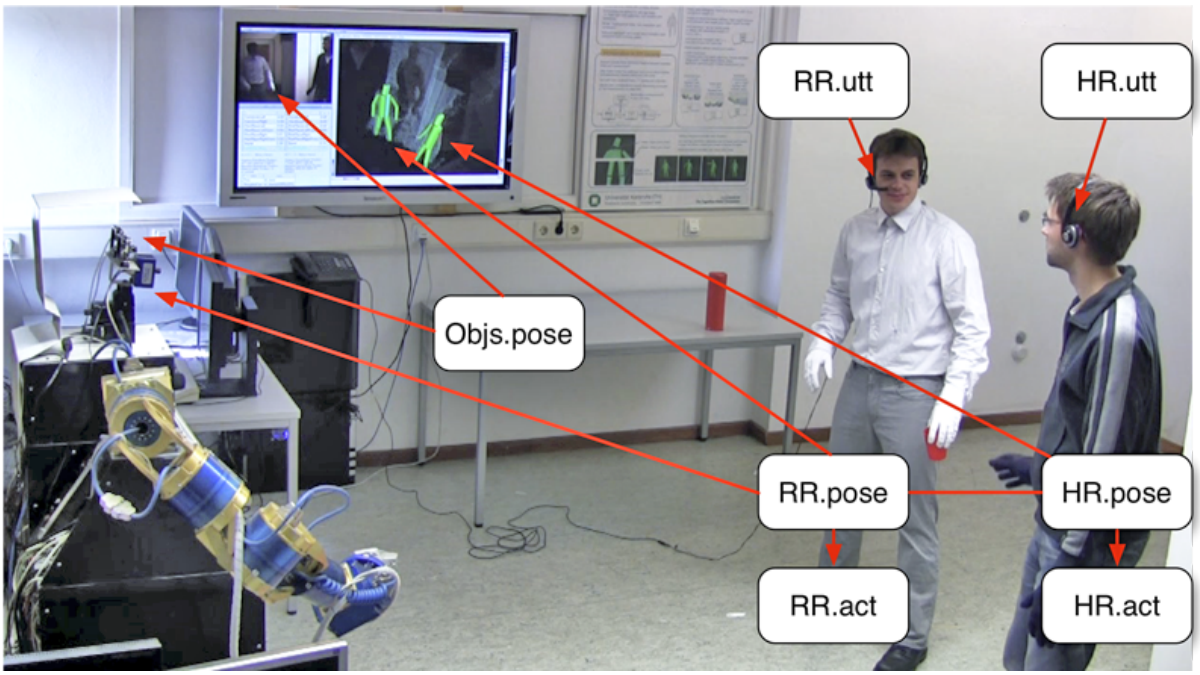


Figure 4.1.: Demonstration setup with observed scene properties annotated. Gloves are worn in this demonstration, discussed in Section 5.5, to improve in-hand object localization. - [131]

Hardware-specific setup details (a setup is depicted in Figure 4.1) are discussed in Section 5.1. To keep important spatial portions of demonstrations in the robots field of view, there are two mechanisms of attention. Primarily, the robot actively follows the RR with its head, to keep the RR's pose in the center of its field of vision. However, when objects or an HR are detected in the RR's vicinity, focus is shifted to keep both in the field of view during RR object manipulation or RR-HR interaction. Furthermore, a mechanism allows the RR to perform pointing gestures signalling the observing robot to shift its focus of attention as shown in Figure 4.2.

Apart from attention control, there is no actuation performed by the robot during demonstration observation. Perception skills continuously record data as described in Section 3.1.1. Perception skills have different temporal frequencies, and utterance has an event-like temporal quality. Therefore, each demonstration recording data point $obs(t)$ contains the last valid information of a perception skill. Frequencies differ significantly with human-pose tracking reaching 20 Hz, localization of small objects around 3 Hz, and localization of furniture approximately 0.3 Hz. Processing stages discussed in Sections 4.2 and 4.3 consider these differences. Demonstration observations result in time series of recording data points, a *trace*, Obs:

$$Obs := (obs(t_1), \ldots, obs(t_n)) \quad [4.1]$$

$$obs(t) = (RR.pose(t), RR.act(t), RR.utt(t), HR.pose(t), HR.act(t), HR.utt(t), Objs.pose(t)) \quad [4.2]$$

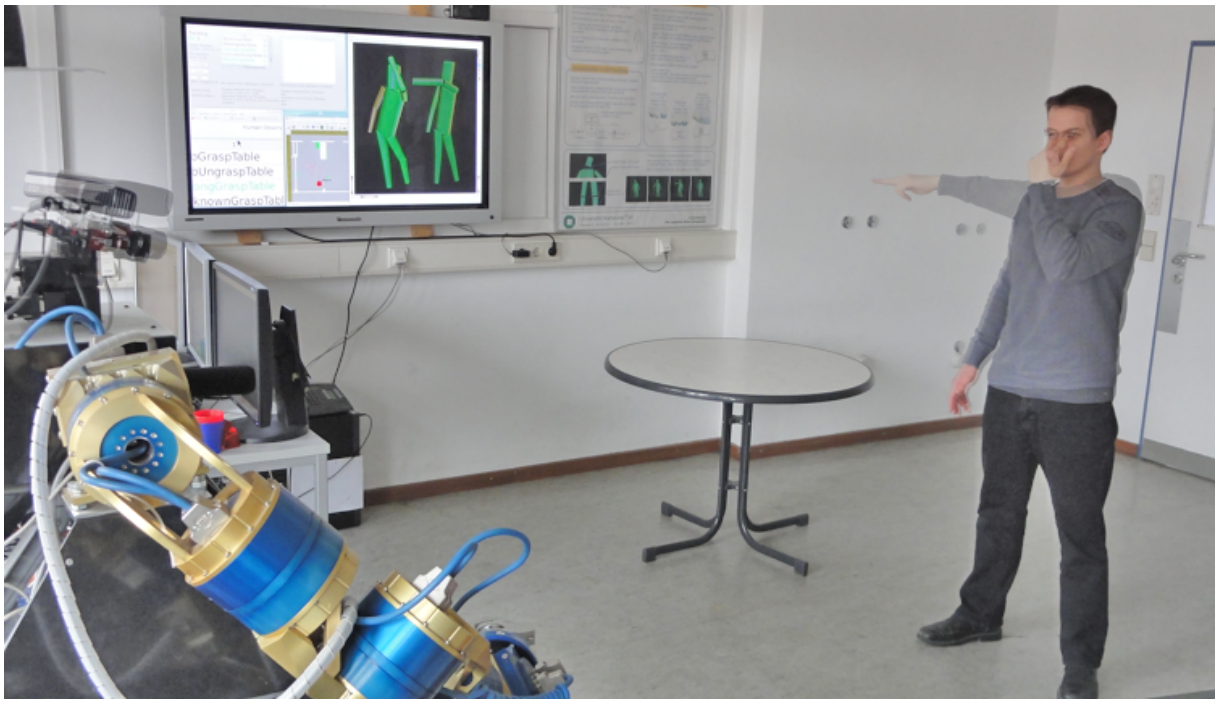


Figure 4.2.: Illustration of the attention mechanism consisting of two special gestures: first pointing at the own eyes and then left or right. - [125]

Conceptually, some of the components reflect state aspects in the mission, while others reflect actions. State aspects of environment situation E_s are represented by $RR.pose(t)$, $HR.pose(t)$, $HR.act(t)$, $HR.utt(t)$, $Objs.pose(t)$ while actions reflecting agent activity G_a are represented by $RR.act(t)$, $RR.utt(t)$ and $\Delta(RR.pose(t))$. By these means, both regarding E_s and G_a, skill domains of mobility, natural human-robot interaction and object manipulation are represented [132]. Consequently, recordings results in a trace representing a mission demonstration:

$$\begin{aligned} obs(t) =& (E_s(t), G_a(t)) = \\ & (RR.pose(t), HR.pose(t), HR.act(t), HR.utt(t), Objs.pose(t)), \\ & (\Delta(RR.pose(t)), RR.act(t), RR.utt(t)) \end{aligned} \quad [4.3]$$

While the process stages presented below are mostly generic and typically scale easily to new scene properties, such as contact forces, not included above, discussion will refer to listed properties. This is done because those properties already incorporate a wide set of skill domains and explanation is better suited using tangible properties. Scalability of process stages to new scene properties is discussed where applicable, otherwise it is either assumed to be trivial or not applicable, for instance if a single scene property or skill domain is the focus of discussion. In the following, a set of multiple, independent mission demonstrations $Demo := (Obs_1, \ldots, Obs_p)$ is assumed.

4.2. Generating State Descriptions Based on Demonstrations

While demonstration traces already represent E_s and G_a, structured along skill domains, no explicit state or action set and grounding exists at that point. Hence, as motivated in Section 3.2.2, feature state space and corresponding input domain discretization has to be inferred from demonstration data [147]. Determining suitable feature discretization applies to $RR.pose(t)$, $HR.pose(t)$ and $Objs.pose(t)$ while considering $delta(RR.pose(t))$, $RR.act(t)$ and $RR.utt(t)$ in the presented system. It is achieved by spatially clustering filtered data point components across multiple input values as discussed below. All illustrative Figures in this Section show data regarding mission CESM-1, which is discussed in Sections 5.1.2 and 5.3. Considered resulting features are:

- $f_{feat_{pose}}(RR.pose) \rightarrow c_x^{self-pose}$
- $f_{feat_{human}}(HR.pose) \rightarrow c_x^{human-pose}$
- $f_{feat_{objs}}(obj.poses) \rightarrow c_x^{objposes}$ here with $objs := furni-state \vee small-obj-state$

4.2.1. Data Preparation

Raw trace data points contain pose information as a 6D-vector (x, y, z, r, p, θ). Irrelevant dimensions can be pruned when values do not change significantly over all traces, such as human or furniture pose z, r, p in many cases. Yet, a small cup that is mostly manipulated on a table may fall over, thus having a highly varying z value, which represents a distinct, significant state.

$$\forall val_{x_i} \in d_{pose} : f_{feat_{pose}} \leftarrow \begin{cases} \emptyset & \forall obs(t) \in Demo : \mu - \varepsilon < val_{x_i}(t) < \mu + \varepsilon \\ val_{x_i} & else \end{cases} \quad [4.4]$$

Therefore, small objects and furniture objects are usually represented in two different features:

- $f_{small-obj-state}(obj.poses) \rightarrow c_x^{smallobjposes}$
- $f_{furni-state}(obj.poses) \rightarrow c_x^{furnposes}$

Data preprocessing can either filter predefined object symbol lists or perform size computations for correct classification. Next, interpolation of trace data has to filter outliers and smooth temporal as well as spatial gaps in recorded data. Outliers arise from noise of imperfect recordings. Recorded poses that denote improbably high velocities, spatial leaps, in human and object poses can be discarded:

$$\forall Obs \in Demo, pose_{x_i}(t) = \begin{cases} \emptyset & pose_{x_i}(t) - pose_{x_i}(t-1) > \Delta_{possible} \\ pose_{x_i}(t) & else \end{cases} \quad [4.5]$$

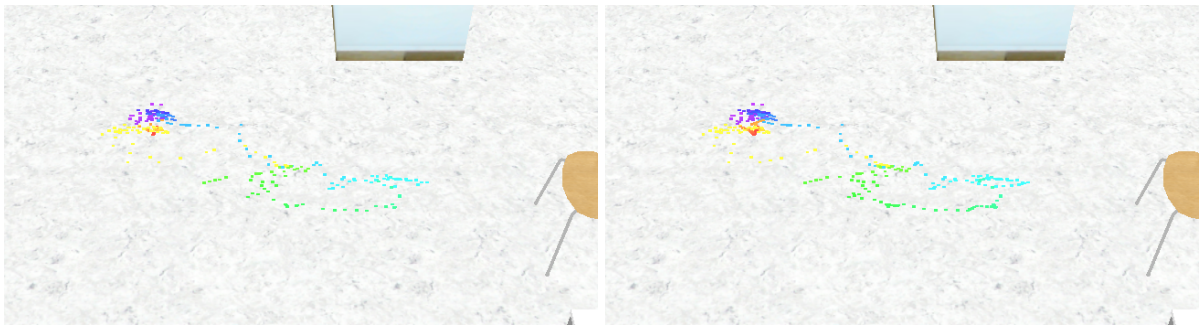


Figure 4.3.: Raw data of one demonstration (left) and temporal interpolation (right). - [147]

In the system used for evaluation, detecting human pose outliers is easier because of much higher data-point frequency compared to object localization. For robust clustering as described in the next Section, irregularly sparse data episodes resulting from recording peculiarities have to be interpolated. Linear temporal and spatial interpolation increases data point density without significantly altering recorded information as illustrated in Figure 4.3:

$$\forall Obs \in Demo, obs(t) \leftarrow \begin{cases} pose_{x_i}(t') & pose_{x_i}(t) - pose_{x_i}(t-1) > \Delta_{min-step} \\ pose_{x_i}(t') & time(t) - time(t-1) > time_{min-step} \end{cases} \quad [4.6]$$

$$pose_{x_i}(t') = pose_{x_i}(t-1) + \frac{pose_{x_i}(t) - pose_{x_i}(t-1)}{2} \quad [4.7]$$

Raw pose information is then augmented by derived attributes for each data point. $RR.pose$ and $HR.pose$ data is augmented by a *motion state* (mos) attribute. It is derived by a preliminary segmentation step based on pose difference, which is necessary to select suitable clustering methods for different kinds of data points later on. Three types of motion state can be distinguished: *standing*, *moving* and *walking*. For a segment $T_{int} := \{t_1, \ldots, t_n\}$ of data points $obs(t_i), t \in T_{int}$, the motion state is *standing* when the maximum pose difference is below a certain threshold Δ_s, *walking* when above a threshold Δ_w and *moving* when none of the other two holds:

$$max_{t_a,t_b \in T_{int}} |pose_{x_i}(t_a) - pose_{x_i}(t_b)| < \Delta_s \Rightarrow mos(Obs(T_{int})) = standing \quad [4.8]$$

$$max_{t_a,t_b \in T_{int}} |pose_{x_i}(t_a) - pose_{x_i}(t_b)| > \Delta_w \Rightarrow mos(Obs(T_{int})) = walking \quad [4.9]$$

Segmentation starts with a single data point $pose_{x_i}(t)$ and if the point is classified as *standing* or *walking* increases the window interval T_{int} until the motion state changes. Maximum intervals with stable motion states are segmented and the rest is tagged as *moving*. Furthermore, data points containing a certain human body activity or dialog state can be additionally filtered. Figure 4.4 shows examples of filtered data.

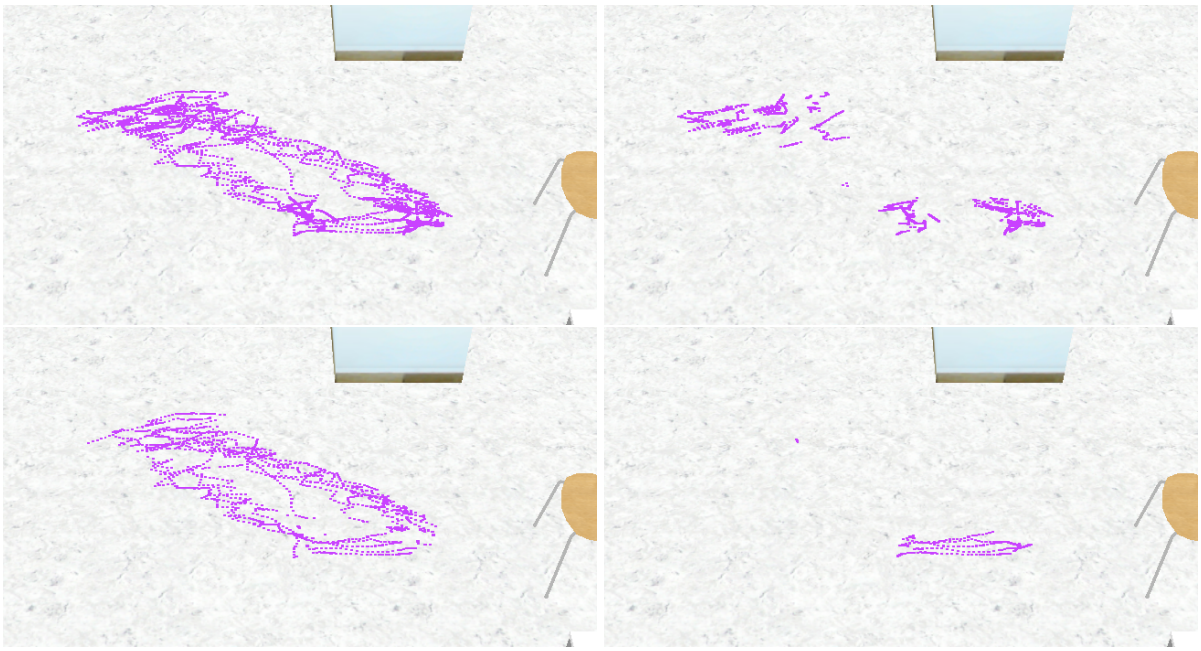


Figure 4.4.: Top left: Original data from five demonstrations. Filters clockwise: *standing*, human activity "Grasp", *walking*. - [147]

$Obj.pose$ data is augmented by an *object state* attribute, while object *motion states* are refined considering perception peculiarities. Object localization can be lagging because of skill component performance or occlusions during manipulation demonstrations. Object pose change timing is adjusted by matching $RR.act$ manipulation activities in object vicinity. Object pose changes during demonstration usually indicate a transport manipulation activity. If a related $RR.act$ was detected shortly before object pose changes, this manipulation activity is assumed to be the trigger. Accordingly, the start of a change in object pose is set to that time point, which is a form of temporal interpolation and an additional *transport* object property added to data points during object pose change.

4.2.2. Recording Data Point Clustering

While clustering can be performed on an arbitrary number of dimensions, practical results are best when dimensionality is reduced. Therefore, clustering is performed on (x,y), z and θ independently which is consistent with the way state grounding descriptions are represented in feature filters and manual category design as outlined in Section 3.2.

First, data point types are distinguished using augmented properties, such as activity filters or motion state filters. Stationary and non-stationary poses represent distinct aspects in grounding on the one hand and have different spatial characteristics on the other. In the way abstract missions are represented, state descriptions primarily arrange around stationary or quasi-stationary

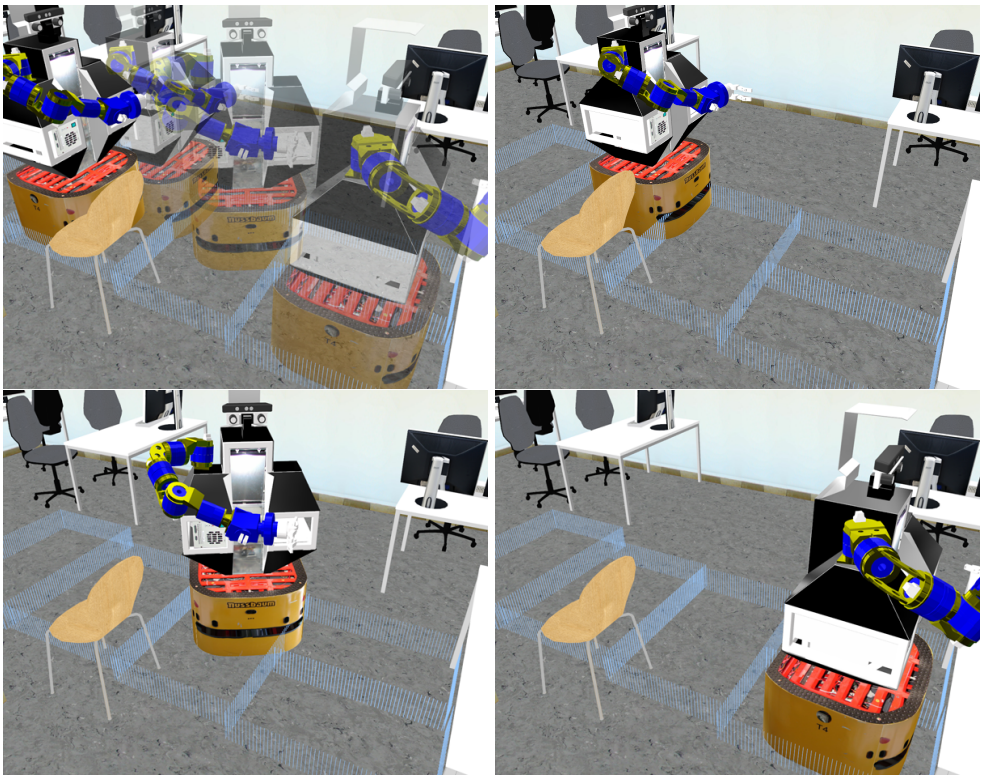


Figure 4.5.: *Illustrative example*: Considering several potential action effects a priori (top left). Afterwards: moved only a little bit, leading to a static transition (top right), stopped half way (bottom left), successful arrival (bottom right). - [125]

environment aspects, while non-stationary aspects typically relate to actions. Yet, each environment situation has to be represented by a distinct state, even during action execution.

While on the abstract level with a discrete time representation, the state is in a limbo after starting but not having finished an action, there has to be a distinct state representation after the action terminates. In practice, for example with a stochastic robot goto action, it is typically modelled by discrete regions as discussed in Sections 2.2.6, 3.2. Thus, the robot has to be in a distinct region after the goto action terminated – if it moved only a little bit and got stuck that may be the region in which it started, as illustrated in Figure 4.5, top right. Thus target regions must represent states, but furthermore all potential transition areas have to be allocated to a relevant state, in case an action terminates there as shown in Figure 4.5. The same principle holds for manipulation and human-robot interaction and is implicitly considered in manual feature modeling. Automatic feature generation, however, has to consider it using two distinct steps:

1. Primary clustering on $mos(obs(t)) = stationary$, with stationary $\sim$ standing, non-moving
2. Secondary clustering on $mos(obs(t)) = transfer$, with transfer $\sim$ walking, transport

Such clear distinction enables application of different clustering methods that can account for different spatial characteristics. While interpolation and preprocessing guarantee a certain minimum density, stationary clusters have a high point density, but transfer clusters do not.

Clustering method choice accounts for these differences (see Section 2.3.1), applying DB-Scan for dense stationary poses and k-means as well as EM with weighted k-quality measures DB, SD and XB index, as discussed in Section D.2, for sparse transfer poses. The result is a number k_s of stationary and a number k_t of transfer clusters for a set of observation *Demo*, with unique data point (pose) assignment for each perception skill input domain.

4.2.3. Category Boundary Computation

While a set of clusters represents a feature state space, including approximate spatial extent of respective categories (feature states), precise discretization cannot be immediately deduced. Execution-time feature mapping as defined in Section 3.2 is able to handle discretization descriptions based on multiple rectangle pixels or single rectangles. Therefore, a category boundary delineation has to be computed based on computed clusters. For automatic generation, the focus was on the single-rectangle representation. These are typically expressive enough, simple to store and understand, allow fast belief computation and can be used for evaluation of automatic against manual expert generation.

However, finding a suitable set of non-overlapping, adjoined multi-dimensional rectangles (*boxes*) representing clusters – as required by execution-time state computation – is not trivial. Both stationary and transfer clusters have to be considered and mostly enclosed by the set of all boxes. The goal is to compute a flexible, grid-like, topological partition of the space. It is achieved by using k-d-trees, a type of Binary Space Partitioning, to partition spaces hierarchically. First, clusters have to be represented by a minimal *cluster enclosing box* (CEB), which contains $(1-\varepsilon)*|X|$ cluster points. ε was determined empirically to be 0.02. By these means, extreme outliers can be ignored when computing a CEB. As CEBs are not adjoined, the aim of boundary computation is to find suitable adjoined category limit boxes c_i, one enclosing each CEB b_i.

First, upper and lower CEB neighbors $un_{x_k}(b_i), ln_{x_k}(b_i)$ in each dimension x_k are computed. For a given CEB in a dimension x_k, the lower boundary of the upper neighbor is closest to, but not less than the upper boundary of that CEB with the lower neighbor defined accordingly, as shown in Algorithm 2 in Appendix B.

Starting with one large box enclosing all CEBs, separating planes p are computed recursively, splitting one limit box l off at each time. At each step, a set of candidate separating planes is computed between each CEB and its upper neighbor pair incorporating the standard deviation

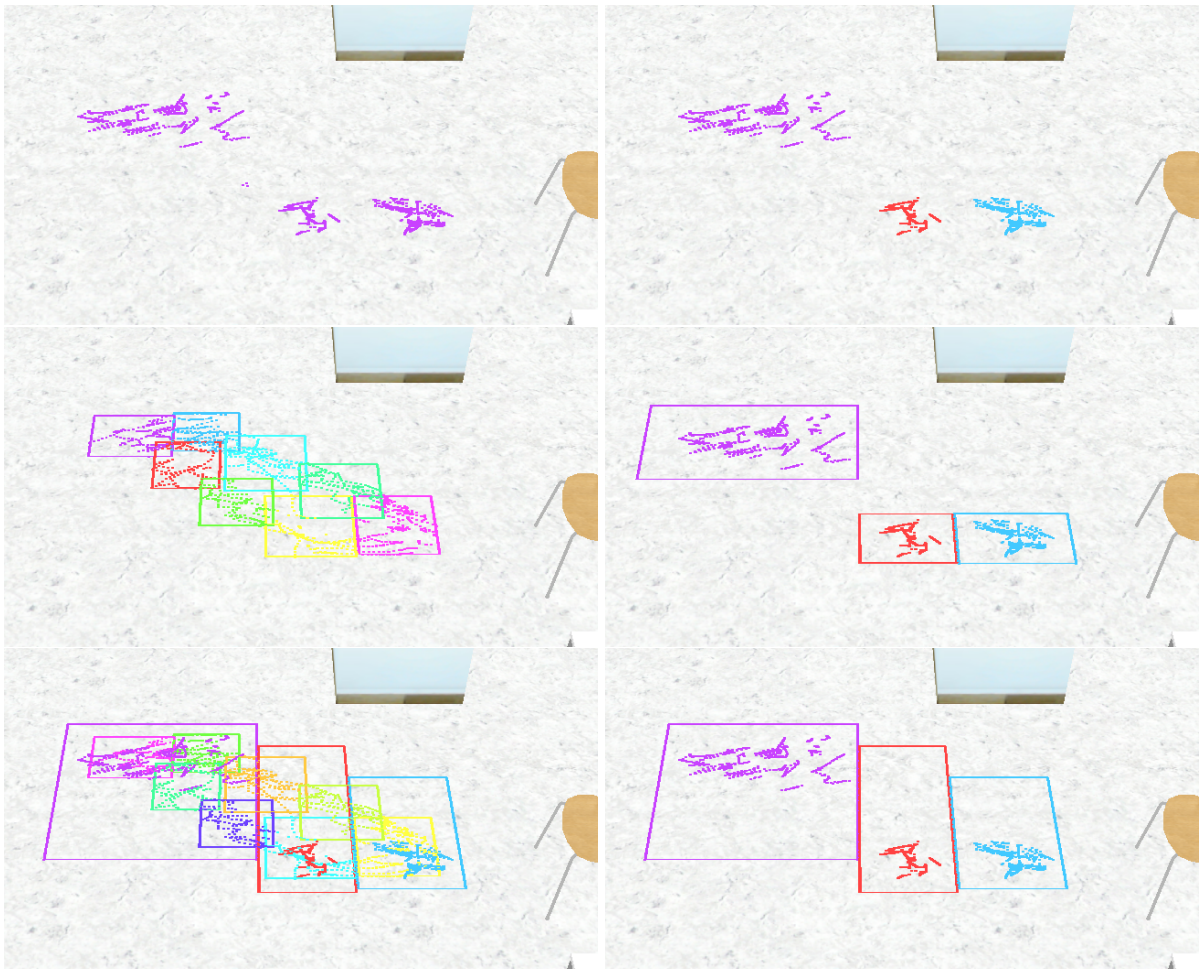


Figure 4.6.: From top left to bottom right: *standing* data for clustering, data clustered with DBscan, k-means clustered transition regions, regions without transitions, regions including transitions and final regions. - [147]

of both corresponding clusters . By these means, higher space requirements of spread-out clusters can be accounted for. In case another cluster intersects with the separator candidate, this separator is dropped . If no valid candidate remains in this recursion step, CEBs have to be shrunk, enclosing fewer cluster points $(1-\Delta_X) * |X_{current}|$, until one is found . However, if a certain ε_X limit with enclosed points $|(1-\varepsilon_X) * X_{cluster}|$ is surpassed, boundary computation terminates, indicating inappropriate clustering parameterization.

From the remaining set of candidates $p_{x_k,i}$, a best separator is chosen, by application of two metrics. Weighted direct neighborhood ω_h computes all CEB neighbors $b_h \in B_h$ to a separator candidate that are not partially occluded by another CEB. Influence of b_h on $p_{x_k,i}$ is computed by its average distance. Total direct neighborhood quality is the influence weighted sum of minimum distances to all $b_h \in B_h$. Cluster similarity ω_{sim} measures similarity of clusters on both sides of a separator, to support separation of dissimilar clusters. Similarity is computed

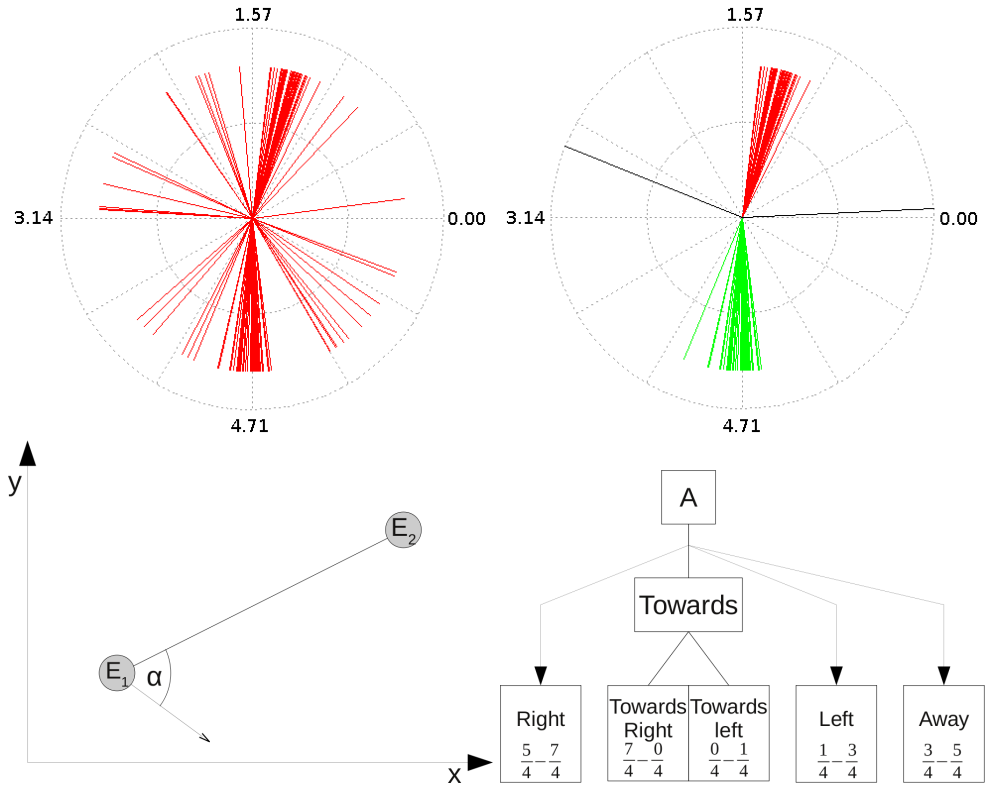


Figure 4.7.: From top left to bottom right: orientration data, orientations clustered with DBscan, relative orientation scheme, relative orientation symbol names. - [147]

from shape and size of clusters enclosed by CEB within groups B_{up} and B_{low}, with smaller values denoting higher similarity. Accordingly, the final quality measure $q(p_{x_k,i})$ combines large neighborhood distance ω_h with large similarity among clusters within a group using a metric preference parameter α .

After computation of separators and resulting adjoined limit boxes for stationary clusters, transfer clusters are integrated as shown in Figure 4.6. As discussed above, these regions may be cut arbitrarily: thus it is not necessary to include them in the elaborate separation process. Neighbouring limit boxes are extended into free space until all transfer cluster enclosing boxes are covered. Finally, boundaries towards directions with no neighboring clusters are determined by computing standard-deviation distances.

4.2.4. Secondary Attributes and Features

Further, secondary attributes are derived from primary spatial computation and either added to features as additional input domains val_{d_s} or used in standalone features.

Relative distances and orientations can be computed between all scene entities with distinct pose data. In the presented system, role demonstrator poses as well as object poses are encompassed. For each pose entity e_a, mean distance $\mu_d(e_a, C_i, e_b, C_j)$, distance standard deviation $\sigma_d(e_a, C_i, e_b, C_j)$ and mean angle $\alpha_{rel}(e_a, C_i, e_b, C_j)$ are computed for each cluster C_i to each pose cluster C_j of every other pose entity e_b. All cluster distances $d_{t_1}, d_{t_2} \in D_C(e_a, e_b)$ for each pose entity pair (e_a, e_b) are then checked for overlapping distance ranges $r_{t_1} := [\mu_d^{t_1} + \sigma_d^{t_1}, \mu_d^{t_1} - \sigma_d^{t_1}], r_{t_2} := [\mu_d^{t_2} + \sigma_d^{t_2}, \mu_d^{t_2} - \sigma_d^{t_2}]$. In case of overlap, the ranges are merged $r_{t_{n+1}} = r_{t_1} \bigcup r_{t_2}$. All distinct ranges that remain after merging all overlapping ones form the set of relative distance categories. Relative angle categories are processed in the same manner as shown in Figure 4.7. Both these relative categories can then describe category boundaries in *one dimension* of a feature with further input-values or form a feature on its own.

Events and object states such as symbolic human activity or object state attributes as discussed in Section 4.2, can be used as input value domain for features. For example, a change from *no object* state to *transport* may induce a new feature state separator in a previously homogeneously spatial cluster and resulting limit box. While computation of these more abstract "dimensions" is trivial, it is an important feature value domain that makes category (feature state) distinction more robust and pragmatically grounded. Finally, an additional feature state c_i^{OTHER} is included which covers all input values, which are not covered by the automatically generate categories.

4.2.5. Discretization Quality Measure

In general, all methods generating category separation descriptors $f_{g1:i}$ (feature-state grounding), can deliver varying results depending on parameterization and order of secondary attribute computations. When considering absolute spatial clustering followed by computation of secondary attributes, slightly superior early separator choices may lead to vastly inferior options later on.

To balance separation quality at different stages, a technique called *feature state description tree* (FSDT) is utilized. An FSDT consists of layers with alternating node types, each layer representing separators within a certain input value domain. These two node types are

- *Separation choice nodes* (SCN), choice node
- *Feature state separation nodes* (SN), separation node

An input domain which is processed before another input domain is defined as the former ranking higher than the latter one. Higher-ranking input domains appear on higher layers in the

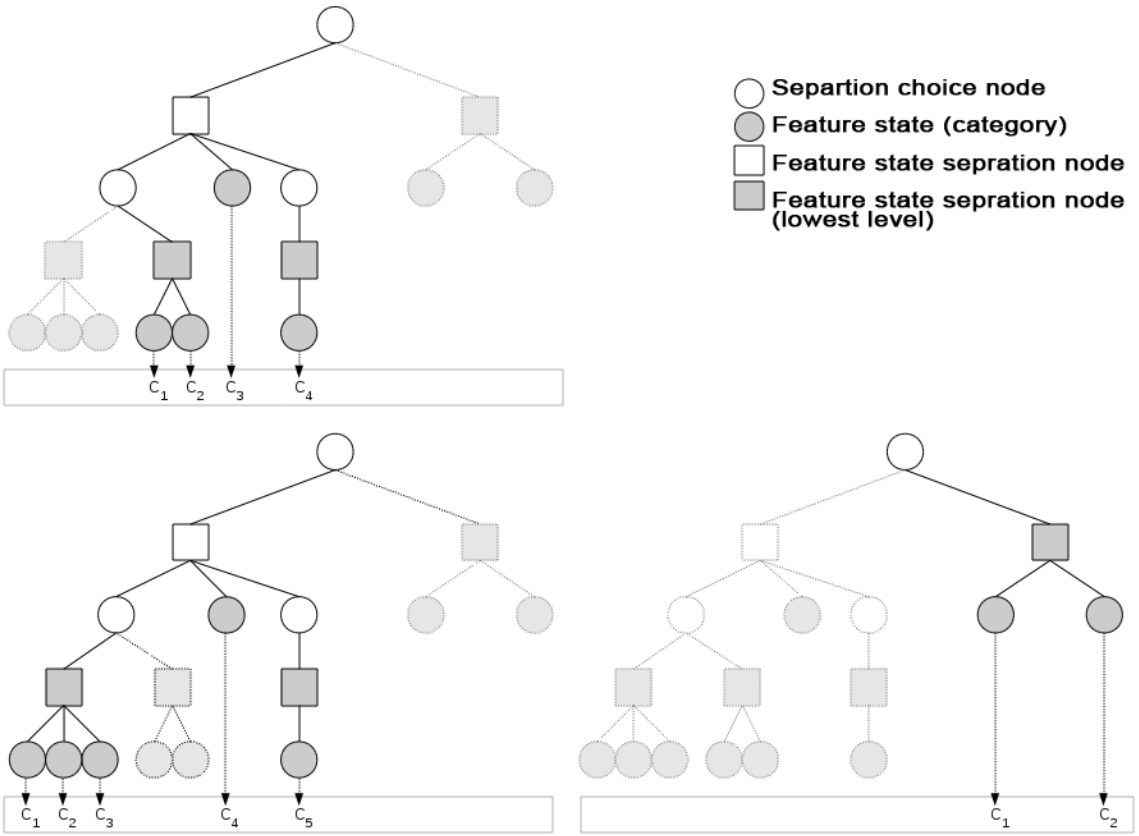


Figure 4.8.: Schematic view of an FSDT with different feature state space choices. - [147]

tree than lower-ranking ones. Each choice node contains a set of separation-node children, which represent different choices of category (feature-state) separation in lower ranking input domains. A certain category separation reflects a distinct input domain discretization using rectangles as described previously. Each separation node contains a set of choice node children, with each child representing a different set of category separation in the input value domain of the corresponding layer. On each layer, there can be *category leaf nodes* (CLN) that terminate a branch and represent the lowest ranking category separation in a branch. Complete category (feature-state) descriptions can then be formed by the path of each leaf node to the root of the tree. Each separation node (SN) adds the grounding description (value intervals) of the category in the corresponding input-value domain on that layer. However, paths from different leaves going through the same choice node (SCN) are mutually exclusive, as they represent different alternatives of splitting up a certain input-domain value. Therefore, different subtrees compose different choices for the feature-state space in a feature as shown in Figure 4.8. Those alternatives encompass both different discretization values as well as varying state-set sizes.

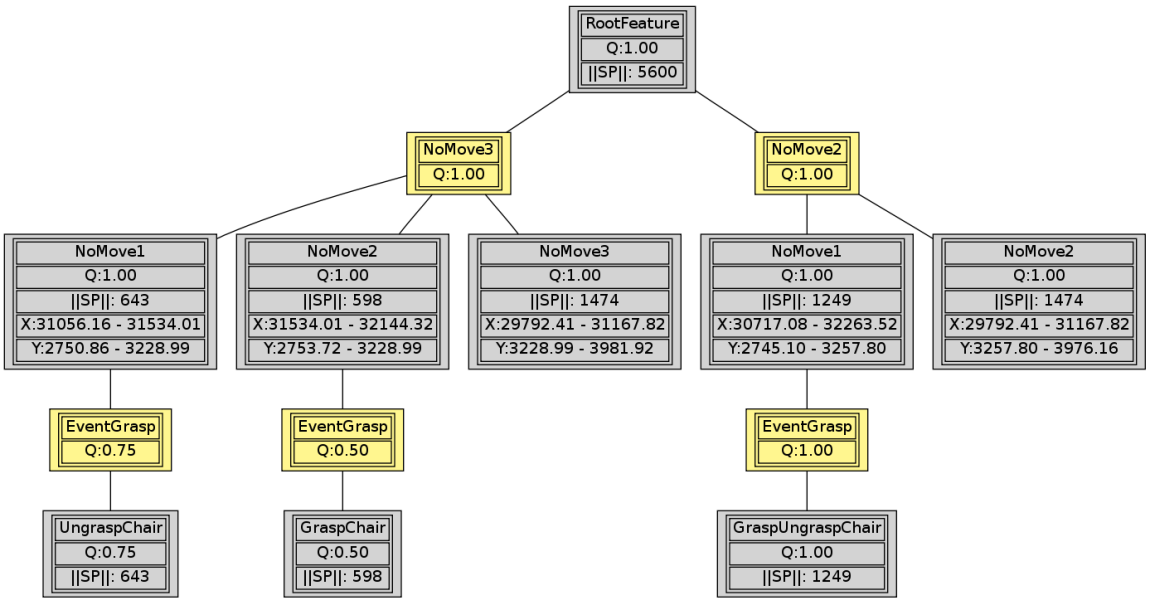


Figure 4.9.: Automatically generated scheme of a simple FSDT with quality values q given. - [147]

To evaluate category separation choices on all layers equally, not letting high-ranking separation choices dominate, path quality from leaf to root in all subtrees has to be evaluated. First, each separation technique represented by a layer has to assign a quality value $q(SN_i) := [0,1]$ to each separation node. Thus, before evaluating a full path, each choice node has to be rated. A choice node can be rated based on the quality values of included categories $c_i \in C_i \sim SN_i$.

Individual category (feature-state) quality depends on the frequency of occurrence of that category c in demonstrations $Obs \in Demo$, which indicates its relevance for the mission:

$$Obs_c = \{o | \exists obs(t) \in Obs : obs(t) = c\} \quad [4.10]$$

$$q_{demo}(c) = \frac{|Obs_c|}{|Obs|} \quad [4.11]$$

The quality also depends on clustering quality criteria as discussed in Section 4.2.2, such as DB, SD, XB metrics Q_k^{method}. The best resulting metric Q_{opt} is set to 1 and inferior results weighted in relation:

$$q_{sep}(c) = \frac{Q_j}{Q_{opt}} \quad [4.12]$$

These individual category qualities $q(c)$ have then to be weighted to get a separation node quality value $q(SN)$. Average weighting puts least emphasis on negative outliers:

$$q_{avg}(SN_i) = \frac{1}{n_i} \sum_{c \in C_i} q_c. \quad [4.13]$$

Product weighting puts multiple less optimal categories to a disadvantage:

$$q_{prod}(SN_i) = \prod_{c \in C_i} q_c. \quad [4.14]$$

Finally, minimum weighting puts most emphasis on a single negative outlier:

$$q_{ok}(SN_i) = \min_{c \in C_i} q_c. \quad [4.15]$$

These different category quality merging methods can then be empirically weighted.

Given individual layer quality values $q(SN_i)$ as shown in Figure 4.9, quality values for subtrees can be computed. Most weighting methods would prefer small trees which is not desirable. Instead, small and much larger high-quality trees have to be comparable based on category qualities alone. Therefore, the total sum of separator qualities is suitable:

$$q_T^T = \sum_{i \in T} q_{SN_i}. \quad [4.16]$$

It has a preference for larger trees with high-quality separations, yet small category sets with high quality can dominate larger ones with bad quality. The best complete subtree gives the final set of category discretization and thus optimal feature state space.

4.2.6. Conclusions and Covered State Concepts

While automatic state grounding generation from demonstration analysis as presented above can cover a wide range of skill domains, there are some limitations concerning conceptual coverage of dynamic aspects. These limitations, however, arise from the fundamental concept of Markov processes with discrete time and alternating states and actions. Therefore, clear distinction of stationary and non-stationary demonstration parts as performed by spatial clustering is just a consequence of that paradigm. Complex, dynamic behavior is subsumed in actions (subtasks), yet the given concept is able to deal with actions terminating anywhere (spatially) and at any time. By covering the input-value domains completely, any possible stochastic action result is covered by a valid state description, making any transition $T(s,a,s')$ occurring in practice well defined. Further details and software implementation specific aspects concerning the state mapping process stage are presented in [147]. In summary, the process of state grounding generation from mission demonstration analysis consists of the stages:

1. Data preparation improves recorded real-world demonstration data.

2. Spatial clustering determines data points associated with common abstract situations.

3. Boundary computation derives a separator representation usable by execution-time filter models as defined in Section 3.2.2.

4. Quality tree computation assesses alternatives of multiple input value domain separation.

At this stage, demonstration data representation concerning situations can be interpreted to change from an external view - a student watches a teacher - to an internal view: the student reasons about the mission as if it had performed the demonstration itself. This is seen as a shift in perspective and happens in humans learning from observation of other humans by activating the mirror neuron area in the human cortex [116]. The resulting feature state space set is the mode the robot uses to reason about the state of the world in task sequences it performs itself. Next, the same perspective shift is applied to actions.

4.3. Mapping Observed Tasks onto Executable Manipulation Strategies

Mission actions can be inferred from demonstrations in the skill domains of mobility, natural human-robot interactions and object manipulation: $\Delta(RR.pose(t))$, $RR.utt(t)$ and $RR.act(t)$. Inferring actions implies finding execution-time actions that correspond to human demonstration action aspects. In the domains of mobility and natural human-robot interaction, correspondences can be found easily. Utterances can be directly imitated while gestures may utilize a limited alphabet. Mobility actions encompass *goto* state actions which can be directly inferred from $\Delta(RR.pose(t))$. A component computing target poses within an abstract region for mobility actions is discussed in Section 4.11.

However, with object manipulation, the most complex robot skill domain, correspondence finding is not straight forward. Both symbolic classification of recording-time human manipulation actions discussed in Section 2.7.1 and execution-time manipulation action representation, discussed in Section 2.7.2, have their own, unconnected sets of reference symbols. Furthermore, the internal motion representation of both methods is highly distinct. Finally, the human body activity representation does not consider objects in the vicinity for classification.

Hence, there is need for a process stage that can map human demonstration activity classifiers in certain demonstrations to applicable manipulation strategies [42]. Because activity classifiers are coarse-grained and object-independent, one classifier may be mapped to different strategies in different demonstration situations. Thus, there is no static manipulation skill alphabet mapping, but an analysis specific to each demonstration and set of observations. Based on the classified human body activity – both its symbol and the actual trajectory as well as object poses – in a portion of a mission demonstration, a manipulation strategy suitable for execution by the robot in that situation is matched.

4.3.1. Manipulation Action Mapping Process

Given inputs are:

1. A single mission demonstration X_m with human hand pose information (contained in $RR.pose(t)$), object pose data $Objs.pose(t)$ as well as symbolic activity labels $RR.act(t)$.
2. A set of manipulation strategies mst_j, each with a set of demonstration trajectories $X_{j,i}$, with the trajectory describing the TCP pose relative to a reference object.

No human activity training data is necessary. As manipulation strategy trajectories $X_{j,i}$ are acquired automatically within strategy-PbD, there is no need for an extra demonstration stage. The available manipulation strategy set with corresponding training trajectories can be stored and extended in a manner decoupled from the mapping process. Therefore, strategy trajectories can be considered as the more static element, with those trajectories being reused, while each mission demonstration sequence X_m is processed only once. First, a given mission sequence X_m is segmented into symbolic human activity tokens $\{X_m^{act}\}$, by extracting segments with a common activity label:

$$X_m^{act} := x_m(t_s), \ldots, x_m(t_e) : \\ \{RR.act(t) \,|\, \forall t : t_s \leq t \leq t_e, RR.act(t_s) = RR.act(t) = RR.act(t_e)\} \qquad [4.17]$$

Subsequently, for each activity segment X_m^{act} all available $X_{j,i}$ are normalized relative to X_m^{act} in the space of (x,y,z,t) of the human tool center point (TCP), the hand pose relative to the object closest to the human TCP. Strategy trajectories which are spatially or temporally larger or smaller than the mission segment trajectory by a factor η: $\|X_{j,i}\|_{x,y,z} \geq \eta \, \|X_m^{act}\|_{x,y,z}$ or $\|X_{j,i}\|_t \geq \eta \, \|X_m^{act}\|_t$ for both $0 < \eta << 1$ or $\eta >> 1$ are discarded as there is definitely little similarity and normalization would distort the trajectory beyond usability for comparision.

After normalization, all TCP trajectories $X_m^{tcp,act}, X_{j,i}^{tcp}$ are defined within the same cartesian space, centered on the closest object and have the same number of trajectory data points $\forall i, j : |X_m^{tcp,act}| = |X_{j,i}^{tcp}|$. Therefore, two trajectory datapoints with the same index t are temporally equivalent. In case there are several close objects, the relevant object being manipulated can be determined by more elaborate manipulation strategy PbD analysis.

Next, for each manipulation strategy mst_j, a GMM model is computed on the set of mst training trajectories $X_{j,i}^{tcp}$ using EM as described in Section 2.3.2. Such a GMM is not suitable for computing similarity to a given $X_m^{tcp,act}$ trajectory, but a GMR can be computed from the GMM:

$$\Phi_j := \Phi(mst_j) = gmr(gmm(X_{j,i}^{tcp})), \{X_{j,1}^{tcp}, \ldots, X_{j,n}^{tcp}\} \rightarrow mst_j \qquad [4.18]$$

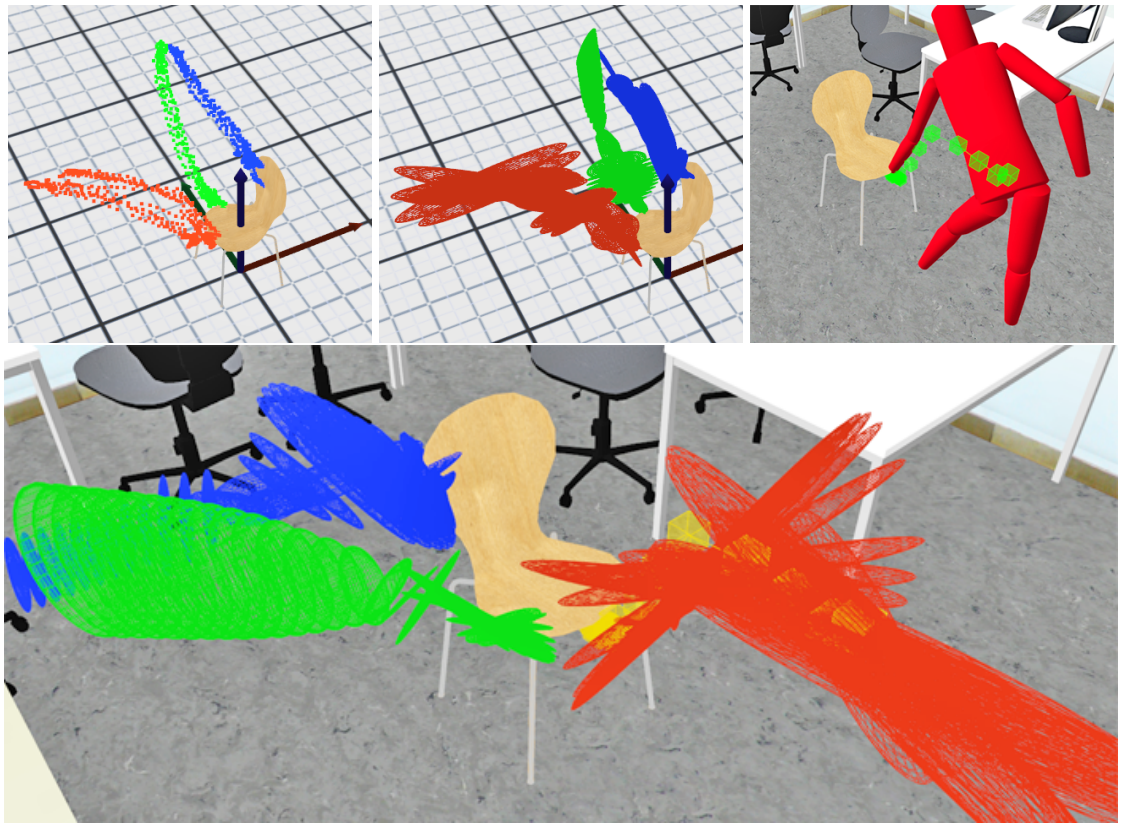


Figure 4.10.: An example of manipulation action mapping. Top left: strategy demonstration data, center top: strategy GMR, top right: mission demonstration data; Bottom: Matching of strategy (red) and mission (yellow) demonstration data. - [125]

Based on the GMR, as shown in Figure 4.10, representation of the manipulation strategy Φ_j, a distance metric between the generalized strategy demonstration trajectories and the mission demonstration segment trajectory $\|\Phi_j, X_m^{tcp,act}\|_{act}$ can be computed as discussed next.

Finally, for a set of manipulation strategies $mst_j \in MST$, the human-activity label in a given demonstration is mapped to the action symbol corresponding to the strategy $act(j)$ with the smallest distance and thus highest similarity: $act_{segment} = act(\,argmin_j(\|\Phi_j, X_m^{tcp,act}\|_{act})\,)$.

4.3.2. Manipulation Action Distance Metric

After computation of Φ_j, there exists a mean μ_t and covariance Σ_t in cartesian (x,y,z) corresponding to each normalized data point x_t in trajectories $X_m^{tcp,act}$, $X_{j,i}^{tcp}$. The distance of a point x to a GMR representative (μ_t, Σ_t) can be computed using the Mahalanobis distance [93]:

$$\|\phi_{j,t}, x\| = \sqrt{(\vec{x} - \vec{\mu}_t)^T * (\Sigma_t)^{-1} * (\vec{x} - \vec{\mu}_t)} \qquad [4.19]$$

Based on that point distance, the weighted sum of distances between corresponding data points $x_t \in X_m^{tcp,act}$ and $\phi_{j,t} \in \Phi_j$ can be computed with a weight ω_t assigned to each pair t:

$$\|\Phi_j, \vec{\omega}, X_m^{tcp,act}\|_{act} = \frac{\sum_{t=1}^{n} \omega_t \sqrt{(\vec{x}_t^m - \vec{\mu}_t^{\Phi_j})^T * (\Sigma_t^{\Phi_j})^{-1} * (\vec{x}_t^m - \vec{\mu}_t^{\Phi_j})}}{\sum_{t'=1}^{n} \omega_{t'}} \quad [4.20]$$

The resulting distance metric $\|\|_{act}$ between a mission-demonstration manipulation action trajectory $X_m^{tcp,act}$ and a correspondingly normalized GMM/GMR-generalized manipulation strategy Φ_j is defined as the $\vec{\omega}$-weighted sum of distances between corresponding point pairs.

Without a weight vector $\vec{\omega}$ or with equal weights $\forall i, j : \omega_i = \omega_j$, all parts of the trajectory have the same influence on the similarity metric. However, such a measure is insufficient for capturing peculiarities of object manipulation motions. Manipulation strategy trajectory portions with little covariance, indicated by small determinant $det(\Sigma_t)$, reflect highly constraint motions. Thus, the type of motion in that section is important. Furthermore, trajectory shapes are more relevant close to the object with small $\|\vec{x}_t^m - obj.pose(t)\|$. Combining both aspects by means of preference weights α_c and α_o gives:

$$\omega_t = \frac{\alpha_c}{det(\Sigma_t)} + \frac{\alpha_o}{\|\vec{x}_t^m - obj.pose(t)\|} \quad [4.21]$$

4.3.3. Conclusions and Covered Action Concepts

Computing a GMR from manipulation strategy demonstration trajectories basically leads to a probabilistic imitation-learning representation as discussed in Section 2.4.1. Such a representation is less powerful than a manipulation strategy constraint graph and cannot reflect complex manipulation motions in scenes with multiple objects within limited space in the same, flexible manner. However, coarse-grained human activity tracking and classification is even more limited, thus such a simplified skill representation does not lead to disadvantages when computing trajectory similarities. Because of the limitations of coarse-grained full body activity classification, mission demonstration scenes and manipulation strategy demonstration setups have to be more similar than demonstration scenes and execution scenes.

Full flexibility of the manipulation strategy representation can then be exploited during action execution where the constraint graph instead of the GMR is applied. Yet, limitations of coarse-grained tracking and activity classification prevent differentiation between dexterous manipulation actions which vary only in fine-grained finger movement. While the presented concept might scale onto distance metrics between all corresponding fingertips instead of just the hand base (TCP), evaluation as discussed in Section 5.4 has only considered actions distinct in more coarse-grained motion patterns. Software implementation specific further details of this process stage are outlined in [42].

As in state-description generation, perspective shift takes place here, with the robot subsequently reasoning about sequences of actions as being performed by itself using its own skill alphabet, represented by manipulation strategies.

4.4. Generating Abstract Sequences by Segmentation of Demonstrations

When mappings of situations onto feature states and human actions onto executable robot skills have been computed, individual demonstration traces $Obs_j \in Demo$ can be segmented into discrete, abstract event sequences $(\ldots, s_t, a_t, s_{t+1}, a_{t+1}, \ldots)$. These sequences make distinctions between courses of events only where relevant. Slightly differing traces that lead to the same sequence, with all differences subsumed within the same states and actions, can be considered equivalent on the abstract level. Perspective shift allows the robot to reason about sequences in a manner as if it executes them itself. Thus, demonstration or execution-time courses of events can be handled equally.

Given a single demonstration trace $Obs_j : obs(t) = (E_s(t), G_a(t))$, states are segmented first, based on recorded situations, for instance in the evaluation setup:

$$E_s(t) = (RR.pose(t), HR.pose(t), HR.act(t), HR.utt(t), Objs.pose(t)) \quad [4.22]$$

Feature mappings $f^{Demo}_{feat_j}$ computed on the whole set of traces are applied to map situations onto feature states $c^j_z = f^{Demo}_{feat_j}(E_s(t))$, leading to a feature-discretized model:

$$f_{g1}(E_s(t)) \rightarrow \{c^1_{z_1}(t), \ldots, c^{|feat|_k}_{z_k}(t)\} \quad [4.23]$$

as defined in Section 3.2. Based on the feature-discretized model, a feature-state mapping for any category combination in *Demo* can be applied:

$$\forall y_1, \ldots, \forall y_k, f^{Demo}_{g2} : (c_{y_1} \in feat_1, \ldots, c_{y_k} \in feat_k) \rightarrow s_i \quad [4.24]$$

Subsequently, a trace with data points t is segmented [132]:

$$s_t = \phi(obs(t)), \phi : f^{Demo}_{g2}(f^{Demo}_{g1}(E_s) \quad [4.25]$$

$$Q'_s(t') = \begin{cases} \emptyset, & \phi(obs_{t-1}) = \phi(obs_t) \\ s_{t'}, & \phi(obs_{t-1}) \neq \phi(obs_t),\ s_{t'} = \phi(obs_t) \end{cases} \quad [4.26]$$

By discarding $\emptyset$ in Q'_s, for instance $Q'_s = (\emptyset, \emptyset, s_3, \emptyset, s_5, \ldots)$, a discrete, abstract time sequence of states $t' = x_i$ is derived $Q_s = (s_{x_1}, \ldots, s_{x_n}) = (s_3, s_5, \ldots)$. In real demonstration setups with noisy observation, temporal inertia filtering is necessary to avoid state oscillations at situations

E_s near feature state separators. Therefore, a state $s_{t'}$ is discarded if the sum of durations of corresponding trace data points is below a certain temporal threshold ε_t:

$$\forall t' \{t_{t'} \in t | s_{t'} = \phi(obs(t_{t'}))\} : Q_s \leftarrow \begin{cases} \emptyset, & \sum_{t_{t'}} |t_{t'}| < \varepsilon_t \\ s_{t'}, & else \end{cases} \quad [4.27]$$

Typical thresholds are discussed in Section 5.5. In contrast to the recorded trace, such a resulting sequence does not contain information about state duration and hence is void of temporal aspects beyond the sequence.

Abstract actions are determined next, analyzing $G_a(t)$ during state transitions, for example in the evaluation setup:

$$G_a(t) = (\delta(RR.pose(t)), RR.act(t), RR.utt(t)) \quad [4.28]$$

Each action aspect that is active during a state $s_{t'}$ contributes to the transition towards $s_{t'+1}$. Abstract actions combining several skill domains in a single subtask can be covered by Flexible Programs (FPs) as described in Section 3.4. Hence, skill domain specific abstract actions can be added to a subtask-action symbol $a_{t'}^{subtask}$, which represents all skill-domain-specific abstract actions that are executed concurrently by the FP.

The following action skill domains are considered in the evaluation setup: mobility, utterances, gestures and object manipulation. Each transition $s_{t'}, s_{t'+1}$ is checked accordingly. If $c_i \in s_{t'} : \textit{self-pose} \neq c_j \in s_{t'+1} : \textit{self-pose}$ a mobility action "Goto c_j" is added: $a_{t'}^{subtask} \leftarrow a_{goto-c_j}$. Precise target position and path planning for such an action is performed by the component described in Section 4.11. In case an utterance $utter_j$ was expressed during $s_{t'}$, a robot speech action is added: $a_{t'}^{subtask} \leftarrow a_{utter_j}$. Gestures and object manipulation are mutually exclusive and only one class can be active at a given point $RR.act(t)$. A gesture action is added if a known gesture skill $a_{gesture_j}$ is detected in the trace: $a_{t'}^{subtask} \leftarrow a_{gesture_j}$ [131]. In case a mapped manipulation strategy mst_j was performed while $s_{t'}$ occurred, a manipulation action $a_{t'}^{subtask} \leftarrow a_{mst_j}$ is added.

Finally, segmentation leads to an abstract sequence of states and actions with discrete time t':

$$Q_{Obs_j} = (\ldots, s_{t'}, a_{t'}, s_{t'+1}, a_{t'+1}, \ldots) \quad [4.29]$$

4.5. Smoothing Imperfect Observation Sequences by Means of Causal Models

In many demonstration observation setups, the assumption of fully observable recording is sufficient because of several particular recording setup properties:

- A scene setup can be tailored to be observation-friendly: for instance, the robot stands still during observation, which leads to better self-localization and object localization.
- Additional sensors can be used: for instance, headsets for speech recording.
- Pronounced motions can be performed by role demonstrators.
- A fixed set of known objects and obstacles can be guaranteed.
- Inaccurately recorded demonstration traces can be discarded afterwards.

However, in some setups it is not possible to guarantee sufficient observation accuracy. In such cases, a recorded and segmented sequence has to be assumed to be a sequence of potentially inaccurate observations and not true, intrinsic states. Hidden Markov models (HMMs) [115] are a technique modeling such dynamics and closely related to POMDPs. A transition model $T_{HMM}(s,s')$ models the likelihood of a state s' following s while an observation model $O_{HMM}(m,s)$ models the likelihood of the true state s generating an observation m.

$O_{meta}(m,s) := O_{HMM}(m,s)$ is very similar to the execution-time robot POMDP observation model O as is has to model the same robot perception limitations and error likelihoods. Therefore, $O_{meta}(m,s)$ can be generated like O with techniques described in Section 4.10.

On the other hand, within the scope of demonstration recording, the transition model $T_{HMM}(s,s') = T_{meta}(s,s')$ is composed of likelihoods indicating fundamental causality in the world: states that are impossible to follow a certain other state directly, states which are unlikely to do so and states which may likely follow certain other states. Thus, the meta transition model $T_{meta}(s,s')$ describes fundamentally impossible or unlikely transitions in an environment setting, independent from any actual mission courses of events.

Hence, $T_{meta}(s,s')$ has to be generated by dedicated methods. Yet, a conservative approach can be taken by initially weighting all transition probabilities equally: $p(s'|s) = \frac{1}{|s|}$. Subsequently, impossible transitions are set to $p(s'|s) = 0$ and unlikely transitions get reduced probability. Finally, the rest of the row is normalized to 1. Such a conservative approach guarantees that no real demonstrated transition is discarded, thus having a preference for false positives over false negatives. More precisely, data for $T_{meta}(s,s')$ can be collected from PbD knowledge collected over the course of many mission as discussed in Section 4.10, geometric analysis discussed in Section 4.11 or learning by trials discussed in Section 4.12. However, there is the limitation that background knowledge can only be related to state classes and not specific states, as state grounding is generated for each mission individually. To fully cover these state-to-state class relationships in the scope of a meta transition model when using automatic state mapping generation, reasoning over state similarities is needed which is beyond the scope of this thesis.

Given a fully defined HMM modeling the demonstation observation setting with $T_{meta}(s,s')$, $O_{meta}(m,s)$, HMM smoothing can be applied to retrieve the most likely sequences of true, intrinsic demonstration states. The inputs are an observed and segmented observation sequence $Q^{measured}_{Obs_j}$, T_{meta} and O_{meta}. Actions are ignored at first and the Viterbi Algorithm [48] computes the most likely hidden sequence $Q^{hidden}_{Obs_j}$. Finally, actions are tailored to match the new state sequence. Software implementation specific processing stage details and alternative methods of meta model knowledge input can be found in [74].

4.6. Analysis of Multiple Sequences to Generate Preliminary POMDP Models

By definition, a mission may encompass multiple courses of events $(\ldots, s_{t'}, a_{t'}, s_{t'+1}, a_{t'+1}, \ldots)$ that represent an agent successfully executing a desired role. Different courses of events emanating from identical initial abstract configurations (start states) require points of event branching, i.e. alternative potential future courses of events. As addressed in Section 2.2.2, only with branching of potential courses of events, true decision making is possible.

Within the POMDP framework, two points of branching exist:

1. Different actions a that can be performed in a state s.
2. Different stochastic effects of an action a performed in state s.

Within the scope of PbD, with no additional model knowledge considered, typically only a single course of events could be learned from a single demonstration sequence. The notable exception are sequences with loops, where certain states s or even state-action pairs (s,a) are revisited and followed by differing courses of events:

$$(\ldots, s_{t_1}, a_{t_1}, \ldots, s_{t_2}, a_{t_2}, \ldots),\ s_{t_1} = s_{t_2} \wedge a_{t_1} \neq a_{t_2} \quad [4.30]$$

$$(\ldots, s_{t_1}, a_{t_1}, s_{t_2}, \ldots, s_{t_3}, a_{t_3}, s_{t_4}, \ldots),\ s_{t_1} = s_{t_3} \wedge a_{t_1} = a_{t_3} \wedge s_{t_2} \neq s_{t_4} \quad [4.31]$$

However, even then little branching information in relation to sequence length can be acquired. Therefore, multiple sequences are considered:

$$\Xi_{Demo} := \{Q_{Obs_1}, \ldots, Q_{Obs_n}\} \quad [4.32]$$

Such multiple sequences may be acquired incrementally while creating and executing a complete mission model in between. Thus, incremental life-long mission learning is automatically covered by the presented concept. In fact, the automatic model space exploration and interactive request approach discussed in Section 4.7 and 4.8 heavily rely on that characteristic.

Beyond typical usage, single demonstration *one-shot learning* may be necessary under some circumstances and is suitable in combination with autonomous model refinement techniques described in Section 4.10 , 4.11 and 4.12, which add further branching options to the model.

To generate a preliminary mission model, including all aspects that can be learned from demonstrations, Ξ_{Demo} is analyzed. First, states are accounted for, leading to a preliminary demonstration state space $S_D = s|\exists Q \in \Xi_{Demo} : s \in Q$. A default error state $s_e \rightarrow S_D$ is added to the state space, being a placeholder for a more diverse set of error states later added in model refinement stages. Next, all actions occurring in the demonstration set compose a preliminary demonstration action space $A_D = a|\exists Q \in \Xi_{Demo} : a \in Q$. Exploiting the Markov property, state-action pairs (s_t, a_t) and resulting effects are s_{t+1} are analyzed as isolated transitions. Basically, the robot is then able to rearrange chains of events by planning on atomic transitions within the POMDP framework. Hence, all unique transitions in the demonstration sequence are counted, leading to a transition frequency model TF_D that is initialized with 0:

$$\forall s_t \in \Xi_{Demo} : T_D^F(s_t, a_t, s_{t+1}) = T_D^F(s_t, a_t, s_{t+1}) + 1 \qquad [4.33]$$

Because a single occurrence of a state s_i leads to its inclusion in the transition model but then is just defined in one pair (s_i, a_k), all entries in transition frequency rows $(s_i, a \neq a_k)$ are zero, which can be considered as undefined. This would lead to a uniform action effect probability distribution in the following normalizing step. That is the worst possible estimate of unknown action effects. Instead, either the assumption that the result of such an observed pair (s_i, a_k) is a default error state $T_D^F(s_i, a \neq a_k, s' = s_e) = 1$ or that there is no change at all $T_D^F(s_i, a \neq a_k, s' = s_i) = 1$ are better initial default estimates for any undefined row. The error state assumption is safer but the more aggressive static assumption is more versatile in combination with generalized transitions. If not stated otherwise, static is the default in experiments, though the system can be configured to take either approach. More realistic estimates than these defaults for action effect prior probabilities of unobserved pairs (s_i, a_k) are derived by a generalization discussed in Section 4.7 and a subsequent refinement presented in Sections 4.10, 4.11 and 4.12.

These frequencies in mission courses of events reflect primary stochastic effects as defined in Section 3.5. Consequently, a preliminary demonstration transition model T_D can be computed from T_D^F by computing s'-row probabilities from frequencies, which means normalizing over stochastic outcomes of (s, a) pairs:

$$\forall (s, a, s') : T_D(s, a, s') = \frac{T_D^F(s, a, s')}{\sum_{s'_i} T_D^F(s, a, s'_i)} \qquad [4.34]$$

Finally, the preliminary reward model R_D has to account for goals. As action costs are known only from background knowledge and goal values computed from costs of actions leading to goals, goals can only be flagged at this point. Accordingly, R_D is initialized with 0 and for each final (s,a) pair of a sequence, a goal is flagged: $(s_r,a_k)_{t_n} \in Q_{Obs_j}; R_D(s_r,a_k) = 1$. This method implies that goal selection can be focussed by demonstration sequence choice.

Such preliminary models generated from abstract demonstration sequences represent an MDP, since partially observable aspects in O are robot-specific and cannot be learned from a human demonstration. Furthermore, secondary stochastic effects as defined in Section 3.5 are missing. Hence this model is considered as preliminary. How to add those aspects is addressed in Section 4.10. Further details and examples are discussed in [132].

4.7. Confidence-based Generalization of Action Effects

Up to this point, human teachers provide demonstration sequences with courses of events and relative frequencies thereof chosen freely based on their understanding of the mission. This leads to a conflict of two principles:

1. Human teachers tend to perform as few demonstrations as possible, and they are, potentially just subconsciously, biased about the structure of a mission.

2. The more valid courses of events are represented in the transition model: the better those values reflect real-world dynamics, the better the performance of the robot.

To attenuate this conflict, the robot has to extend its role beyond a passively watching student. Instead, it has to analyze transition models acquired from demonstrations [117], [134] and estimate where demonstrations are likely missing in transition model space as described in the following. Based on these estimates, it can then judge the potential impact of lacking effect knowledge on its decision-making options and request further demonstrations actively, as discussed subsequently in Section 4.8.

4.7.1. Transition Generalization

At this stage of the process, all learned transitions are primary action effects as classified in Section 3.5.1. Two main types of transition entries $T(v) := T(s_i,a_k,s'_j)$ can be distinguished: zero $T_0(v) : p(s'_j|s_i,a_k) = 0$ and nonzero $T_\exists(v) : p(s'_j|s_i,a_k) > 0$. Zero transitions $T_0(v) \in T_D$ have never occurred in a demonstration at this stage. It has to be determined if the transition does not occur in a mission or if it was just never demonstrated. Furthermore, in the latter

case it has to be determined if the transition may be part of a promising course of events and is therefore important to be considered by decision making.

Typically, a transition model acquired from demonstrations, before further refinement computations, is extremely sparse with $|T_0| >> |T_\exists|$ because $|T_0| = |T| - |T_\exists|$ and $|T| = |S| \times |S| \times |A|, |T_\exists| < |\Xi_{Demo}|, |\Xi_{Demo}| << |T|$. At most one new $T_\exists$ is added for each demonstrated pair (s,a). The state representation based on feature states $s_i := (c_{x_1} \in F_1, \ldots, c_{x_n} \in F_n)$ applied without the *combine* operation as discussed in Section 3.6, leads to potentially many states $s_{new'}$ being added to S_D and thus T_D when a state s_{new} with a feature state $c^k_{x_k} \in s_{new}, c^k_{x_k} \in F_k$, not encountered previously is accounted for in an observation sequence:

$$c^k_{x_k} \notin \Xi_{Demo}([0:t-1]), c^k_{x_k} \in \Xi_{Demo}(t) \Rightarrow$$
$$\forall F_i \text{ with } i \neq k, \forall x_i \in F_i : f_{g2}(c^1_{x_1}, \ldots, c^k_{x_k}, \ldots, c^n_{x_n}) = s^{\vec{x}_i}_{new'} \rightarrow S_D \qquad [4.35]$$

All these additionally added states $s^{\vec{x}_i}_{new'}$ initially result in zero transition rows. Thus T and T_0 sizes grow much faster than $T_\exists$. Because of this sparsity, selection of a subset of candidates $T_0^{candidates} \subset T_0$ can be interpreted as an exploration process in transition model space, emanating from $T_\exists$. In contrast, in a hypothetical non-sparse model with $T_0 << T_\exists$, an analysis of the "last gaps" would not make exploration necessary as they can all be tested.

To enable exploration in transition model space, an expressive norm $\|\|_T$ needs to define a space on the model. With such a norm, the similarity between two transitions $\|T(v_1), T(v_2)\|_T$ can be determined which in turn allows for exploration and assessment. The aim is to determine which $T_0(v_g)$ most likely are similar to encountered transitions and thus may have been omitted, although they are in fact $T_\exists$, as illustrated in Figure 4.11:

$$\{T_0^{candidate}(v_g) \in T_0 | \; \exists T_\exists(v_o) : \|T_\exists(v_o), T_0^{candidate}(v_g)\|_T < \varepsilon\} \qquad [4.36]$$

Basic dimensions of v are S, A and S', therefore the similarity norm has to be defined on these attributes. In turn, S, S' are based on features with $S = F_1 \times \ldots \times F_n$. Distance metrics can be defined within features $\|c_{x_i}, c_{x_j}\|_F$, taking into account the characteristics of a domain, such as similarities of human utterances or distances between pose regions. Suitable metrics are described in Section 3.3. Subsequently, state norms can be based on individual feature norms:

$$\|s_i, s_j\|_S := \sqrt{(\|c^1_{x^i_1} \in s_i, c^1_{x^j_1} \in s_j\|_{F_1})^2 + \ldots + (\|c^n_{x^i_n} \in s_i, c^n_{x^j_n} \in s_j\|_{F_n})^2} \qquad [4.37]$$

An alternative state norm is the minimum feature state distance:

$$\|s_i, s_j\|_{F-min} := argmin_f(\|c^f_{x^i_f} \in s_i, c^f_{x^j_f} \in s_j\|_{F_f}) \qquad [4.38]$$

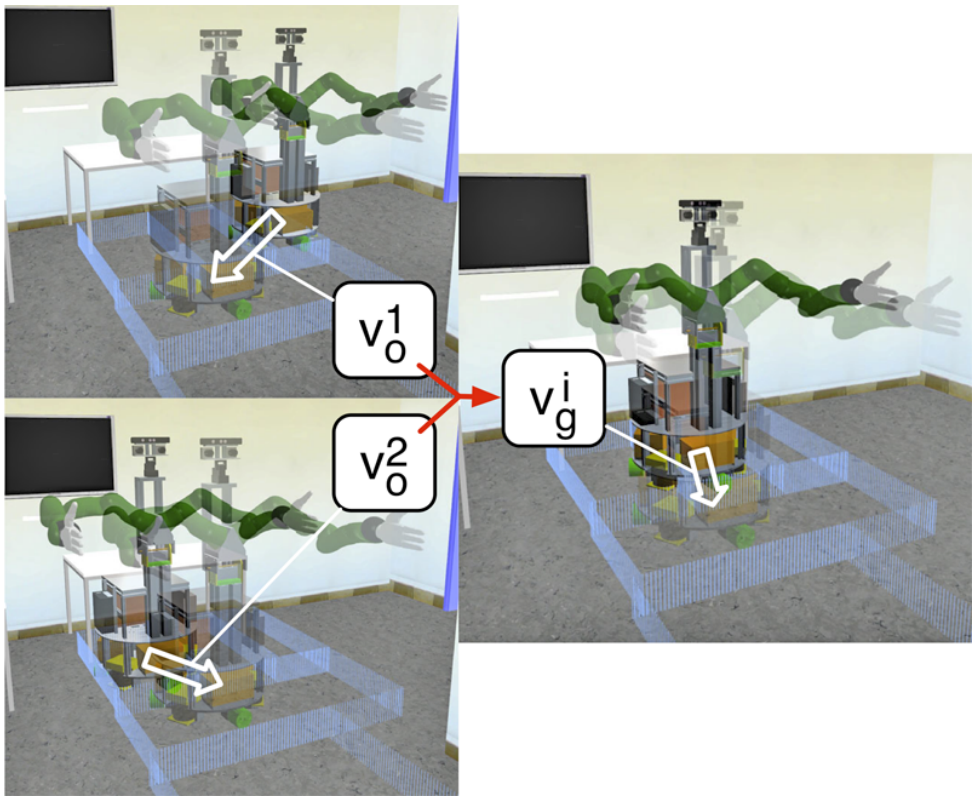


Figure 4.11.: *Illustrative example*: Given two observed transitions v_o^1, v_o^2 with $s'^1 = s'^2$ (left) and another feature state close by, an unobserved transition v_g^i with $s'^i = s'^1$ may be valid. - [125]

Action similarities can be defined based on skills. Finally, a set of transitions encountered in demonstrations $\{T_\exists(v_{o_1}), \ldots, T_\exists(v_{o_n})\}$ can be *generalized* by finding a set of similar not encountered transitions $\{T_0^{candidate}(v_{g_1}), \ldots, T_0^{candidate}(v_{g_m})\}$ for which an initial assumption is made that they are related to their *baseline transitions* $T_\exists(v_o)$.

Guided exploration is achieved by search for candidates within S, S' and A using three complementary approaches:

1. Generalizing transition origin s keeps a, s' fixed with baseline $v_{o_x} = (s_{i_x} \in S, a_k, s'_j)$ and candidate $v_g \in V^{candidates} = (s_{g_y} \notin \{s_{i_1}, \ldots, s_{i_n}\}, a_k, s'_j)$. This mode of generalization takes into account that many actions converge to the same effect given a related set of situations they are executed in, for example, goto actions or question utterances.

2. Generalizing transition effect s' keeps s, a fixed with baseline $v_{o_x} = (s_i, a_k, s'_{j_x} \in S')$ and candidate $v_g \in V^{candidates} = (s_i, a_k, s'_{g_y} \notin \{s'_{j_1}, \ldots, s'_{j_n}\})$. By these means, given a set of observed stochastic effects resulting from a pair (s_i, a_k), it is assumed that effects closely related to a set of observed ones are more likely to be omitted in demonstrations.

3. Generalizing transition action a keeps s, s' fixed with baseline $v_{o_x} = (s_i, a_{k_x} \in A, s'_j)$ and candidate $v_g \in V^{candidates} = (s_i, a_{g_y} \notin \{a_{k_1}, \ldots, a_{k_n}\}, s'_j)$. In this mode, action invariant transitions among sets of similar origins and similar effects are determined, typically encountered when a dynamic environment effect is not directly related to an action but occurs conditionally independent to the action in certain related sets of states.

To describe and access candidate sets of transitions v_g in a compact way, a *transition mask* is defined in the form of a selection filter function κ. It represents sets of related transitions V_{mask} with wildcards $*$ denoting a whole subspace of a feature or actions that are selected:

$$A^* = A \cup \{*\}, A^*(a_i^*) = \begin{cases} A & \text{if } a_i^* = * \\ a_i & \text{else} \end{cases} \quad [4.39]$$

$$F_f^* = F_f \cup \{*\}, F_f^*(c_i^*) = \begin{cases} F_f & \text{if } c_i^* = * \\ c_i & \text{else} \end{cases} \quad [4.40]$$

$$S^* = F_1^* \times \ldots \times F_n^* \quad [4.41]$$

$$V_{mask} = \kappa(s^*, a^*, s'^*) = \{s^* \in S^*, a^* \in A^*, s'^* \in S^*\} \quad [4.42]$$

4.7.2. Concept Learning of Generalized Transition Hypotheses

A first stage of rough exploration determines sets of candidate transitions by concept learning. Hypotheses represented by a mask V_{mask} are generated by generalization from a set of baseline transitions V_o. A set of hypotheses $V_g = \kappa_g$, can be acquired utilizing the following elements:

1. A set of input transitions V_o generating the hypothesis.

2. A transition mask $\kappa_{scope}(s^*, a^*, s'^*)$ defining the scope which covers the elements over which may be generalized.

3. A transition mask $\kappa_g(s^*, a^*, s'^*)$ containing the resulting generalized set of transitions.

In generalization modes 1) and 2), a scope mask κ_{scope} is generated by analyzing which features are affected by actions in the set V_o. Here, actions a are not generalized and can thus be used to derive κ_{scope}. For each transition v_o, it is checked which feature states are changed by the action and wildcards are inserted accordingly:

$$\forall v_o = (s_i, a_k, s'_j), \forall F_f \in S : \kappa_{scope}(F_f) \leftarrow \begin{cases} * & if\, c_{x_f} \in s_i \neq c_{y_f} \in s'_j \\ c_{x_f = y_f} & else \end{cases} \quad [4.43]$$

In mode 3), just a wildcard over all actions is applied to generate κ_{scope}. In the process, κ_g is incrementally generalized to cover all inputs $v_o \in V_o$: at every step it describes the most specific mask that covers all inputs in the compact form defined above. This is a form of concept learning from specific to general: each input transition v_o is a positive example of the hypothesis κ_g. At each incremental step p, a unique input transition v_o^p is added:

$$\kappa_g^p = \begin{cases} \kappa_g^{p-1} & if v_o^p \in \kappa_g^{p-1} \\ gen(\kappa_g^{p-1}, v_o^p, \kappa_{scope}) & else \end{cases} \quad [4.44]$$

An operation gen adds wildcards, where the new input v_o^p is both not covered by the previous mask κ_g^{p-1} and generalization is allowed by mask κ_{scope}, for instance for mode 1):

$$gen(\kappa_g, v_o = (s_o, a_o, s'_o), \kappa_{scope}) :$$
$$\forall F_f \in S : \kappa_g^{result}(s^*) \leftarrow \begin{cases} * & if (c_{x_f} \in s_o \not\subseteq c_{y_f} \in s^*_{\kappa_g}) \wedge (c_{z_f} \in s^*_{\kappa_{scope}} = *) \\ c_{x_f = y_f} & else \end{cases}$$
$$\text{with } c_{x_f} \in s \subseteq c_{y_f} \in s^* := (c_{x_f} = c_{y_f}) \vee (c_{y_f} = *) \quad [4.45]$$

As a result, with $p_1 < p_2$, $\kappa_g^{p_1}$ is always equally or more specific than $\kappa_g^{p_2}$.

4.7.3. Generalization Confidence Computation

Following rough exploration based on concept learning, more fine-grained relationship and relevance confidence computation takes place. A potentially large set of candidates generated by concept learning is differentiated and reduced. It is achieved by computing a *generalization confidence* value $gc(v_g)$, reflecting both likelihood that a transition candidate is indeed nonzero as well as a preliminary relevance. Differentiation incorporates three different aspects:

1. The distance in model space $\|T(v_o), T(v_g)\|_T$, as defined in Section 4.7.1, to the closest baseline transition v_o determines basic confidence by similarity $sim(v_g)$.
2. A *non-observation bias* $nob(v_g)$ influences confidence.
3. A *certainty bias* $ceb(v_g)$ influences confidence.

All three aspects are defined for v_g in relation to observed transitions V_o. For each candidate $v_g \in V_g$, the minimal feature norm to the nearest baseline transition $v_o \in V_o$ is determined:

$$\text{Mode 1): } sim'(v_g) = argmin_o(\|s_{v_o}, s_{v_g}\|_{F-min}), v_o \in V_o \quad [4.46]$$
$$\text{Mode 2): } sim'(v_g) = argmin_o(\|s'_{v_o}, s'_{v_g}\|_{F-min}), v_o \in V_o \quad [4.47]$$
$$\text{Mode 3): } sim'(v_g) = argmin_o(\|a_{v_o}, a_{v_g}\|_A), v_o \in V_o \quad [4.48]$$

From this minimum-distance value, a confidence value is computed using a negation function with a scale factor α_{gc}, such as Sugeno-Negation:

$$sim(v_g) = \frac{1 - sim'(v_g)}{1 + \alpha_{gc}\, sim'(v_g)}, \alpha_{gc} \in (-1, \infty) \quad [4.49]$$

The higher the number of observations of the most similar baseline transition, the higher the likelihood that a generalized transition v_g was not omitted accidentally. Hence, a higher baseline occurrence count *baseocc* nearby in model space decreases confidence of a non-observed generalized transition, introducing the *non-observation bias*, *nob* with scale factor β_{gc}:

$$v_o^{mindist,v_g} = min_o(\|s_{v_o}, s_{v_g}\|_{F-min}), v_o \in V_O \quad [4.50]$$

$$baseocc = |v_o^{mindist,v_g}|_{Demo} \quad [4.51]$$

$$nob(v_g) = (1 - \beta_{gc})^{baseocc}, \beta_{gc} \in [0,1] \quad [4.52]$$

While stochastic action effects have to be considered in a model and close approximation of real-world dynamics is beneficial, action effect probabilities tending towards uniform distributions are not preferable. Yet, such behavior occurs when for some observed transitions $v_{o_x} = (s_i, a_k, s'_{j_1}), v_{o_y} = (s_i, a_k, s'_{j_2})$, a generalized transition hypothesis $v_g = (s_i, a_k, s'_{j_3}), s'_{j_1} \neq s'_{j_2} \neq s'_{j_3}$ is generated. Such a transition hypothesis pushes the effect probabilities of (s_i, a_k) towards a more uniform distribution and reduces the share of observed transitions. This has somewhat detrimental effects because:

- Probabilities of observed transition are reduced, yet those are the preferred estimate.
- Real-world effects do not tend towards a uniform distribution - typical models in the literature for robotic domains as addressed in Section 2.2 are all sparse with $|T_\exists(s_i, a_k, s'_x)| << |T_0(s_i, a_k, s'_y)|$ in a transition row (s_i, a_k).
- Models tending towards uniform distributions lead to policies with short sequences and unstable, oscillating courses of events which is not a property of real-world missions.

To allow for the introduction of some new stochastic effects, but also limiting it, a progressively resisting factor, *certainty bias* $ceb(v_g)$, is introduced. It balances exploration and exploitation when computing confidences of new transition hypotheses:

$$V_r = \{v_r |\ v_r = T(s_i, a_k, s'_l), s'_l \neq s'_j\}, v_g = T(s_i, a_k, s'_j) \quad [4.53]$$

$$n = \sum_{v_r} |v_r|_{Demo} \quad [4.54]$$

$$ceb(v_g) = \begin{cases} 0 & \text{if } n = 0 \\ \sqrt[n]{\gamma_{gc}} & \text{else}, \gamma_{gc} \in [0,1] \end{cases} \quad [4.55]$$

Parameters α_{gc}, β_{gc} and γ_{gc} reflect internal weights of different aspects on generalization confidence and have to be determined empirically for each setting. Finally, generalization confidence can be computed:

$$gc(v_g) = sim(v_g) * nob(v_g) * ceb(v_g),\ gc(v_g) \in [0,1] \quad [4.56]$$

At this point, hypotheses below a certain confidence $gc(v_g) < \varepsilon_{gc}$ can be discarded, reducing the candidate set.

Based on the confidence value, one approach is to compute a transition probability and add it to the model directly. Another approach is using it for relevance computation in the verification process discussed in Section 4.8 and then discarding it, while it is implicitly replaced by values derived from resulting further demonstrations.

Confidences above a certain threshold ε indicate that for a given transition v_g, a nonzero transition probability is more likely than a zero one. At this point it has to be remembered that each transition probability is only *a Bayesian estimate of real-world stochasticity*. Therefore, an initial zero value, following from no observation of v_o, has no absolute authority in the face of a most likely incomplete demonstration space Ξ_{Demo}. A transition model serves to enable best available prior estimates of potential future events and not as an asymptotically precise frequentist approximation of a controlled experiment. To compute a new transition probability, the assumed correlation confidence with a similar, observed baseline transition $v_o^{mindist,v_g}$ is combined with the frequency of that transition:

$$p_T(v_g) = gc(v_g) * p_T(v_o^{mindist,v_g}) \quad [4.57]$$

Because $gc(v_g) \leq 1 \Rightarrow p_T(v_g) \leq p_T(v_o^{mindist,v_g})$ which basically means that the robot will regard the estimated effect as less likely as the related, observed one. Accordingly, planning will only take this less certainly known effect into account when long-term expected utility values are impacted more by potential courses of events unfolding after that transition. This is in line with being more conservative about effects for which less information exists.

Normalization of rows $T(s,a)$ has to occur after all generalized transition probabilities have been computed, to avoid distortions by normalizing multiple times over some values. Default assumption action effects $p(s'|s,a)$, both static $(s' = s)$ or for the default error state of S_D, $(s' = s_e)$, are changed considerably by generalized transitions $p(s'_g \neq s, s_e|s,a) > 0$: the robot may acquire significant new options regarding courses of events.

There are several strategies to continue from this point:

1. Aggressive: the model is complemented with all generalized, non-observed transition probability estimates directly without further verification. The disadvantages of such lacking verification are discussed in Section 4.8.

2. Conservative: generalization hypotheses are ranked for relevance, relevant transitions are verified as discussed in Section 4.8 and a new model is generated based on these demonstrations - no generalized transition probability estimate is directly adopted into the model used for policy computation. Verifying all candidates – no matter their relevance – takes a lot of interaction effort for the human teacher.

3. Balanced: transitions scheduled for verification are not adopted into the final model and several iterations of further requested demonstrations are integrated until a remaining set of generalized transition estimates below a certain relevance threshold remains which are added without further verification. The balanced approach combines the advantage of interactive verification of doubtful, but critical generalization candidates while avoiding extensive verification of less critical candidates.

If not stated otherwise, the balanced approach is used in experiments of the presented system discussed in Section 4.8.

4.7.4. Limitations

In practice, the quality of generalization confidences depends on good model space similarity metrics $\|\|\|_T$, but only a limited number of metrics were investigated within the scope of this work. While the reuse of observation model metrics is efficient, more specific metrics could be investigated.

There is limited real-world stochasticity: the real world relevant in robot missions on an abstract level, while not fully deterministic, is not symmetric concerning occurrence of more deterministic vs. more uniform distributions in transitions. As argued previously, real-world robot missions have sparse transition models, even when most carefully engineered to reflect all potential real-world dynamics. While the certainty bias takes that into account, the lack of symmetry makes aggressive exploration estimates more error-prone. Retrieving a well suited bias (asymmetry) function for a domain like general service robot missions would require most extensive experiments, beyond the scope of this thesis. Therefore, a more conservative approach relying heavily on interactive human verification has been taken as discussed next. It still relies on the generalization and confidence computation step which therefore is in no way redundant. Experiments regarding this process stage are discussed in Section 5.6. Further software imple-

mentation specific details and extensive evaluation of suitable parameter values, applicable for diverse types of missions are presented in [117].

4.8. Interactive Requests for Human Demonstrations to Verify Generalizations

A problem of products of confidence and baseline probability estimates $p_T(v_g) = gc(v_g) * p_T(v_o^{mindist,v_g})$, as defined in the previous Section, is that they open up more options regarding courses of events. This can make a policy unstable in terms of selected actions and ensuing courses of events. In an environment with symmetric stochasticity bias that is no problem, because the probability to miss a risk or opportunity with a false zero probability transition (false negative) and the probability to introduce an invalid option (false positive) balance out. As stochasticity is not symmetric, a verification mechanism for generalized transition hypotheses is necessary. Because observed transitions and corresponding generalizations at this stage consider only primary transitions, human demonstrations are a suitable context for verification.

Consequently, verification incorporates the human teacher closely in an interactive process after generalization relevance analysis and subsequent generation of demonstration requests to the teacher [81]. By these means, generalization and request generation together enable a robot to steer the demonstration set towards covering initially missing model aspects in the form of courses of events omitted, at first. Hence, PbD where the robot is passively watching demonstrations is enhanced to a mutually interactive learning process with human and robot, teacher and student working actively together.

4.8.1. Generalized Transition Relevance Analysis

Generalization may generate a large number of new nonzero transition hypotheses T_0. However, the number of requests for new human demonstrations should be minimized. Hence, generalized transition hypotheses have to be ranked for impact on decision making. For example, if the origin state s of a transition hypothesis v_g is never reached by demonstrated courses of events, this generalized transition is not relevant, even if its own probability is high. Similarly, if demonstrations do not contain courses of events continuing from the effect state, the transition is not interesting, leading only to a default error state or remaining static.

Such a ranking does not model any real-world dynamics precisely, but instead is a rough heuristic for the potential impact of a nonzero transition added to an existing model. By further demonstrations resulting from a selection of the transition hypotheses based on the ranking, a human teacher will then implicitly evaluate the transition probability. Therefore, the exact ranking value will not be adopted further into the model. Because of complexity issues dis-

cussed further below, in the following process transition hypothesis candidates v_g are added to the model, with their relevance computed individually. Consequently, tuples of additional transitions that have a relevant impact only when added together are not evaluated.

In case an observation model can be added at this point by the component discussed in Section 4.10, a POMDP policy $\pi_{POMDP}(b(S))$ can be computed: otherwise an MDP policy $\pi_{MDP}(s)$ is computed as an approximation, indicating underlying real-world dynamics. The reward model R has to be completed with costs added for all actions as discussed in Section 4.10, but no further states beyond the default error state are added. Then, a number of m simulated policy π_{*MDP} executions with n steps are performed with transitions and potentially observations sampled according to the model. Initial states s_1 in simulation runs are sampled from relative initial state frequencies in demonstration sequences $p(s = s_1 \in \Xi_{Demo})$. The result is a set of sequences $\Xi_{sim*MDP}$. In simulation, the true state of the world is always known to the simulation system and can thus be recorded for both MDP and POMDP. By these means, even high probability transitions with a high utility $U(s)$ will never be reached during execution in case they are never reached from mission-typical initial states $s_1 \in Q$ as indicated by demonstrations.

Based on the original model plus one transition hypothesis v_g, T_{D+v_g} and a corresponding policy π_{*MDP}, a set of simulation sequences $\Xi_{sim*MDP}$ can be retrieved. With those properties and a set of demonstration sequences Ξ_{Demo} on which the original model is based, a relevance measure $\rho^v(v_g), v_g = (s_i, a_k, s'_j)$ can be computed.

State relevance $\rho^s(s) \rightarrow [0,1]$, a measure for states s_i, s'_j is computed first . State relevance is in turn composed of a set of measures $\rho^s_e(s) \rightarrow [0,1]$.

1. Relative state occurrence frequency in demonstration sequences $Q_{Obs_j} \in \Xi_{Demo}$ accounts for states s_i being visited in the baseline model T_D:

$$\Xi_{s_i \in Q} := \{s_i \in Q_{Obs_j} | Q_{Obs_j} \in \Xi_{Demo}\} \quad [4.58]$$

$$\rho^s_1(s_i) = \rho^s_{s_i \in Q_{Obs}}(s_i) = \frac{|\Xi_{s_i \in Q}|}{|\Xi_{Demo}|} \quad [4.59]$$

   Based on demonstrations alone, it considers relevance of a state s_i, from the point of view when the transition v_g is *not* included in a model.

2. Relative state occurrence frequency in simulation sequences $Q_{sim*MDP_j} \in \Xi_{sim*MDP}$ accounts for states s_i that are visited in the model T_{D+v_g}:

$$Q_{sim*MDP_j} = (s_1 \sim p(s_1 \in \Xi_{Demo}), a_1 = \pi_{*MDP}(s_1), s_2 \sim p_T(s_1, a_1), \ldots, s_n) \quad [4.60]$$

$$\Xi_{sim*MDP} = \{Q_{sim*MDP_1}, \ldots, Q_{sim*MDP_m}\} \quad [4.61]$$

$$\Xi_{s_i \in Q_{sim*MDP}} := \{Q_{sim*MDP_j} | s_i \in Q_{sim*MDP_j}, Q_{sim*MDP_j} \in \Xi_{sim*MDP}\} \quad [4.62]$$

$$\rho_2^s(s_i) = \rho^s_{s_i \in Q_{sim*MDP}}(s_i) = \frac{|\Xi_{s_i \in Q_{sim*MDP}}|}{|\Xi_{sim*MDP}|} \quad [4.63]$$

This measure considers relevance of a state s_i in terms of occurrence frequency after adoption of v_g to the model. In contrast to the previous measure, action cost rewards are also implicitly considered here, leading to potentially stronger differing frequencies of courses of events compared to demonstrations. Number m and length n of simulation runs can be derived from demonstrations that indicate typical length and overall number to reflect of courses of events: $m = 2 * |\Xi_{Demo}|$, $n = 2 * avg|Q_{Obs}|$.

3. Based on T_{D+v_g}, the relative number of nonzero transitions, of which the state is an effect indicates its relevance in varying courses of events:

$$\rho_3^s(s_i) = \rho^s_{|T_\exists(*,*,s_i)|}(s_i) \frac{|\{(\bar{s}, a, s_i) | p_T(s_i | \bar{s}, a) > 0, \bar{s} \in S, a \in A\}|}{|S| * |A|} \quad [4.64]$$

This measure needs neither a simulation, nor computation of a policy.

4. The probability sum of transitions in T_{D+v_g} leading to a state s_i is a crude, local indicator of its relevance:

$$\rho_4^s(s_i) = \rho^s_{p_T^{relsum}(s_i)}(s_i) \frac{\sum_{\bar{s} \in S, a \in A} p_T(s_i | \bar{s}, a)}{|S| * |A|} \quad [4.65]$$

This measure also does not need simulation or computed policy.

5. Relative utility of a state s_i in a value function $U^{\pi_{*MDP}}$ indicates that it is on courses of events leading to goals, bases on high positive rewards propagated backwards by value iteration:

$$U(s) = U^{\pi_{MDP}}(s) \text{ or } U^{\pi_{POMDP}}(b(p(s) = 1.0)) \quad [4.66]$$

$$\rho_5^s(s_i) = \rho^s_{U_{rel}}(s_i) = \frac{U(s_i) - min_{s \in S}(U(s))}{max_{s \in S}(U(s)) - min_{s \in S}(U(s))} \quad [4.67]$$

To compute this measure, a policy with a value function – either MDP- or POMDP-based – is necessary, but no simulation runs have to be accumulated.

Finally, a state relevance can be computed by the $\vec{\omega}^s$ weighted sum:

$$\rho^s(s_i) = \frac{\sum_{e=1}^{5} \omega_e^s \rho_e^s(s_i)}{\sum_{e=1}^{5} \omega_e^s} \quad [4.68]$$

Apart from measure 1), the other relevance measures are partly redundant. However, they vary both in terms of quality and proportional amounts of additional knowledge as well as computation time required. For measures 2) - 5) the following order holds concerning quality and effort: $2 >> 5 >> 4 > 3$. Given enough simulation samples, simulated courses of events are the best approximation, since they take into account consideration how often a state will be visited and therefore its relevance. However, a policy, in turn requiring a fully defined reward model, is necessary as well as a large number of computationally expensive simulation runs. While the learning process is not time-critical as it takes place offline, depending on model size and complexity policy computation or simulation may still be too costly in time, as it has to be performed for each generalized transition hypothesis. Just computing a value function and estimating relevance from utility values as done by measure 5) is simpler. Even less demanding measures 3) and 4) are faster, not requiring a fully defined model and policy computation, but in turn only very crude estimates of state relevance. As a result, different circumstances require different weight vector $\vec{\omega}^s$ configurations in descending order of quality and computational effort:

1. With fully defined models O and R, POMDP policy computation and initial simulation beliefs sampled from demonstration sequences, POMDP simulation runs can be performed. This leads to weights: $\vec{\omega}^s = \{1,1,0,0,0\}$ and overall computational effort:

   - Generating O, generating R
   - $|v_g|$ POMDP policy computations
   - $|v_g| \times m$ simulation runs, typically $m = 2 * |\Xi_{Demo}|$
   - $|v_g| \times m \times n$ simulation steps, typically $n = 2 * avg|Q_{Obs}|$

   With POMDP policy computations taking several minutes with the best state-of-the-art algorithms for models used in experiments and generalization candidates ranging in the thousands, practical computation time would be around a week for a large model, which is infeasible. However, for small models with policy computation times of around one second and only several hundreds of v_g, this method is feasible. Simulation runs take less time than POMDP policy computation in practice.

2. Without O, but with a reward model R, MDP policy computation and initial simulation states sampled from demonstration sequences, MDP simulation runs can be performed. Weights are: $\vec{\omega}^s = \{1,1,0,0,0\}$ and the overall computational effort is:

   - Generating R
   - $|v_g|$ MDP policy computations
   - $|v_g| \times m$ simulation runs, typically $m = 2 * |\Xi_{Demo}|$
   - $|v_g| \times m \times n$ simulation steps, typically $n = 2 * avg|Q_{Obs}|$

   As MDP policy computation is much faster than POMDP policy computation, this method is also feasible for larger models. Here, simulation runs dominate overall computation time, for instance $|v_g| = 1000, m = |100|, n = |20| \Rightarrow 2\,000\,000$ simulation steps. Therefore, it may still take several minutes for larger models as investigated during experiments. In turn, resulting simulation courses of events are more coarse-grained approximations of realistic frequencies of courses of events than when using POMDP policies. Unless stated otherwise, this approach was taken in experiments.

3. With the same setup as in the previous point, but without simulation runs, value function utilities can be used, with a resulting weight vector of $\vec{\omega}^s = \{1,0,0,0,1\}$ or $\vec{\omega}^s = \{1,0,0.1,0.1,0.8\}$. The overall computational effort is composed of:

   - Generating R
   - $|v_g|$ MDP policy computations.

   In this case, MDP policy computations dominate the computational effort being faster than the previous simulation based setup.

4. Finally, without R and any value function, a very crude and very fast heuristic, transitions having a state as effect are taken as the primary criterion for the extended model with $\vec{\omega}^s = \{1,0,0.2,0.8,0\}$ and no further relevant computational effort. Evaluating all nonzero transitions on a typical sparse model takes negligible computational effort.

Transition relevance $\rho^v(v) \rightarrow [0,1]$ is computed for generalized transition hypotheses $v_g = (s_i, a_k, s'_j)$ and is composed of a set of measures $\rho^v_e(v) \rightarrow [0,1]$.

1. State relevance of the origin state s_i of v_g is considered:

$$\rho^v_1(v_g) = \rho^v_{originrelevance} = \rho^s(s_i) \qquad [4.69]$$

2. State relevance of the resulting state s'_j of v_g is considered:

$$\rho_2^v(v_g) = \rho^v_{effectrelevance} = \rho^s(s'_j) \quad [4.70]$$

3. While occurrence of origin and result are covered by state relevance, the transition v_g itself can be accounted for in the same manner, analyzing simulation runs:

$$Q_{sim*MDP_j} \in \Xi_{sim*MDP}$$
$$v_g \in Q_{sim*MDP_j} := (\ldots, s_i, a_k, s_j, \ldots) \in Q_{sim*MDP_j} \quad [4.71]$$
$$\Xi_{v_g \in Q_{sim*MDP}} := \{Q_{sim*MDP_j} | v_g \in Q_{sim*MDP_j}\} \quad [4.72]$$
$$\rho_3^v(v_g) = \rho^v_{v_g \in Q_{sim*MDP}}(v_g) = \frac{|\Xi_{v_g \in Q_{sim*MDP}}|}{|\Xi_{sim*MDP}|} \quad [4.73]$$

Ξ_{Demo} in turn cannot be assessed likewise as no v_g appears in demonstrations.

4. A crude heuristic is the relative reward of the pair (s_i, a_k), triggering the transition v_g:

$$\rho_4^v(v_g) = \rho^v_{relreward} = \frac{R(s_i, a_k) - min(R(\tilde{s}, \tilde{a}))}{max(R(\tilde{s}, \tilde{a})) - min(R(\tilde{s}, \tilde{a}))} \quad [4.74]$$

This measure indicates a likelihood that the pair (s_i, a_k) is included in courses of events and is used only when no simulation runs are performed.

5. Another crude heuristic is the probability of a transition v_g:

$$\rho_5^v(v_g) = \rho^v_{prob} = p_T(s'_j | s_i, a_k) \quad [4.75]$$

Again, this measure indicates the likelihood that the pair (s_i, a_k) is included in courses of events: thus, it is redundant when simulation runs, which provide a far better estimate, are performed.

6. While generalization confidence is implicitly included in most other measures by means of $p_T(v_g)$, it can be used separately without baseline transition probability playing a role:

$$\rho_6^v(v_g) = \rho^v_{genconfidence} = gc(v_g) \quad [4.76]$$

Finally, the weighted sum forms the overall transition relevance measure:

$$\rho^v(v_g) = \frac{\sum_{e=1}^{6} \omega_e^v \rho_e^v(v_g)}{\sum_{e=1}^{6} \omega_e^v} \quad [4.77]$$

Different circumstances require different weight vector $\vec{\omega}^v$ configurations in the same way and corresponding descending order of quality and computational effort discussed for state relevance. Although generalization confidence dominates all other measures, it is not emphasized so much in weight vectors as it is already indirectly involved by means of $p_T(v_g)$ in all other measures.

1. POMDP policy computation and simulations lead to weights
   $\vec{\omega}^v = \{0.5, 0.5, 1, 0, 0, 1\}$.

2. MDP policy computation and simulations lead to weights
   $\vec{\omega}^v = \{0.5, 0.5, 1, 0, 0, 1\}$.
   This is the default mode in experiments unless stated otherwise.

3. MDP policy computation without simulations leads to weights
   $\vec{\omega}^v = \{0.5, 0.5, 0, 0, 0, 1\}$.

4. Without MDP policy computation, a crude and fast heuristic can use
   $\vec{\omega}^v = \{0.5, 0.5, 0, 0.5, 0.5, 1\}$.

Finally, all generalized transition hypotheses $v_g \in V_g$ can be ranked accordingly. Based on the ranking, relevant hypotheses to be verified can be selected either above a certain ranking threshold ε_{ρ^v} or up to a maximum number N_v (the latter was the method used in experiments unless stated otherwise):

$$V_g^{verify} = \{v_g \in V_g |\ \rho^v(v_g) > \varepsilon_{\rho^v}\} \text{ or} \quad [4.78]$$

$$V_g^{verify} = \{v_g \in V_g |\ |v_g^{morerelevant}| < N_v, \rho^v(v_g^{morerelevant}) > \rho^v(v_g)\} \quad [4.79]$$

4.8.2. Interactive Request Generation

Relevant transition hypotheses V_g^{verify} have to be verified by acquisition of additional demonstrations $\Xi_{Moredemo}$, indicating nonzero occurrence probability or in case there is zero occurrence probability, non-occurrence confirmed by teacher rejection of potential demonstration courses of events. Such demonstrations, or their rejections, can be acquired by the robot posing demonstration sequence requests to the human teacher which include transition hypotheses $v_g \in V_g^{verify}$ in the courses of events. By these means, the original intention of transition generalization, namely relevance estimation and request generation, is achieved: supporting the human teacher in selection of demonstration sequences such that the number of demonstrations is kept low, but the number of important potential courses of events omitted is minimized.

To compose reasonable verification demonstration sequences, relevant transition hypotheses $v_g = (s_i, a_k, s'_j)$ have to be embedded into larger sequences $Q^{request}$, beginning at typical starting configurations as present in demonstration sequences S_1^{Demo}:

$$S_1^{Demo} := \{s_i \in S_1 | s_i = s_1 \in Q_{Obs_1} \vee \ldots \vee s_i = s_1 \in Q_{Obs_n}\} \quad [4.80]$$

$$Q^{request} = \{s_1 \in S_1^{Demo}, \ldots, s_i, a_k, s_j, \ldots, s_n\} \quad [4.81]$$

Such an internal representation of a request is suitable to be composed autonomously, while request output may be relaxed to allow more demonstration flexibility as described below.

As discussed in Section 4.4, demonstration sequences may vary considerably in length. Short sequences are suitable to demonstrate effect probabilities of specific transitions while minimizing redundant demonstration effort. Such shorter sequences may be subsections of longer coherent courses of events and the request-generation process tries to generate requests as short - and thus as efficient - as possible. To generate short sequences containing as many hypotheses v_g as possible and being connected to a demonstrated initial state $s_1 \in S_1$, path search is performed. A *valid path* between two states $Q_{path}(s_{start}, s_{end})$ is defined here as a sequence of consecutive transitions $(v_1, v_2, \ldots, v_n)$ with

$$\begin{aligned} Q_{path}(s_{start}, s_{end}) :=& (v_1 = (s_{start}, a_1, s'_1), v_2 = (s_2 = s'_1, a_2, s'_2), \ldots, \\ & v_n = (s_n = s'_{n-1}, a_n, s'_{end})), \quad [4.82] \\ & \forall 1 \leq i \leq n : \; p_T(v_i) > 0 \quad [4.83] \end{aligned}$$

In this context, finding a valid path can be achieved by searching on a graph represented by the transition model T_D. States s are nodes and nonzero transitions $v = (s, a, s'), p_T(v) > 0$ are edges in that graph. Yet generally, a transition model T_D forms a multigraph, with potentially multiple actions $a_{k_1} \neq a_{k_2}$ containing transitions from one distinct state s_i to another state s_j:

$$\begin{aligned} & v_{k_1} = (s_i, a_{k_1}, s'_j), v_{k_2} = (s_i, a_{k_2}, s'_j), p_T(v_{k_1}) > 0, p_T(v_{k_2}) > 0 \\ & \Rightarrow \exists Q^1_{path}(s_i, s_j), Q^2_{path}(s_i, s_j) : Q^1 \neq Q^2 \quad [4.84] \end{aligned}$$

Because path search in a multigraph is computationally expensive while for the purpose here, any likely path is sufficient, two restrictions are introduced to turn the multigraph into a graph:

1. For a set of immediate edges between two states $\{v_{k_1} = Q^1, \ldots, v_{k_n} = Q^n\}$ that are only distinguished by action a_k, only the edge with the highest probability $\forall v_k : p_T(v_{max}) \geq p_T(v_k)$ is chosen. In case of multiple transitions with equal, highest probability, an arbitrary edge can be selected.

2. From these transition edges, among all transitions with the same s, a, only that with the most likely effect is selected: $\forall v_{i,k,1}, \ldots, v_{i,k,n} : p_T(v_{i,k,max}) \geq p_T(v_{i,k,j})$.

Based on the resulting graph, a breadth-first search can be performed. It prefers short paths as desired and is able to discard all paths longer than a certain maximum sequence length.

Generating a request sequence is performed by applying such a path search, starting with the most relevant hypotheses $v_g^1 = argmax_{v_g}(\rho^v(v_g))$. First, a path to the closest demonstration initial state $Q_{path}^{min}(s_1 \in S_1, s \in v_g)$ is searched. If none is found, the hypothesis v_g is isolated from initially demonstrated courses of events and can be discarded. This should not happen if relevance was computed using simulations, because such an isolated transition then gets little relevance. If a path could be found, next a path with at most length max_l is searched for from v_g^1 to the next closest distinct hypotheses $v_g \neq v_g^1$. This is repeated at most max_g times. By these means, multiple hypotheses can be integrated into one request, saving effort on the one hand and leading to interesting new courses of events on the other. In case having these multiple new transitions in a single course of event is not valid within the scope of the mission, partial sequence rejection by the human teacher as discussed below can divide these longer sequences again.

The path search process is repeated for each v_g which has not yet been inserted into a sequence, starting at an initial state s_1, until no remaining v_g can be connected with an $s_1 \in S_1$ as shown in Algorithm 1. Because request generation uses the preliminary transition model T_D, generated from demonstrations, selected paths keep close to demonstrated courses of events which avoids generating sequences unsuitable for demonstrations.

Transition hypotheses v_g in a request sequence may actually not occur in a mission. While nonzero hypotheses can be confirmed in a respective frequency in further demonstrations, zero probability hypotheses are false hypotheses. Verification of the latter type can only be communicated to the robot by explicit rejection of corresponding demonstration sequences. However, in the request sequence composition process presented, several different hypotheses v_g may be included in a single sequence to increase efficiency. Some of those hypotheses may actually be nonzero while others are not. Furthermore, real transition probabilities may differ significantly, thus requiring different demonstration frequencies. To be still able to process multiple hypotheses in a single request, a binary selection tree mechanism is applied.

In case a request $Q^{request}$ containing multiple hypotheses v_g is rejected, it is split up into two parts, each containing a rounded equal number of hypotheses v_g: $Q^{request} = Q_{partA}^{request} + Q_{partB}^{request}$. Each part is recursively further processed, until a part either contains only a single hypotheses or is demonstrated. In case a sequence part contains only a single hypotheses

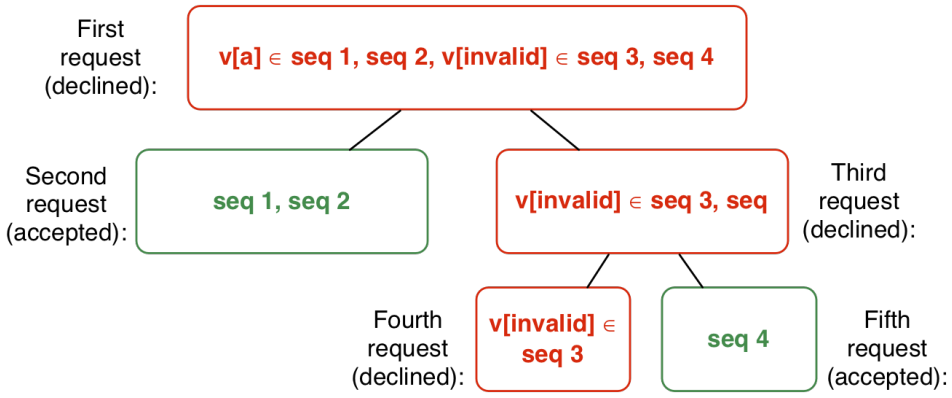


Figure 4.12.: Scheme of the binary tree request technique to discover which part of a rejected sequence is invalid. - [125]

$v_g^{inpart} \in Q_{part}^{request} \Rightarrow v_g^{inpart} \in V_g^{inpart}(part), |V_g^{inpart}(part)| = 1$ and is rejected, the hypothesis can be marked as false and is discarded. This is an efficient way of interactive hypothesis rejection as depicted in Figure 4.12.

Comfortable and flexible requests are provided by transforming the internal state-action request sequence $Q_{request}$ representation. Some flexibility is desired as requests should just point human teachers to interesting areas in model space, but not fully patronize them. Four basic types of lean request output representations can be used:

- Sequence of actions: just the action sequence in $Q_{request}$ is given. While this is a compact representation, the teacher may miss intended states.
- Sequence of states: just the state sequence in $Q_{request}$ is given. During demonstrations, the human teacher may choose any action to reach desired states. Unintended stochastic effects may be corrected by further actions, leading back to intended courses of events. On the other hand, being composed of features, state descriptions may be long.
- Sequence of feature state changes: only feature states in states which should change from one state to the next in $Q_{request}$ are given. This representation is very similar to the state sequence in functionality but much more compact concerning output.
- A combination of feature state changes and action sequences balances advantages best. A sequence of feature state changes is given with two special cases: a) in case a transition has $s = s'$, only the action is given, b) for v_g (but not other v_i), the action is stated explicitly, in addition to the feature state change. This is the default mode.

Finally, sequence requests have to be communicated to the human teacher. Three methods are available in the presented system:

- Written sequences, show sequences of symbol names for states or actions as text. It is the standard procedure in experiments unless stated otherwise.
- Verbal utterances read written sequences processed by the robots text-to-speech system. It is the most natural way, leading to a real student-teacher dialog, especially together with spoken teacher comments as discussed in Section 4.9.
- Visual display of the sequence in terms of an abstract sequence graph together with a visualization of states and actions. This is the best procedure in case expressive state or action symbols are not available for state or action space.

4.8.3. Limitations

Several conceptual limitations of the process as presented have to be considered in practical use as discussed briefly in the following. Most importantly, transition hypotheses are only generated and ranked for relevance *individually*, yet associated tuples of transition hypotheses may only introduce relevant new courses of events together. However, there is the issue that each potential combination of a set of hypotheses would need to be evaluated for relevance, which leads to utterly infeasible computational complexity. Investigation of sophisticated heuristics used for efficient selection and relevance evaluation of promising tuples was beyond the scope of this thesis. Hence, if tuples are only relevant if added together, this will not be detected because each alone is not sufficiently relevant. However, requests based on single hypotheses may still remind the human to include new demonstrations with directly coupled tuples of new transitions.

Overly precise requests are also a limitation although a human teacher is always free to add more demonstrations unrelated to, but inspired by requests. Demonstrations following requests may distort original frequencies of courses of events and thus transition probabilities. Therefore the teacher has to take care to correct altered probabilities in turn.

With autonomous feature generation, state symbols may not be expressive enough for human teachers to understand presented state sequences in written or verbal form. To a lesser extent the same may be true for action sequences. In this case, either a visual display or supportive visualization of textually addressed states and actions has to be used. Another way to acquire sufficiently descriptive state symbols is presented in the next Section.

Experiments regarding this process stage are presented in Section 5.6 while further software-implementation-specific details are outlined in [81].

4.9. Enriching Action Information with Spoken Human Comments

Interactive request generation enables an active participation of the robot student in the learning process. On top of this, an interactive two-way dialog supplementing passive learning from observations can be achieved by introducing spoken teacher comments. First, a clear distinction has to be made between spoken human utterances that are part of a demonstrated dialog between RR and HR on the one hand and spoken comments explaining aspects of demonstrations to the robot on the other. It can easily be achieved by introducing key words: any utterance starting with "Comment: [...]" is then interpreted as a demonstration comment, while all other utterances are part of a dialog in the presented system.

While demonstration requests by the robot remind the human teacher of demonstrations which might have been missed, human teacher comments explain demonstration aspects which might contain insufficient information when solely observed. Hence, the two-way interactive dialog helps the robot student to acquire more information than with a non-interactive passive PbD approach. Comments can be used to explain three types of demonstration properties:

1. Classify actions performed by the robot role demonstrator.
2. Tag states with expressive symbols, understood by teachers in demonstration requests.
3. Introduce additional action costs, for example when an HR is annoyed by certain RR actions.

When observing new actions during a demonstration, these actions can be autonomously mapped to executable skills as discussed in Section 4.3. However, it is not known which actions are closely related and thus suitable to derive action cost penalties, error states and secondary effect probability estimates as explained in the next Section 4.10. A spoken comment during demonstration of such a new action allows the knowledge base to classify the new action in relation to previously encountered classes of actions. E.g "Comment: this is a chair grasping" can derive a rough estimate of action cost and error effect likelihoods from a generic chair grasping class in the knowledge base.

Feature states generated autonomously as discussed in Section 4.2 usually do not have symbolic labels that can be easily interpreted by human teachers. Normally, that is not a problem as any abstraction during learning and execution is handled internally. However, when using state sequence spoken output in request generation, the human teacher has to understand state sequences. In this case, internal symbols representing mental models of courses of events have to be grounded between human and robot. Both agents need to have the same understanding of what a certain symbol means. It can be accomplished by commenting all relevant feature states

Figure 4.13.: *Illustrative example:* Use of spoken comments leading to symbols understood by humans e.g. in demonstration requests. - [125]

during demonstrations at least once as depicted in Figure 4.13. Recording points in traces with comments can be mapped on their respective states and these states labeled accordingly.

Action-cost negative rewards are assigned based on basic action class concepts as discussed in the next Section. However, certain interaction actions may be especially annoying to interacting humans. Further negative reward costs can be introduced using comments.

4.10. Inference of Missing Model Properties Using the Knowledge Base

By using inference on background knowledge valid for multiple missions as described in Section 3.7, the preliminary POMDP model can be completed. In the PMPM-PbD process, both exploitation of background knowledge for model generation as well as incremental expansion of the background knowledge by learned model data is suitable.

4.10.1. Inference of Knowledge to Complete Preliminary Models

As discussed in Section 3.7.1, several different POMDP model components can be partially inferred from DL background knowledge, given a preliminary model S_D, A_D, T_D, R_D. Below, inference and rule processing is described using examples based on persistent background knowledge DL axioms, facts and OPPL rules as presented in Section 3.7.4.

Based on A_D, action cost negative rewards R_C can be inferred. For example, the effort of grasping a chair can be inferred from the following persistent DL axioms:

$$Chair \sqsubseteq MovableObj \sqsubseteq Object \sqsubseteq TangibleThing \sqsubseteq \top \quad [4.85]$$

$$Grasp \sqsubseteq Action \sqsubseteq POMDPModelComponent \sqsubseteq \top \quad [4.86]$$

$$Grasp \sqsubseteq (\exists relatedObject.MovableObj) \quad [4.87]$$

$$Grasp \sqsubseteq (((\exists relatedObject.PortableObj) \sqcap (\exists reward.\{-1\})) \sqcup ((\exists relatedObject.\neg PortableObj) \sqcap (\exists reward.\{-2\}))) \quad [4.88]$$

Given a specific action instance generated from demonstrations observations, *Grasp Chair Front* $\in A_D$ is tagged by the action mapping process stage as a "Grasp", targeting the object "WoodenChair" with the manipulation strategy "Front". Hence, the action can be *realized* as an instance of a *Grasp*: *Grasp(GraspChairFront)*, with the properties *relatedObject(GraspChairFront, WoodenChair)* and *manipulationStrategy(GraspChairFront, Front)*. Subsequently, the Pellet reasoner can infer from the *Grasp* concept corresponding rewards as stated in Expression 4.88, the *Chair* concept and its parent concept *MovableObj*, which is not a *PortableObj*: *reward(GraspChairFront)* = *-2*. Such action cost values are then valid for all origin states as discussed in Section 3.7.

Based on A_D, robot capability specific error effect states S_E can be derived as shown by the example in Table C.3.

Based on A_D, S_E, S_D, robot capability secondary effect probabilities T_E can be inferred, which describe transitions that have never been observed in demonstrations of the current mission. Failures to grasp a chair, for instance, are modeled as instances by persistent DL axioms representing an n-ary relation with $n = 4$, as shown by the example in Table C.4. Additionally, an SWRL rule is necessary that matches *Grasp* instance properties *relatedObject* and *manipulationStrategy* with transition instances *TrGraspChairFrontMissed* and *TrGraspChairFrontAway* as shown in Table C.5. For an action *GraspChairFront*, related transitions *TrGraspChairFrontMissed* and *TrGraspChairFrontAway* are acquired, including effect state properties.

An example of a generic manipulation error recovery action A_E is shown in Table C.6.

To generate information gain actions A_I, one has to introduce a special relation *optimizedFor* has to be introduced, pointing at a *relatedObject*:

$$optimizedFor \circ relatedObject \sqsubseteq relatedObject \qquad [4.89]$$

It uses action "target" analysis as discussed in Section 4.11.1. Hence, an axiom and an OPPL rule can be applied, as shown in Table C.7.

4.10.2. Reducing Demonstration Requests Using Background Knowledge

Beyond completing preliminary models, the knowledge base can also be used to reduce the number of generalized transitions v_g to be verified by demonstration requests. Persistent knowledge about impossible transitions can filter the set of V_g, thus discarding some and reduce the necessary verification effort. For example, while there is an object inhand, the manipulation

strategy planner will not consider any command to grasp another object. Such a generally applicable static transition rule can be modeled as follows:

$$\begin{aligned} Grasp &\sqsubseteq (\exists cannotModifyAny.\{ObjectStateInhand\}) \\ ObjectStateInhand &\sqsubseteq ObjectState \end{aligned} \quad [4.90]$$

A similar rule is applicable for objects that are not present:

$$\begin{aligned} Grasp &\sqsubseteq (\exists cannotModifyAny.\{ObjectStateNotPresent\}) \\ ObjectStateNotPresent &\sqsubseteq ObjectState \end{aligned} \quad [4.91]$$

4.10.3. Extending the Knowledge Base Incrementally by Lifelong Learning

Persistent background knowledge that can be learned within the PMPM-PbD process includes secondary action effect states S_E and probabilities T_E as computed by geometric analysis and trials in simulation refinement stages in Sections 4.11 and 4.12.

4.11. Geometric Planning Analysis to Compute Scene-Specific Action Effects

Up to this point in the PbD process, mobility actions *goto pose* a_{gp} are regarded as abstract symbols, in turn transferring a robot from one class of abstract pose situations $c^{robot-pose}_{x_{robot-pose}} \in s$ into another $c^{robot-pose}_{x_{robot-pose}} \in s'$. In contrast, robot dialog utterances a_{ut} are easily instantiated and manipulation actions a_{mp} are clearly defined by the powerful manipulation strategy concept as outlined in Section 2.7.2 and mapped in Section 4.3. Symbolically referenced manipulation strategies are an abstract representation of manipulation motions within a constraint geometric, temporal and force profile as defined by the strategy graph. Constraint-based motion planning instantiates actual trajectory motion instances during execution time, anchored geometrically on objects in the scene. Accordingly, a component is necessary that is able to instantiate abstract mobility actions a_{gp} by geometric path planning.

However, the concept can be extended within the scope of PMPM-PbD: first, by coupling it closely to manipulation strategy actions a_{mp} and hence considering optimal positioning for manipulation actions as well as estimating transition probabilities for mobility actions a_{gp}. To achieve the former aspect, the given process stage [163] builds on the concepts of *Capability Maps* outlined in [167] and *Action Related Places* presented in [150].

Basically, this PMPM-PbD process stage has two areas of application:

1. Online computation of target poses to execute manipulation strategies with optimal success likelihood and path planning from a current pose to the target pose. Furthermore,

transition probabilities are computed for several classes of effects that are usable by the belief-filter system and thus online belief estimation.

2. Offline computation in learning-time PMPM-PbD transition model refinement uses Monte-Carlo simulation to compute abstract positioning and path planning effect prior probabilities from representative robot- and object-pose samples in feature states learned by PbD. It is based on exactly the same positioning, path and probability computations as the online, execution-time computation, thus representing it well in the transition model.

In the following discussion, only geometric aspects are considered for path and motion planning, hence the name of the process stage, but no physical dynamics world behavior which is addressed in Section 4.12.

4.11.1. Navigation Targets for Optimal Execution of Manipulation Strategies

First, it has to be noted that in a resulting POMDP model, an arbitrary mobility action a_{gp} and an arbitrary manipulation action a_{mp} are not directly linked. In decision making, a_{mp} may potentially follow a_{gp}, but so may any other available action as there is no fixed action sequence plan. Consequently, there is no exclusive tie between a_{gp} and any a_{mp} in a decision making policy. However, in the given system there is a source of information linking such two actions: demonstration sequences may show clear links with some manipulation actions a_{mp} most frequently being executed in certain $F_{robot-pose}$ feature states, which are effects of certain a_{gp}. Such information may be utilized in two ways:

1. Fusing a_{gp} and a_{mp} into a single subtask action a_{gp+mp} as discussed in Section 3.4.1.

2. Specializing a_{gp} in a way that it is optimized for execution of a selected a_{mp} afterwards.

In the following, only the latter option is discussed, because no true mobile manipulation is considered with manipulation strategy execution *during* robot mobility navigation and thus a clear temporal segmentation exists in any case. Thus, explicit distinct modeling of abstract mobility and manipulation actions is more flexible and coherent in the face of typical segmentation and handling of those actions. Nonetheless, *instantiation* of certain mobility actions $a_{gpformp}$ is then specialized to facilitate manipulation actions that are most frequently executed in typical effect states after an $a_{gpformp}$ in demonstrations. This leads to the following definition:

A subset of abstract mobility actions $A_{gpformp} \subseteq A_{gp} \subseteq A_D$ can be determined in a preliminary mission model acquired from demonstrations. Actions $a_{gpformp} \in A_{gpformp}$ are defined in the following way:

1. $a_{gpformp,i}$ is a mobility action.

2. There exists a manipulation action $a_{mp,k}$ that is the action most frequently performed directly after $a_{gpformp,i}$ in demonstrations:

$$\forall Obs_l, \forall obs(t') \in Obs_l : (a_i = a_{t'}, a_k = a_{t'+1}) \Rightarrow \hat{A}^{pairs}_{i,k} \leftarrow (a_i, a_k) \quad [4.92]$$

$$a_{i,maxfollows} = a_j \text{ with } j = argmax_k(|\hat{A}^{pairs}_{i,k}|) \quad [4.93]$$

$$a_i \begin{cases} \in A_{gpformp} & \text{if } a_i \in A_{gp} \wedge a_{i,maxfollows} \in A_{mp} \\ \notin A_{gpformp} & else \end{cases} \quad [4.94]$$

3. $a_{gpformp,i}$ is tagged in the knowledge base to be optimized for $a_{i,maxfollows} \in A_{mp}$ when instantiated.

4. Robot (RR) spoken utterances a_{ut} taking place after $a_{t'}$ are discarded when determining $a_{t'+1}$ as they will not alter pose and manipulation strategy origin.

Additional actions in the knowledge base included beyond demonstrations $A_{gpformp+} \subseteq A_{gp+} \subseteq A_E$, as outlined in Section 4.10, may be tagged in a similar way.

As objects related to a manipulation strategy action a_{mp} are known to the knowledge base – for instance as shown in Table C.4 – an object related to $a_{gpformp}$ is also known in turn. Based on that reasoning, actions $a_{gpformp}$ can be instantiated in a way that resulting most likely effect state situations increase desirable effect probabilities of the following a_{mp}. In this context, two specific feature state types concerning manipulation, corresponding to S_E as introduced in Section 4.10.1, have to be defined:

$$c^{\prime mp}_E = c^{\prime mp}_{success} := f_{furni-state}(mst :: finalnode \wedge c^{\prime obj-state}_{(OtherObj)} = c^{obj-state}_{(OtherObj)}) \quad [4.95]$$

$$c^{\prime mp}_E = c^{\prime mp}_{static} := c \quad [4.96]$$

In the context of planning a manipulation strategy a^{mst}_{mp} this means:

1. $c^{\prime mp}_{success}$:= A trajectory was found in the given scene and is is suitable to be executed under the constraints of the *mst*.

2. $c^{\prime mp}_{static}$:= No trajectory could be found during planning, thus no strategy can be executed. Any object-related feature remains the same after executing a^{mst}_{mp}.

Accordingly, these two definitions can model planning failure and in a respective transition model T_{GA} represent *estimated prior planning success probabilities* in a given class of scene configurations c^{mp}_{origin}. In the following, c^{mp}_x is a short notation regarding a feature state combination of $c^{robot-pose}_{x_i} \wedge c^{object-pose}_{x_j}$.

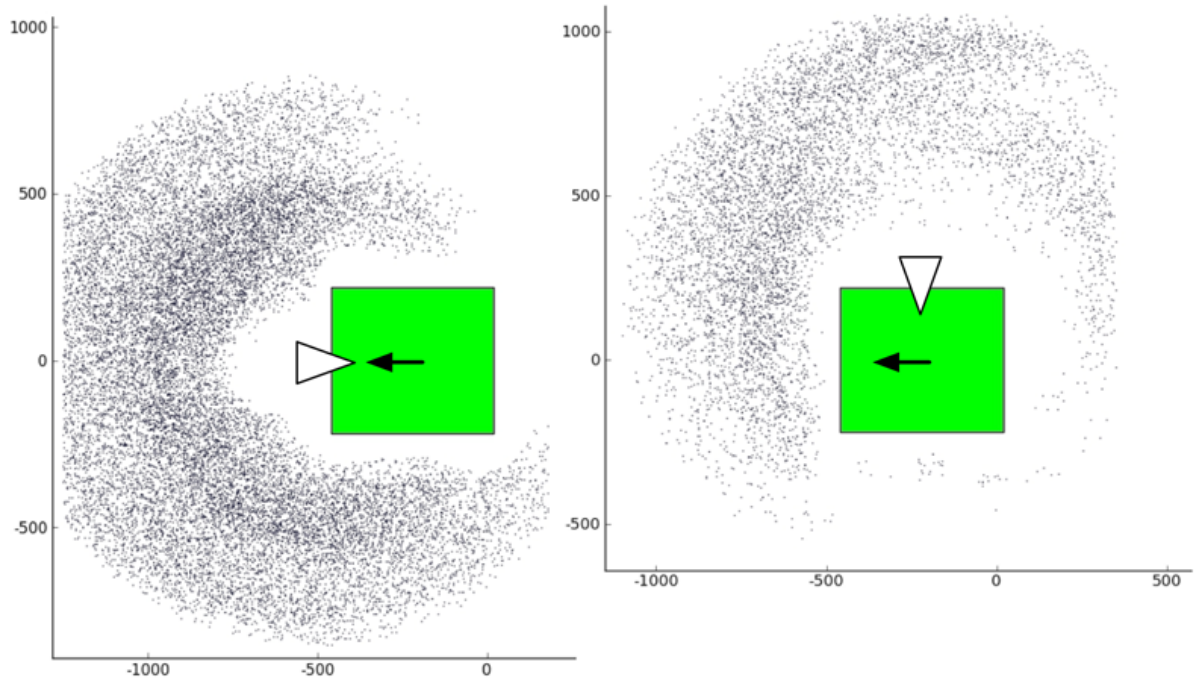


Figure 4.14.: Exemplary MST-ORSALM for "grasp chair below front" (left) and "grasp chair right side" (right) with chair bounding box in green and approximate *mst* grasp region shown by white triangles. Points indicate shoulder position of the robot in poses with successful trajectory planning. - [163]

1. $p_{T_{GA}}(c^{mp}_{success} \mid c^{mp}_{origin}, a^{mst}_{mp})$ = Manipulation strategy planning success probability.
2. $p_{T_{GA}}(c^{mp}_{static} \mid c^{mp}_{origin}, a^{mst}_{mp})$ = Manipulation strategy planning failure probability.

As successful planning is in any case a mandatory necessity for execution - and exactly modeled that way in software, with no motion being executed if no trajectory is found – any execution-specific effect probabilities are directly conditionally dependent on the planning success probability, as described in detail in Section 4.12.1. In this Section, only planning success and failure probabilities are considered as defined, with real execution effects added in Section 4.12.

With the given ingredients, the main purpose of the process component can be defined:

Computing a robot mobility target pose which keeps to the abstract definition of a given mobility action $a_{gpformp,i}$, trying to reach c^{mp}_{origin} in a way that planning success probability $p_{T_{GA}}(c^{mp}_{success} \mid c^{mp}_{origin}, a^{mst}_{mp})$ of the potentially following manipulation action $a^{mst}_{i,maxfollows}$ is maximized. Furthermore, computing both an online prediction as well as abstract transition-model estimate of transition probabilities for both $a_{gpformp,i}$ as well as $a^{mst}_{i,maxfollows}$.

This goal is achieved by combining and extending the ideas of *Action Related Places* and *Capability Maps*. First, for a given manipulation strategy on a certain object, a *Manipulation Strategy Object Relative Spacial Applicability Likelihood Map* (MST-ORSALM), inspired by Capability Maps, is computed in an extra offline stage. It is achieved by sampling robot poses relative to a given object, followed by MST constraint-based motion planning in that pose. Planning failure or success is recorded for each sample. Sampling has to uniformly cover all pose regions (x, y, θ) around an object.

An MST-ORSALM has to be computed only once for each combination of object and a_{mp}^{mst}. It can subsequently be used as fundamental background knowledge. Nonetheless, computational effort is enormous as every sample test needs a complete motion-planning step. Successful exemplary samples are shown in Figure 4.14. In the system used for evaluation, this took between 3 and 12 seconds per sample. Thus, sample numbers have to be kept low. It is achieved by defining an object-relative 2D+1D-grid voxel $g(x_i, y_j), (\theta_k)$ and taking the same number of samples in each grid field. The grid extends around an object as far as the maximum reachability of the robot for stationary manipulation. Grid resolution and sample density number have to be chosen in practice so that computational effort remains feasible as shown in the example in Table C.8.

Basically, MST-ORSALM gives a Monte-Carlo based prior probability estimate of manipulation strategy planning success $mst :: plansuccess$ of robot poses inside a given (x, y, θ)-voxel. Subsequently, computation uses voxel-based success probability estimates
$\text{MST-ORSALM}_{mst}^{obj} := p(mst :: plansuccess \,|\, f_{robot-pose}(x_i, y_j, \theta_k), a_{mp}^{mst})$.

Next, a positioning error can be accounted for easily, at least when assuming scene-independent navigation errors, sufficiently approximated by unidimensional Gaussians with means in x,y and θ. This is exactly the assumption made in the filter for robot-pose, discussed in Section 3.2.1. However, furthermore, fixed, average variances have to be assumed here to manage computational complexity. Using the approach described in Section 3.2.1, the probability to end up in neighboring voxels when aiming at the center of a voxel (x, y, θ), can be computed. The intention is to rank target poses for positioning more highly, where neighboring poses will more likely lead to success in subsequent manipulation strategy planning: thus, these poses are more robust to navigation errors with respect to strategy planning.

4.11.2. Navigation Path Execution Effect Probabilities

Ranking target poses for $a_{gpformp,i}$ is not sufficient, since the path towards that target pose also has to be assessed. A pose voxel highly suitable for strategy application may be reachable only under high navigation risk or not at all.

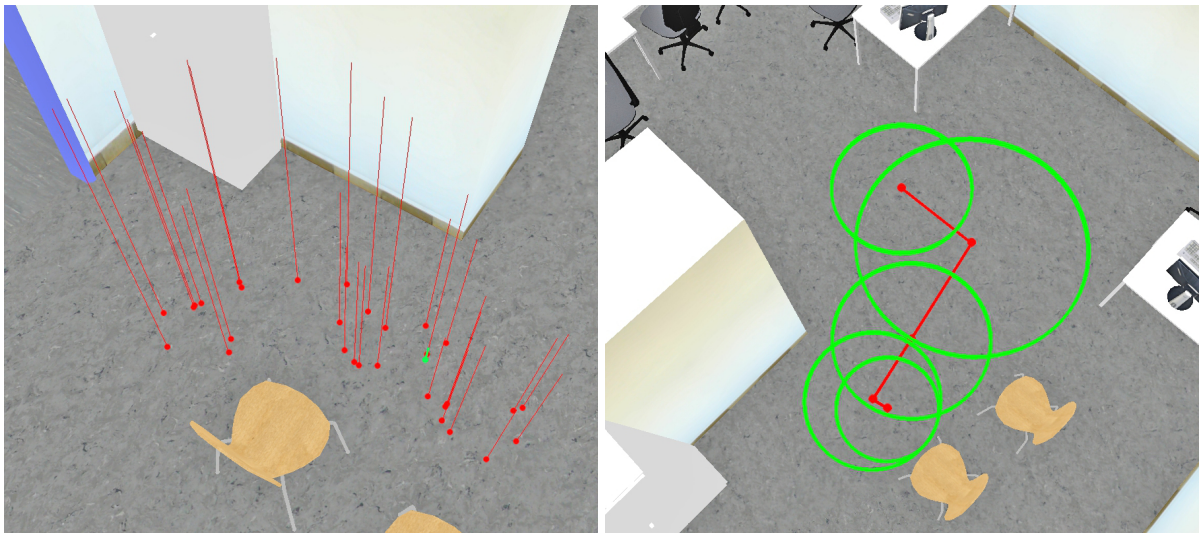


Figure 4.15.: Exemplary $n_{bestvoxels}$ target pose candidates for *mst* "grasp chair below front" with the chosen one in green (left). Robot pose θ is indicated by line height. A path towards the target pose with bubbles, but without interpolation spline (right). - [163]

Within the scope of the presented system, a specific path-planning component was developed, as described in detail in [163]. It is based on elastic bands and especially suitable for computing navigation-error estimates, based on geometry and navigation behavior peculiarities of a certain robot. Without loss of generality – any suitable path planning technique could be substituted - the method is able to compute a collision-free path for a non-holonomous base on a given map. This specific method tries to optimize both staying away from obstacles as much as possible and reducing path curvatures that lead to increased navigation errors.

If no path is found, *nopath* holds: $p^{goto}_{target,start}(nopath) = 1$, else $p^{goto}_{target,start}(nopath) = 0$. For a given path, deviations for each path segment g can be computed based on path curvature and robot mobility behavior, that is deviation angle per segment, in the form of representative deviation points x^d_g. Collision probabilities in each segment are calculated in turn from the relation between deviation points in collision with obstacles on the map and those without:

$$p^{segment}_g(success) = 1 - \frac{|x^d_g(collision)|}{|x^d_g|} \quad [4.97]$$

$$p^{path}_{target,start}(success) = \prod_g p^{segment}_g(success) \quad [4.98]$$

$$p^{path}_{target,start}(collision) = 1 - p^{path}_{target,start}(success) \text{ if } path \quad [4.99]$$

Finally, for a number of $n_{bestvoxels}$ best ranked target pose voxels (x_i, y_j, θ_k), a path is computed to the center of the voxel. Three effective action effects of executing the path as $a_{gpformp,i}$ are considered for each path computation:

$$c_E'^{gp} = c_{success}'^{gp} := f_{robot-pose}(c_{robot-pose}'^{gp} \in c_{origin}^{mp}) \quad [4.100]$$

$$c_E'^{gp} = c_{static}'^{gp} := c \quad [4.101]$$

$$c_E'^{gp} = c_{collision}'^{gp} := f_{robot-pose}(c_{robot-pose}'^{gp} = "Collision") \quad [4.102]$$

1. $c_{success}'^{gp} \simeq path(success)$ reflects successful arrival in the goal region, encompassing also moderate deviations from the exact target pose.

2. $c_{static}'^{gp}$ occurs in cases no collision free path could be found – even without considering deviations. It may happen in case all potential paths to the target pose are blocked by obstacles.

3. $c_{collision}'^{gp}$ considers prior collision estimates as predicted by path planning, defined above.

The path planning component tries to maximize overall likelihood for success of a_{mp}^{mst}:

$$\begin{aligned} &target(i,j,k) = \\ &\quad argmax_{i,j,k}(p(mst :: plansuccess \,|\, f_{robot-pose} = (x_i, y_j, \theta_k), a_{mp}^{mst}) \\ &\qquad * p^{path}_{(x_i,y_j,\theta_k),current-pose}(success)) \end{aligned} \quad [4.103]$$

$$p^{path}_{a_{gpformp},current-pose}(success) := p^{path}_{target(i,j,k),current-pose}(success) \quad [4.104]$$

During execution time, given a mobility action $a_{gpformp}$, the path planning component either plans and executes a path to a respective target pose or returns when no path could be found. Additionally, in case a path is taken, a prior collision estimate can be given to the belief filter.

4.11.3. Computing Abstract Mobility Action Probabilities for Mission Models

In the PMPM-PbD process, this processing stage contributes, by computing more realistic transition model probabilities for actions a_{mp}^{mst} and $a_{gpformp}$. Hence, it is a crucial part of the refinement stages: improving preliminary models generated from demonstrations to better reflect real-world dynamics, especially robot-capability-specific stochastic action effects.

Execution-time action effects are governed by peculiarities of path- and manipulation-strategy planning as outlined above. Therefore, during model generation, this stage replicates settings as observed during demonstrations. Respective prior action effect probabilities in T are computed by Monte-Carlo sampling of situations in feature states, corresponding to the execution of $A_{gpformp}$ and A_{mp}^{mst} and evaluating resulting planning effect prior probabilities.

For $n_{posepairs}$ pairs of robot poses and object poses $E_i = E_s^{GA}$ sampled from a scene layout, optimal target voxels, the respective manipulation strategy planning success probability, a path and path effect probabilities are computed. Scene layouts are represented by feature-state descriptions $c^{robot-pose}_{origin_a}$, $c^{object-pose}_{origin_a}$ generated from demonstrations. By using setups learned from demonstrations, the process can be sufficiently focussed on a few relevant state-action pairs. Consequently, action effect prior probabilities for $A_{gpformp}$ can be computed:

$$p_{T_{GA}}(c^{gp}_{success} \,|\, c^{gp}_{origin}, a_{gpformp}) = \frac{\sum_{i=1}^{n_{posepairs}} p^{path}_{a_{gpformp},E_i}(success)}{n_{posepairs}} \quad [4.105]$$

$$p_{T_{GA}}(c^{gp}_{collision} \,|\, c^{gp}_{origin}, a_{gpformp}) = \frac{\sum_{i=1}^{n_{posepairs}} p^{path}_{a_{gpformp},E_i}(collision)}{n_{posepairs}} \quad [4.106]$$

$$p_{T_{GA}}(c^{gp}_{static} \,|\, c^{gp}_{origin}, a_{gpformp}) = \frac{\sum_{i=1}^{n_{posepairs}} p^{goto}_{a_{gpformp},E_i}(nopath)}{n_{posepairs}} \quad [4.107]$$

For manipulation planning effect prior probabilities only those $n_{mppairs}$ pairs $E_j = E_s^{GA}$, resulting from a valid path target pose at the beginning of actions A^{mst}_{mp} are considered:

$$p_{T_{GA}}(c^{mp}_{success} \,|\, c^{mp}_{origin}, a^{mst}_{mp}) = \frac{\sum_{j=1}^{n_{mppairs}} p^{mst}_{a_{mp},E_j}(planning-success)}{n_{mppairs}} \quad [4.108]$$

$$p_{T_{GA}}(c^{mp}_{static} \,|\, c^{mp}_{origin}, a^{mst}_{mp}) = \frac{\sum_{j=1}^{n_{mppairs}} p^{mst}_{a_{mp},E_j}(planning-failure)}{n_{mppairs}} \quad [4.109]$$

Subsequently, the knowledge base has to anchor feature states for conditional dependence of probabilities in T_{GA} on probabilities in T_D. A Bayesian Network is implicitly created, with conditional dependence links tied to transitions from demonstrations by means of the knowledge base. This linking is performed in the same manner as outlined in Section 4.12.3. An example is shown in Table C.9.

4.11.4. Discussion and Limitations

By means of this component, abstract actions are grounded in path and motion planning behavior. Furthermore, abstract actions $a_{gpformp}$ are instantiated. Considering path and motion planning peculiarities in abstract reasoning can help action selection to make robust decisions.

However, there are many limitations to this specific approach that could potentially be tackled in future investigations. Foremost, an MST-ORSALM is only valid for a fixed scene of objects, such as a chair or a cup on a fixed table, but not for other nearby obstacles, that were absent from the scene during generation of the MST-ORSALM. Variable sample density of the

MST-ORSALM could allow faster computation times and a focus on interesting regions in the workspace relative to objects. True mobile manipulation with navigation during manipulation motions is not considered here at all, as non-holonomous motions do not coexist well with manipulator motion planning. Additionally, the instantiation is specialized to directly execute the manipulation action after a navigation action $a_{gpformp}$. While any other action can be chosen afterwards, path planning does not consider further alternative courses of events. Consequently, only the most likely subsequent manipulation action is considered. In contrast, a more extensive probabilistic approach could reflect probabilities of all potentially applicable manipulation strategies and optimize positioning for the overall maximum effect likelihood. Limitations of various Monte-Carlo methods furthermore limit the ability to approximate real-world dynamics.

These and further limitations non-withstanding, reflecting detailed skill level action effect peculiarities in abstract level transition models and acquiring numeric values comfortably is a big challenge in robotics, even far more than reflecting perception peculiarities as discussed in Section 3.3.4. The challenge is tackled at least to some extent by this crucial component in the given system. Extensive details regarding the specific elastic-band planning technique and deviation computations applied as well as software-specific details of this stage can be found in [163].

4.12. Trials in Dynamics Simulation to Refine Manipulation Effect Probabilities

While logic-based inference and geometric analysis can give crude estimates of robot-specific transition probabilities p_T, learning from experience gives the best approximation for robot action effect prior probabilities. In combination with PbD this means that a robot executes observed state-action pairs (s,a) and then is able to gather experience about secondary effect probabilities as defined in Section 3.4.2. In practice, there are severe limitations to learning from experience:

1. State-action pairs have to be experienced a number of times to gather frequencies sufficiently approximating $p_T(s,a,s'_1),\ldots,p_T(s,a,s'_n)$.

2. Situations instantiating abstract states s and actuator commands instantiating abstract actions a have to be sampled in sufficient variance.

3. Effect states s'_i have to be observed with sufficient accuracy for the system to be able to assume a fully observable setting, which is necessary for learning effect frequencies.

While learning with a real robot in this way may approximate real world dynamics best, it is far too slow, dangerous to hardware and environment. In addition, it is difficult to control

situation choice and to record action effects. Consequently, an alternative approximating real settings is necessary, but without the need for real hardware. Physical-dynamics simulation is able to compute realistic effects of forces of bodies onto each other. In combination with artificial deviations reflecting real-world sensor and actuator uncertainty, it enables simulated learning from experience [12], [135]. In the scope of the presented system, physical-dynamics simulation is used for learning manipulation action secondary stochastic effect prior probabilities from simulated experience. Manipulation actions are the most complex actuation-domain with secondary effects prevalent in real missions and these effects cannot be computed to a sufficient approximation by geometric analysis alone.

4.12.1. Simulated Trial Setup

A simulation trial is defined by the tuple state s, action a, observation deviations $\{\sigma_i^m\}$, actuation deviations $\{\sigma_j^a\}$ and sample densities and frequencies ρ_k: $Trial := (s, a, \{\sigma_i^m\}, \{\sigma_j^a\}, \rho_k)$. A state s is an abstract class of situations as defined in Chapter 3. For trials, a state has to be re-instantiated, typically from its feature definitions f_{g1} and thus filter models as described below. More specifically, a state s_{manip} in this context is defined as a class of configurations ξ of a robot and object poses in an environment:

$$s_{manip} := C_{Rob}, (Env := Obj^1_{pose}, \ldots, Obj^n_{pose}) \quad [4.110]$$

$$\xi := c_{Rob} \in C_{Rob}, (x^1_{obj}, y^1_{obj}, z^1_{obj}, r^1_{obj}, p^1_{obj}, y^1_{obj}) \in Obj^1_{pose}, \ldots \quad [4.111]$$

This simulation environment definition assumes rigid objects with known geometric models, mass distributions and unambiguous poses. A manipulation action in this context is a manipulation strategy a^{mst}_{mp} as defined in Section 2.7.2. Typically, a trajectory is planned by applying the manipulation strategy in a given scene s_{manip}. However, for simple grasping strategies it may also be suitable to plan with a simplified representation a^{sgrasp}_{mp}. Such a simplified manipulation action representation is defined by

- Target object $Obj^t = Obj^i$,
- Approach vector $\vec{p} = (x_p, y_p, z_p)$,
- Approach vector position $\vec{bp} = (x_{bp}, y_{bp}, z_{bz})$,
- Hand roll θ_r.

Furthermore, robot-specific intermediate and final configurations resembling strategy nodes have to be defined, which describe interim targets to be reached by planning:

- Interim target arm configuration *arm approach* $W_p : (w_1, \ldots, w_{armDOF})$
- Interim target hand configuration *hand approach* $H_p : (h_1, \ldots, h_{handDOF})$

Using classic motion planning techniques, based on $a_{mp}^{sgrasp} := \{Obj^t, \vec{bp}, \vec{p}, \theta_r, W_p, H_p\}$, an approach trajectory for arm and hand joint angles can be computed.

Observation deviations $\{\sigma_i^m\}$ define object pose measurement probabilities relative to real objects poses. In simulated trials, the observed object pose is assumed for trajectory planning, reflecting real-world behavior. A simple example is normal distributions for $(x_{obj}^i, y_{obj}^i, z_{obj}^i)$, with $\mu_x^i = x_{obj}^i, \mu_y^i = y_{obj}^i, \mu_z^i = z_{obj}^i$ neglecting orientation and covariance. Observation deviation distributions can be acquired from observation models as discussed in Section 3.3.

Actuation deviations $\{\sigma_j^a\}$ reflect known low level actuator imprecisions, such as finger angles in cable driven hands or interpolation errors in trajectory execution. As observation and motion planning deviations dominate in many settings, actuation deviations can often be neglected.

Finally, sample densities ρ_k define how many configurations ξ are sampled for states by which distributions and if a single or multiple observed configurations ξ^{obs} are considered.

4.12.2. Simulated Trial Execution and Result Evaluation

Learning from trials in dynamics simulation consists of four process stages for each trial run:

1. Setting up intrinsic and observed world configurations.
2. Motion planning based on the observed world configuration.
3. Physical-dynamics motion execution in the intrinsic world configuration.
4. Evaluating the resulting world configuration.

In a single trial run, an intrinsic world configuration ξ_i is sampled from the state description, first: $\xi_i \sim \rho_{real}(s)$. In case there are multiple objects with potentially overlapping pose regions in a state s, collision checking has to ensure that a sampled world configuration is valid and objects do not overlap in simulation. For example, $\rho_{real}(s)$ could reflect Halton sequences in feature state description boxes of $f_{robot-pose}$ and $f_{furni-state}$. Next, multiple observations $\xi_{j,i}^{obs}$ can be sampled based on the intrinsic configuration $\xi_{j,i}^{obs} \sim \rho_{obs}(\xi_i)$. By these means, characteristic object localization uncertainties are reflected.

Based on each observed world configuration $\xi_{j,i}^{obs}$, motion planning parameterized by a_{mp} generates an actuator trajectory. As observed and intrinsic scene differ, the resulting trajectory

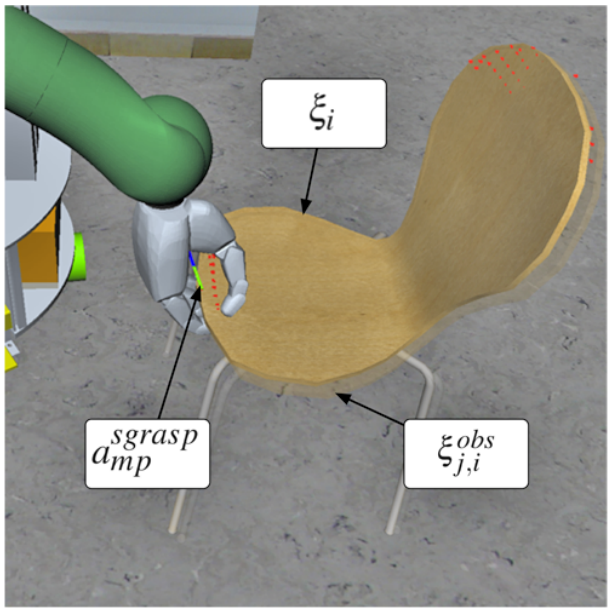


Figure 4.16.: Illustration of a dynamics simulation trial as visualized in the framework used: execution of an a_{mp}^{sgrasp} leading to $c_{success}^{!mp}$ despite some deviations $\rho_{obs}(\xi_i)$. - [125]

may lead to collisions in the intrinsic scene, even when perfectly executed. Such behavior is typical for motion planning and execution on the real robot with real-world object localization.

Hence, each planned trajectory is executed on the intrinsic scene ξ_i with further, typically minor, actuation deviations $\{\sigma_j^a\}$ applied to joint angles. Such an execution takes place in the simulation environment with a rigid-body physical-dynamics engine active. Consequently, contact forces between robot and objects as well as between objects lead to realistic effects. Capacity to closely simulate real-world action effects depends on quality and parameterization of the dynamics engine used. In experiments, the OpenRAVE environment [33] with physics engine ODE [143] and some custom extensions [12] was used.

Dynamics simulation is performed until the scene settles down or a termination condition of a_{mp} is reached. Examples of the latter are an end node of a strategy a_{mp}^{mst}, final joint poses of a simple action a_{mp}^{sgrasp} or stable grasp forces at simple actions a_{mp}^{sgrasp}.

After dynamics simulation has stopped, the resulting configuration ξ_r can be determined. It reflects a sample of the effect configurations resulting from the action a_{mp} executed in a configuration $\xi_i \in s$ as illustrated in Figure 4.16. Therefore, it is an instance of the transition (s, a, s') with $\xi_r \in s'$ and reflecting a single simulation particle. Consequently, the whole dynamics simulation trial stage to compute effect probabilities is a typical *Monte-Carlo method* [22].

The resulting configuration ξ_r is evaluated for robot configuration c_{Rob}, object configurations $\{Obj_{pose}^k\}$ and forces on the robot. The latter can lead to new special feature-state descriptions

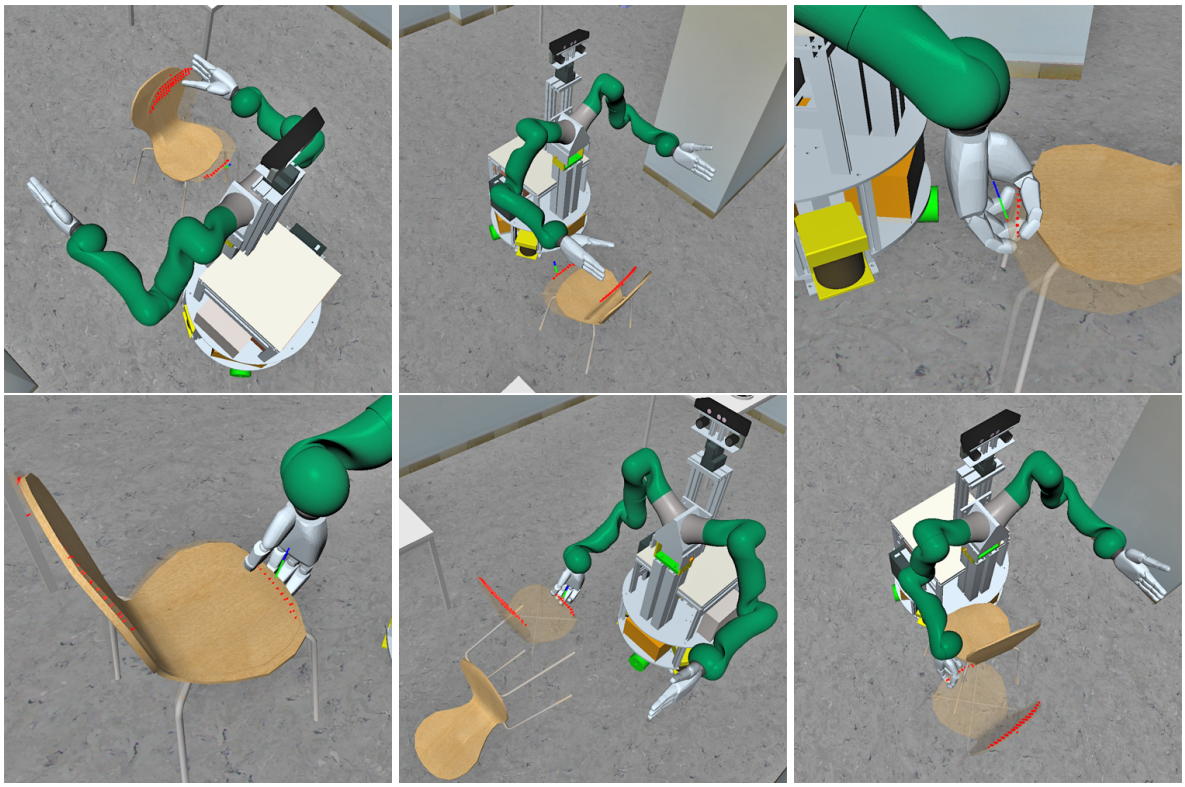


Figure 4.17.: Illustration of dynamics simulation effect categories. From top left to bottom right: c, c, c', $c^{\prime mp}_{success}$, $c^{\prime mp}_{gone}$, $c^{\prime mp}_{jammed}$. - [125]

not reflected by the initial state set S_D used to determine origin state s. Configurations represented by such special feature states c^{mp}_i evaluated in this context are:

$$c^{\prime mp}_E = c^{\prime mp}_{success} := f_{obj}((mst :: finalnode \wedge c^{\prime obj-state}_{(OtherObj)} = c^{obj-state}_{(OtherObj)}) \quad [4.112]$$

$$c^{\prime mp}_E = c^{\prime mp}_{static} := c \quad [4.113]$$

$$c^{\prime mp}_E = c^{\prime mp}_{gone} := f_{obj}(c^{\prime obj-state}_{(TargetObj)} = \text{"}Other\text{"} \wedge c^{\prime obj-state}_{(OtherObj)} = c^{obj-state}_{(OtherObj)}) \quad [4.114]$$

$$c^{\prime mp}_E = c^{\prime mp}_{jammed} := f_{obj}(\exists c^{\prime obj-state}_{(Obj)} = \text{"}Jammed\text{"}) \quad [4.115]$$

$$\{c^{\prime mp}_E\} = \{c^{\prime mp}_{moved}\} := f_{obj}(\neg mst :: finalnode \wedge c^{\prime obj-state}_{(TargetObj)} \neq c^{obj-state}_{(TargetObj)} \wedge c^{\prime obj-state}_{(TargetObj)} \neq (\text{"}Other\text{"} \vee \text{"}Jammed\text{"})) \quad [4.116]$$

$$c^{\prime mp}_E = c^{\prime mp}_{chaos} := \neg c^{\prime mp}_{success} \wedge \neg c^{\prime mp}_{static} \wedge \neg c^{\prime mp}_{gone} \wedge \neg c^{\prime mp}_{jammed} \wedge \neg c^{\prime mp}_{moved} \quad [4.117]$$

1. $c^{\prime mp}_{success}$ reflects successful execution of a^{mst}_{mp}, reaching the final goal constraint node while not altering any other object state (Equation 4.112).

2. $c^{\prime mp}_{static}$ occurs after unsuccessful execution of a^{mst}_{mp}, without changing the object layout. This may result from missing the manipulation goal while not moving any objects around significantly (Equation 4.113).

3. $c^{\prime mp}_{gone}$ represents pushing a target object (and only that) outside any known feature state (Equation 4.114).

4. $c^{\prime mp}_{jammed}$ reflects jammed configurations: an object Obj^l exerts a force onto the robot at points outside a stable grasp configuration (Equation 4.115).

5. $\{c^{\prime mp}_{moved}\}$ includes all potential obj-states where a^{mst}_{mp} has failed and objects have been pushed into other feature states, but not outside or into $c^{\prime mp}_{jammed}$ (Equation 4.116).

6. $\{c^{\prime mp}_{chaos}\}$ includes all potential obj-states where only some other object in the scene was pushed around (Equation 4.117).

Further differentiation can be made when applying a^{sgrasp}_{mp} grasps. In this case, resulting grasp stability can be evaluated by measuring simulated forces of the fingers on the object and thus a resulting *Grasp Wrench Space* W_r [17] . Such W_r is a value representing the stability of the grasp and applied in a^{sgrasp}_{mp} dynamics simulation result evaluation [12]. Consequently, there is further distinction within $c^{\prime mp}_{success}$:

$$c^{\prime mp}_{E} = c^{\prime sgrasp}_{success} := f_{obj}(W_r \geq \varepsilon^{wrench}_{grasp} \wedge c^{\prime obj-state}_{(OtherObj)} = c^{obj-state}_{(OtherObj)}) \quad [4.118]$$

$$c^{\prime mp}_{E} = c^{\prime sgrasp}_{unstable} := f_{obj}(W_r < \varepsilon^{wrench}_{grasp} \wedge c^{\prime obj-state}_{(OtherObj)} = c^{obj-state}_{(OtherObj)}) \quad [4.119]$$

1. $c^{\prime sgrasp}_{success}$ reflects a stable grasp on a target object Obj^k with *Grasp Wrench Space* W_r above a threshold $\varepsilon^{wrench}_{grasp}$ (Equation 4.118).

2. $c^{\prime sgrasp}_{unstable}$ represents a partially successful, unstable grasp on a target object Obj^k with *Grasp Wrench Space* W_r below a threshold $\varepsilon^{wrench}_{grasp}$ (Equation 4.119).

Subsequently, state membership s'_r of ξ_r is determined and the transition occurrence recorded. Trials are performed for all sampled observed configurations $\xi^{obs}_{j,i}$. This process is repeated for all sampled intrinsic configurations ξ_i. In the end, transition frequencies of all occurring trial effects s'_r are counted for a state-action pair (s,a) transition frequency model T^F_{Sim} and normalized to get action effect probabilities $p_{T_{Sim}}(s'|s,a)$ in the same manner as described for demonstration transition frequencies in Section 4.6.

Just as the robot learns effect probabilities from frequencies of courses of events another agent has demonstrated, it learns from experiencing frequencies of courses of events itself. The

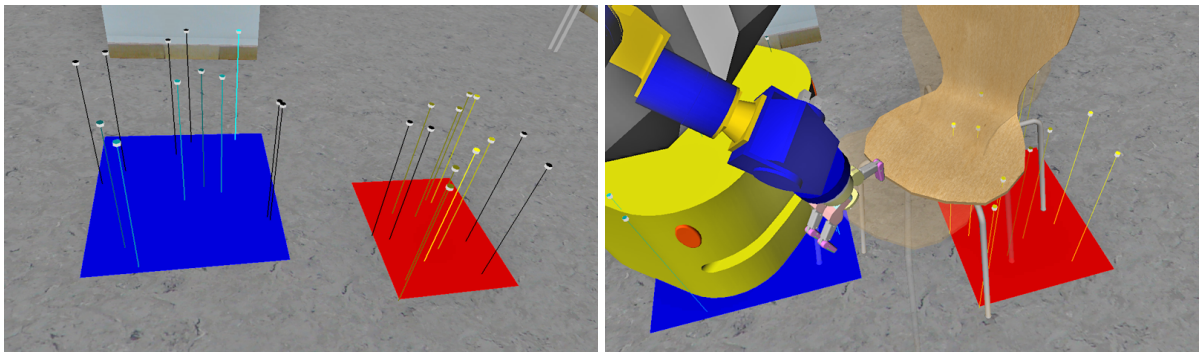


Figure 4.18.: Illustration of a dynamics simulation trials based on PbD generated feature states. Observation deviation $\rho_{obs}(\xi_i)$ is exaggerated, ξ_i are shown by points. The reference pose of the chair Obj_{pose}^{chair} is located in its backrest. Bright colors of points indicate that a trajectory could be planned and simulated for many corresponding part (robot or object) poses. - [125]

former is encompassed by PbD, the latter by learning from trials, with the simulation aspect only introduced to make the method feasible in practice.

4.12.3. Simulating Experience Based on Demonstrated Missions

The biggest challenge for trial learning is to find a suitable focus of configurations to gather beneficial experience from. As discussed in Section 5.8, exploring a whole transition model $\forall s \in S, \forall a \in A : doTrials(s,a)$ for learning from experience is infeasible because of the computational effort to compute dynamics simulations and the requirement to test multiple configurations for good approximations of effect frequencies.

However, when integrated with mission level PbD, a narrow focus can be achieved. Manipulation actions a_{mp} are only considered in states s in which they were performed during demonstrations. State-actions pairs (s,a), probed in trials are significantly reduced by this approach which makes learning from experience complementary to learning from demonstrations. Considered features are generated during demonstration analysis stages and sufficiently focussed to be efficiently applied for configuration ξ sampling.

After effect probabilities T_{Sim} for selected pairs (s,a) have been computed as described in the previous Section, they have to be integrated with effects in T_D or T_{D+GA}. When integrating T_{Sim} and T_{D+GA}, two aspects have to be considered:

- T_{Sim} may contain new (error) states not yet present in T_{D+GA}, even after including background knowledge.
- Fusing rows in T_{Sim} and T_{D+GA} has to be clearly defined.

Accordingly, as shown in Section 4.11, the knowledge base is used to determine anchor feature states for conditional dependence of probabilities in T_{Sim} on probabilities in T_{D+GA}. The Bayesian Network formed by conditional dependences between T_D and T_{GA} is extended, linking T_{Sim} and T_{GA}. During simulation trials, only the case that the motion planner found a solution and no further external interference, for instance from humans, is considered. Therefore, all probabilities are relative to that case, which is basically the transition to $c^{mp}_{success}$ in T_{GA}. As a result, all effects in T_{Sim} are conditionally dependent on $T_{GA} : c^{mp}_{success}$ with $c_o := c^{mp}_{origin}$:

$$p_{T_{D+GA}}(c^{mp}_{success} \,|\, c_o, a_{mp}) \Rightarrow p_{T_{Sim}}(c^{mp}_E \,|\, c_o, a_{mp}, T_{D+GA}(c^{mp}_{success})) \quad [4.120]$$

$$\Rightarrow \forall c^{mp}_E \in T_{Sim} :$$

$$p_{T'_{D+GA+Sim}}(c^{mp}_E \,|\, c_o, a_{mp}) = p_{T_{Sim}}(c^{mp}_E \,|\, c_o, a_{mp}) * p_{T_{D+GA}}(c^{mp}_{success} \,|\, c_o, a_{mp}) \quad [4.121]$$

$$p_{T'_{D+GA}}(c^{mp}_{success} \,|\, c_o, a_{mp}) = \begin{cases} p_{T_{D+GA}}(c^{mp}_{success} \,|\, c_o, a_{mp}) & \text{if } c^{mp}_E \neq c^{mp}_{success} \\ 0 & \text{if } c^{mp}_E = c^{mp}_{success} \end{cases} \quad [4.122]$$

$$p_{T_{D+GA+Sim}}(c^{mp}_E \,|\, c_o, a_{mp}) = p_{T'_{D+GA}}(c^{mp}_E \,|\, c_o, a_{mp}) + p_{T'_{D+GA+Sim}}(c^{mp}_E \,|\, c_o, a_{mp}) \quad [4.123]$$

It has to be noted, that semantics between $c^{mp}_{success}$ in T_{GA} and $c^{mp}_{success}$ in $T_{D+GA+Sim}$ have changed, with the former just denoting successful motion planning, while the latter is more strict, representing successful manipulation strategy execution. As execution can take place only after successful planning, conditional dependence is well defined in this case. An example is outlined in Table C.10.

By these means, very robot-specific secondary effect probabilities are added to the model while the trial space is kept small keeping to the demonstration model skeleton.

4.12.4. Discussion and Limitations

Learning from experience in dynamics simulation is the final stage in the presented process, generating the last values for a final POMDP mission model. Any true online learning from experience, not considered in the scope of this thesis, would seamlessly fit in after this stage.

With the given approach there are naturally several limitations. Apart from any limitation introduced by the simulation aspect, there is also the problem that skill domains beyond manipulation are not considered. For example, human interaction behavior is considered conditionally independent from manipulation action error effects. This may not reflect real-world behavior properly, but could only be learned from physical learning from experience, including assessment of concurrent human behavior, for instance when a jamming of objects occurs. Feature

interdependences in transition probabilities are therefore not considered in this stage as they are impossible to include in the concept of learning from experience in simulation.

Furthermore, an important limitation is the temporal demand of this process stage as it is extremely computationally expensive. Although it is an offline stage, computational requirements may be excessive. Priming simulation by feature states, learned from PbD is a major aspect to tackle this challenge, but some limitations remains. This, and other evaluation aspects are discussed further in Section 5.8. More software-implementation-specific details concerning this process stage can be found in [12] and [135].

4.13. Programming by Demonstation Conclusion and Discussion

As presented, the PbD process computes state and action grounding from demonstrations, followed by generation of a preliminary model, model space exploration in interaction with a teacher and finally completing the model by computing robot-specific model aspects from various sources.

In this process, the robot interacts with both human teachers and simulated objects in the environment to gather knowledge about the mission. Thus, the *interactive learning* aspect spans not only Human-Robot Interaction, but also interactions with all domains of the world.

However, the learning process is strictly distinct from the robot acting during execution time. In a way, offline PbD with refinement computations is similar to human dreaming, with new observations being reorganized to generate refined action capabilities by reanalyzing memory traces [161]. While not investigated within the scope of this thesis, the refinement concept scales to learning from experience and refining models during execution-time. Furthermore, the process could easily be extended to models similar but not equivalent to pure (PO)MDP models if the complexity issues of the latter are shown to be unsurmountable in the future. Any task model with abstract states, actions and probabilistic Bayesian action effects can benefit from some of the presented process stages.

5. Evaluation

Evaluation of the concepts and algorithms presented in the previous two chapters has to show the ability of the system to generate models for abstract-level decision making and execute them autonomously on service robots acting in real-world settings.

Hence, all previously presented information-processing components are realized in software and integrated in a coherent architecture. This architecture is tightly integrated with physical robot sensors and actors on real robots as presented in Section 3.1.1. Experiments including autonomous execution-time action selection and recording of human mission demonstrations can therefore utilize data measured with physical sensors and actions steering physical actuators.

However, when analyzing experiments in this complex evaluation setting, there are different, conflicting aspects. On the one hand, integration is to be shown with process stages working together. On the other hand, capabilities and limitations of individual stages have to be analyzed in more detail. Autonomous execution and learning have to be assessed individually and in combination, while successful interplay of multiple skill domains must to be demonstrated.

To achieve a focus on both component details and the whole process, it has to be evaluated from different perspectives. On the on hand, some focussed experiments with individual process stages have to be performed, highlighting peculiarities of these stages. On the other hand, some integration experiments with multiple parts involved have to be carried out. Analysis of the former puts a focus on individual parts and then shows multiple process stages from recording to autonomous execution participating to enable an experiment mission.

Roughly the following evaluation setup types can be distinguished:

1. Experiments with real robots including only the execution time decision making system without any learning and thus relying on manually designed models. These can be considered to be experiments within the scope of Chapter 3 alone.

2. Evaluation of individual PbD process stages with naturally or artificially created data.

3. Recording of real demonstrations and basic model generation with manually designed state and action mappings and fixed background knowledge, followed by policy execution.

4. Naturally or artificially recorded demonstrations processed by multiple PbD process stages, followed by policy execution.

As hardware platforms, available robots Albert and Adero were used. These capable robots with a wide range of sensors and actuators are well-suited representatives of typical contemporary full grown autonomous service robots.

5.1. Experiment Setup Overview

In this Section, setup properties valid in multiple experiments as well as a setup classification notation are presented. Robot setups or simulation environments on the one hand and specific service missions on the other are reusable settings that can be utilized in various experiments.

5.1.1. Robot and Simulation Setups

For experiments with both the physical robot and in simulation, the general setting of an indoor cafeteria with the robot having the role of a butler was chosen. Such a setting is versatile and can easily be replicated in laboratory premises. For each skill domain, a number of aspects was chosen, depending on the capabilities of available skill components, as presented in Section 3.1.1:

1. Mobility: laboratory rooms with a known map, including walls, cupboards and fixed furniture, sufficient spacing for large robots to navigate in and several fixed tables on which to manipulate small objects.

2. Spoken human-robot interaction: flexible sets of small, trained grammars usable for speech recognition.

3. Human-robot interaction by body posture: flexible sets of small, trained human body activity classifier sets.

4. Object manipulation: a set of rigid objects with a known mesh model and detectable by object localization mechanisms. This set includes a round table, round chairs, cups of different colors, a pringles can, plates, a spatula and cereal boxes, some of them shown in Figure 5.1.

For visualization, path and motion planning, geometric analysis as well execution in dynamics simulation, geometric models of all physical objects in the scene exist. Surface friction and mass distribution models of manipulable objects are used by dynamics simulation. Several geometric environments are used to visualize the scene as represented inside the robot:

Figure 5.1.: Locatable and manipulable objects used in several missions. - [125]

- OViSE [73], based on the open 3D graphics engine OGRE [148] is a high-performance visualization system developed in close collaboration with development of the presented system. It is used to visualize the laboratory, robots in their current configuration, rigid objects in their current configuration, human body configurations and additional information like trajectories, feature state regions or pose samples. In combination with a mediator component, OViSE is able to visualize the scene as observed by the robot in real time or alternatively can replay scenes and configurations observed by the robot in the past as shown in Figure 5.2. Most computer-graphics based visualizations of scenes in this and the previous Chapters depict scenes as visualized in OViSE.

- Visualization of the manipulation strategy motion planner, based on the OpenInventor 3D graphics engine [29]. It visualizes the current configuration of the robot, target configurations for planning, objects known to motion planning as well as virtual walls artificially restricting the workspace of the robot.

- Dynamics simulation uses the visualization component of OpenRAVE [33], also based on OpenInventor, to show robot configuration, the environment, objects and additional information like observed object poses – in contrast to intrinsic ones. A screen shot is shown in Figure 4.16.

Perception observations and actuation execution of skills presented in Section 3.1.1 can be simulated. Both visualization systems and the execution-time decision-making system described in Chapter 3 are indifferent to real hardware or data simulations driving the skills. Given visualization systems, a human experiment supervisor can record robot behavior in the simulated case in the same way as is done during execution with the real hardware. Consequently, there are sev-

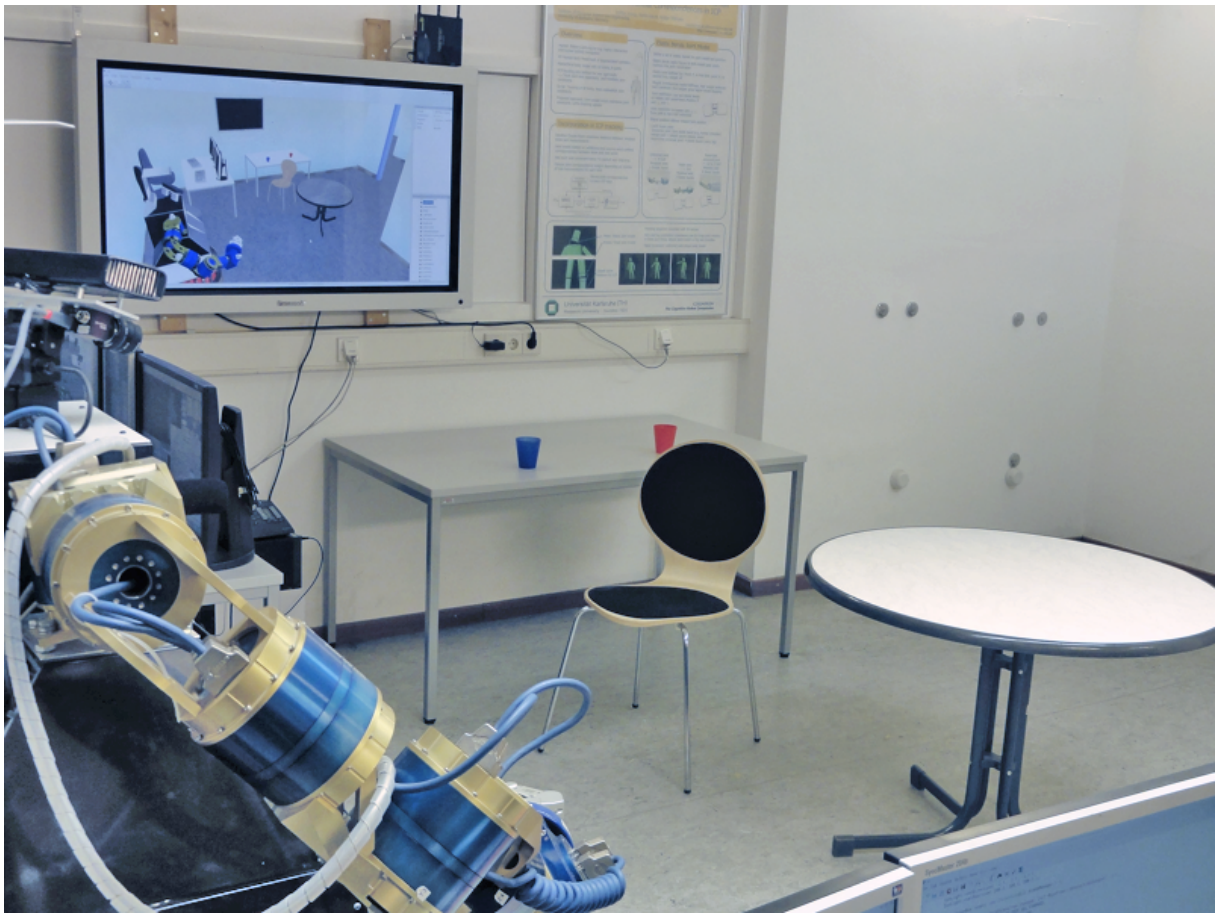


Figure 5.2.: OViSE visualization of the scene as perceived by the robot Albert by means of self-localization, furniture and small object localization as mentioned in Section 3.1.1. Objects other than those shown in Figure 5.1 have a fixed position in the lab. - [125]

eral possible experiment robot setup types between abstract virtual simulation and autonomous physical robot execution:

1. Isolated simulation. Demonstration recordings are simulated or generated from fully artificial data input. There is only simulated, partial execution or no execution at all. This includes purely abstract POMDP simulation, based on states and actions without robot skills and the belief filter involved.

2. Only autonomous simulated robot execution. A decision making model (POMDP, MDP or FSM) acquired by any means is executed using a corresponding policy by the execution-time architecture and corresponding skills on simulated perception and actuation in a virtual scene.

3. Only autonomous physical robot execution. The policy of a model acquired by any means is executed on the physical robot in a real scene.

4. Solely physical demonstration recording. Demonstrations of a mission performed by human teachers are recorded by the physical robot and processed further for specific process stage analysis. No corresponding policy is tested.

5. Physical demonstration recording and autonomous simulated robot execution. In contrast to the previous setup, execution of a policy is performed by a robot in a virtual scene.

6. Physical demonstration recording and partly autonomous physical robot, partially simulated robot execution. In this case, some skills are simulated and some are performed physically. For example, perception may be simulated and execution performed with the physical robot. In other cases, for instance, object and human localization may be performed with real sensors, while self-localization and execution are simulated.

7. Physical demonstration recording and autonomous physical robot execution. This is basically the complete setup without any skill simulation.

Each setup has different advantages and disadvantages concerning:

- Guarantee to control situations
- Speed of conducting experiments
- Focus on singular aspects, domains, skills or, on the contrary, an interplay of many components, including all of their peculiarities
- Precision to measure results of experiment
- Duration of physical robot availability

5.1.2. Service Missions Applied in Several Evaluation Experiments

For some execution-only experiments and several PbD process stages, specific missions were designed to highlight and analyze properties regarding these system parts alone. These missions are described in Sections on the respective experiments.

However, some missions were used for both integrated system-evaluation experiments, as well as detailed process stage analysis. These Comprehensive Evaluation Service Missions (CESM) are described in the following. For each mission, slightly differing models may be possible, not only concerning transitions, but also state and action spaces. Precise descriptions of actual models and their state and actions spaces are given in the respective experiment descriptions. While these missions seem simple from the point of view of scripted execution

sequences as prevalent in contemporary service robot presentations, (PO)MDP representations lead to a vastly larger number of action choices and considered transitions as they model the world in a comprehensive manner. This leads to hundreds of states and thousands of non-zero transitions. Therefore, the robot is able to consider a vast array of potential events and does not just follow a simple action sequence execution pattern in a dumb manner.

Actual layouts in the laboratory used for these CESMs were influenced by the spatial necessity to execute the mission including robot, human and furniture in areas which could be observed by the robot watching from the sidelines during demonstrations. CESMs contain proactive robot behavior as well as various primary and secondary action effect uncertainties.

CESM-1 "Serving a chair" has the robot pulling a chair towards a human who might be interested in sitting down.

There are three distinct entities in this mission: the robot, a chair and a human. In this mission, human and robot do not interact explicitly. Instead, the robot serves the role of a proactive, discreet butler and depending on pose of the human in the scene, there are differing likelihoods of a desire of the human to sit down. Hence, the robot may decide in certain situations to pull a chair present in the scene towards the human. That chair may stand in different orientations with the robot choosing in turn varying strategies to pull the chair.

As a result, the following features are involved:

1. Robot pose $f_{robot-pose}$ encompasses some relevant locations.
2. Human pose $f_{human-pose}$ reflects relevant locations and orientations.
3. Furniture state $f_{furni-state}$ contains locations, orientations and inhand of the chair.

In experiments in the laboratory, a roughly triangle-shaped layout was chosen, such that the corners approximately correspond to the initial waiting region of the robot, initial place of the chair, and the region where humans enter the scene.

Proactive robot behavior is required because the robot has to decide proactively in which situations, that is human poses, it dares to try and pull a chair. Primary stochastic action effects occur in human behavior and chair poses. Secondary stochastic action effects occur in navigation and chair manipulation actions.

CESM-2 "Serving cups" is a mission where the robot serves cups to a human on demand with a preferred cup potentially blocked by a chair.

Figure 5.3.: Illustration of CESM-2 during transport of the cup after pulling the chair out of the way. Shown clockwise: execution, visualization of feature states and demonstration. - [125]

Five distinct entities are considered in this mission: the robot, a chair, two cups and a potentially interacting human. In contrast to CESM-1 and CESM-3, there is explicit interaction as the human commands the robot to bring tea which is in the red cup. In response the robot has to check if there is a red cup available and if potentially the red cup is blocked by the chair. There are several decision options the robot can take in case the red cup is blocked by the chair and another blue cup is available:

1. Pull the chair away and then fetch the red cup.
2. Ask the human if the blue cup (tea) is also sufficient. Depending on that answer it can fetch the blue cup, fetch the red cup after pulling the chair away or fetch no tea at all.
3. Fetch the blue cup without further reassurance.
4. Fetch nothing at all and decline.

The precise choice depends on action costs and specific action effect probabilities in respective mission models. A typical setup is shown in Figure 5.3. The mission contains the following features:

1. Robot pose $f_{robot-pose}$ encompasses some relevant locations.

2. Dialog state $f_{dialog-state}$ reflects relevant dialog interaction states.

3. Furniture state $f_{furni-state}$ contains locations, orientations and inhand of the chair.

4. Small object state $f_{small-obj-state}$ reflects relevant locations and inhand of the cups.

In the laboratory, a drawn-out area layout was chosen with the human sitting at a table on the one end and cups standing on another representing a bar, with the chair potentially located in front. Proactive robot behavior is required when choosing the way a non straightforward object layout is solved after a request. Careful consideration of a variety of options is necessary. Primary stochastic action effects arise from human behavior and initial object poses, while secondary effects occur in the various navigation and manipulation actions.

CESM-3 "Helping to pull a table" is a mission in which the robot assists a human in pulling the table depending on detected activity.

Three distinct entities are relevant this mission: the robot, a table and a human. In contrast to CESM-1 with no real human-robot interaction and CESM-2 with explicit interaction, there is some implicit interaction in this mission. A human trying to pull a table is observed by the robot discreetly from nearby. Depending on the human body activity, the robot assesses if the human may succeed alone or may potentially need help. It can the approach the table and pull it together with the human. The following features compose the mission:

1. Robot pose $f_{robot-pose}$ encompasses some relevant locations.

2. Human body activity $f_{human-act}$ reflects relevant human pull-effort motions.

3. Furniture state $f_{furni-state}$ contains locations, orientations and inhand of the table.

A compact layout centered around the table was chosen in the laboratory experiments. Proactive behavior by the robot has to occur when deciding if active help is necessary or not. Primary action effect probabilities arise from human behavior, secondary effects during table manipulation actions. A typical setup is shown in Figures 1.6, 1.7.

CESM-4 "Swapping the toast" is a mission without any direct human-robot interaction in which the robots swaps a toast from one plate to another, using a spatula, as shown in Figure 5.4.

To accomplish the mission aim, the robot may fetch the target plate from another table or cupboard, in case it is not present on the main manipulation table. Furthermore, it may push the target plate as well as the spatula towards poses more suitable for grasping. Finally, the robot

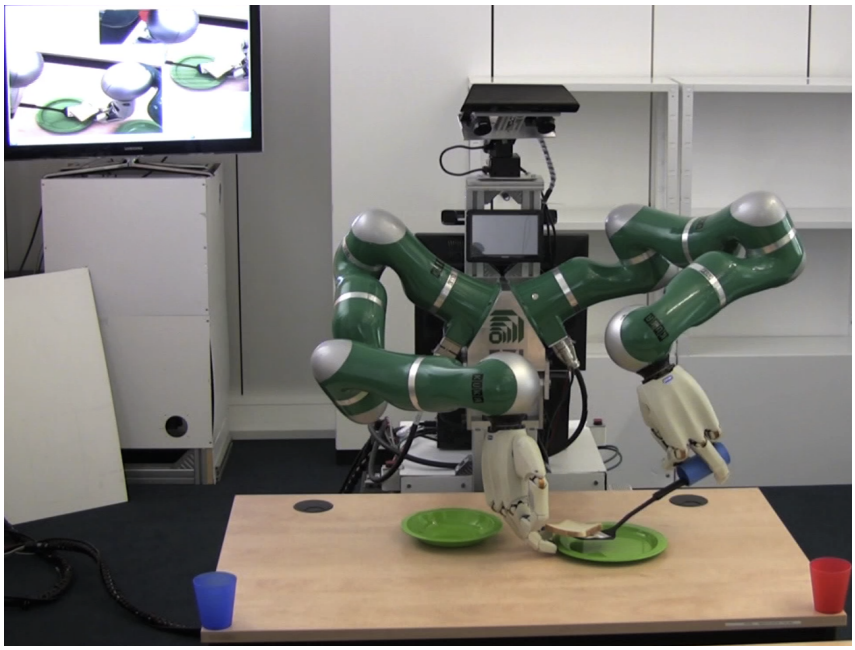


Figure 5.4.: Illustration of CESM-4 during execution by Adero. - [125]

may trigger further observation actions or call human help for rearranging the objects more favourably.

The following features compose the mission:

1. Robot pose $f_{robot-pose}$ encompasses some relevant locations.
2. Spatula state $f_{spatula-state}$ contains locations, orientations and inhand of the spatula.
3. Deep plate state $f_{deep-plate-state}$ contains locations, orientations and inhand of the deep plate.

Proactive behavior by the robot has to occur when deciding where to look for objects, if an when to push objects, look around or call for help. Primary action effect probabilities arise from initial object placement as well as placements after human intervention while secondary effects arise from navigation and manipulation action precision noise.

5.1.3. Setup Classification

In the following, experiments are classified by an *experiment setup classification scheme*, composed of: ExperimentType(Evaluation setup type, Robot setup type, Mission ID). For example, learning a model of mission "serving cups" as defined above with manually defined state (feature) and action mappings from real demonstrations and in turn executed by the real robot is

classified as: ExperimentType(3, 7, CESM-2). In turn, various properties can be measured and evaluated in each experiment setup type.

5.2. Evaluation of the Decision-Making Architecture

Within the scope of Chapter 3, the suitability of the execution-time decision-making architecture and filterPOMDP concept had to be evaluated as it is a foundation of the PMPM-PbD process.

In that evaluation [128], a comparison of different decision-making methods on top of exactly the same skills in exactly the same setup was performed. Behaviors resulting from four different decision-making methods: finite state machine (FSM), MDP, classic POMDP and filterPOMDP were measured and compared. In this evaluation, a dedicated, manually modeled Decision Architecture Evaluation Service Mission (DAESM) was used.

DAESM-1 "Bringing cups" is a mission in which the robot serves a cup when requested by the human.

There is a location where the robot may wait for a human to enter the scene. The latter may utter a spoken request to the robot to bring tea. When the robot has taken a cup, a combination of gesture and utterance indicate where to bring the tea. In this mission, object manipulation is not modeled explicitly in the (PO)MDP but subsumed in navigation actions. In the end, the cup can be brought to one of two specified locations and the robot may return to its waiting position.

This mission has the following features:

1. Robot pose $f_{robot-pose}$ encompasses some relevant locations.
2. Human body activity $f_{human-act}$ reflects the human pointing to different locations.
3. Dialog state $f_{dialog-state}$ reflects relevant dialog interaction states.

Manual modeling, with heavy use of combine states $S := \Psi(FS)$, as described in Section 3.6, reduces the state space to 28 states. The mission contains 11 actions, including the idle action. POMDP and MDP policies were generated from this model while a distinct FSM using fixed perception probability thresholds was manually designed by an expert to reflect the mission. Primary stochastic action effects result solely from human behavior.

5.2.1. Conducted Experiment

With fully autonomous, physical robot execution, but no PbD, ExperimentType(1, 3, DAESM-1) applies. Because DAESM-1 is open-ended, time is a relevant measure concerning the sum

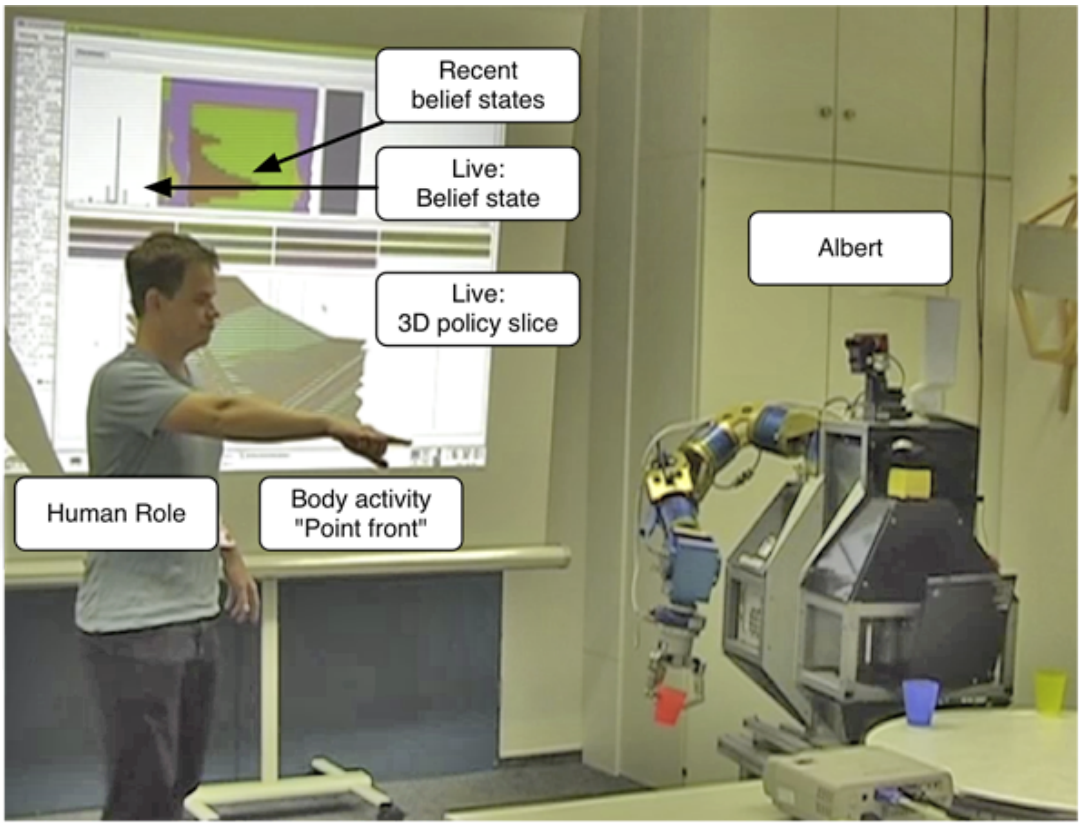


Figure 5.5.: Annotated still capture of a video of described DAESM-1 experiments with an interacting HR. Decision making runs completely autonomously on Albert, belief state and policy slice are pushed live and unidirectionally over WLAN for visualization. At this time, Albert had a Swissranger-3000 enabling full body tracking. - [128]

of total rewards. For instance, when a lot of reassuring, information gain actions are performed to determine a human intention, a cup will be brought less often, leading to a lower reward sum within a certain time span. To account for this, experiments with all four decision-making methods were given the same duration of exactly 30 minutes and the same initial situation. Furthermore, the behavior of the interacting human reflected on average the probabilities in the transition model. Finally, true requests and actual robot behavior, thus ground truth, was recorded by a human supervisor distinct from the interacting human. A picture of the setting is shown in Figure 5.5.

5.2.2. Results

Tables 5.1 show correlations between human request (Req.) and actually performed behavior (Perf.) for important parts of the mission. Desired behavior is indicated by incidence frequen-

cies on the main diagonal, while information gain action choices are shown in the reassurance column. The rest of the entries can be considered undesirable robot behavior.

Perf. / Req.	Re-assure	Fetch cup	Put to A	Put to B
Other	0	1	0	1
Fetch cup	3	4	0	0
Put to A	5	0	2	0
Put to B	7	0	0	2

(a) FSM

Perf. / Req.	Re-assure	Fetch cup	Put to A	Put to B
Other	0	0	0	3
Fetch cup	0	8	0	0
Put to A	5	0	2	1
Put to B	2	0	0	2

(b) MDP

Perf. / Req.	Re-assure	Fetch cup	Put to A	Put to B
Other	0	4	0	1
Fetch cup	1	5	0	0
Put to A	6	0	4	1
Put to B	3	0	0	3

(c) POMDP

Perf. / Req.	Re-assure	Fetch cup	Put to A	Put to B
Other	0	0	0	1
Fetch cup	1	9	0	0
Put to A	3	0	5	1
Put to B	1	0	0	2

(d) *filter*POMDP

Table 5.1.: *Results* - DAESM mission behavior applying different decision-making methods.

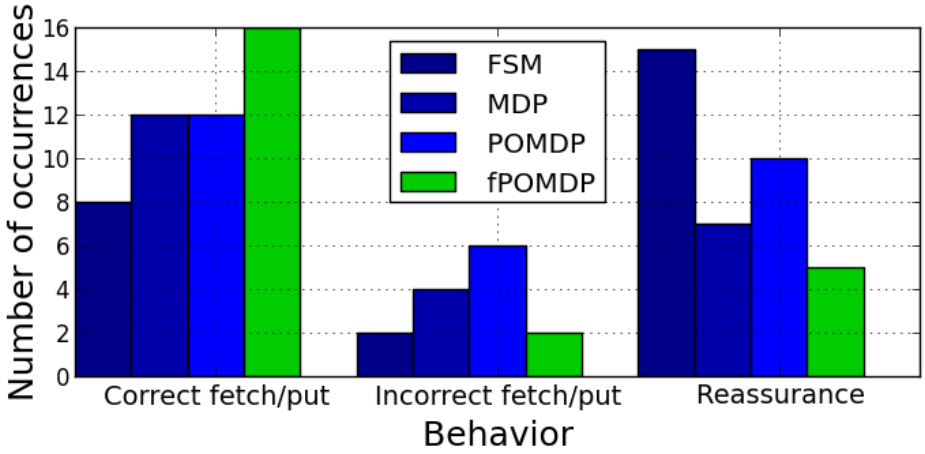
Lacking aggressive risk and opportunity assessment as in POMDPs, the FSM is conservative and annoys the human with many reassurance questions. Thus, it is not able to perform many deliveries in the given time, but does not make many big mistakes either.

The MDP is not able to handle contradicting *point* and *say* indicators well. As soon as one of the two indicators is measured - even if it is only slightly ahead of the other - the MDP will decide for the wrong location. In contrast, a POMDP still performs information gain actions in such a case. As a result, some fetch and bring runs are made when none was requested.

Using a classical POMDP, information gain actions are selected until the robot is quite sure that both indicators (*say* and *point*) refer to the same location, resulting in a better performance than the MDP. However, there is a tendency to bring the cup even when not requested, because of the reliance on a uniform distribution concerning prediction of human intention and the static observation model when computing the belief.

The *filter*POMDP shows to match the strengths of both MDP and classical POMDP. When requested to fetch the cup, action selection is based on the current speech recognition probability. In the stage of delivering the cup, the robot sometimes chooses information gain, but

not as often as the classical POMDP, as the *filter*POMDP relies on probabilities given by skill components.



Behavior \ Method	FSM	MDP	POMDP	fPOMDP
Correct fetch/put	8	12	12	16
Incorrect fetch/put	2	4	6	2
Reassurance	15	7	10	5

Table 5.2.: *Results* - DAESM mission behavior comparing execution by different decision-making methods. In these experiments, *filter*POMDP performed best.

Results shown in Table 5.2 indicate that exploiting available information about specific uncertainty of *current* perception measurements as performed by *filter*POMDP can improve service robot decision making. Static, classical POMDP observation models contain just information about *average, expected* probabilities. Using available dedicated methods computing measurement uncertainty estimates is suitable for service robots with complex perception skills like speech recognition or human activity recognition. However, the POMDP model is still a valid approximation for policy computation, reflecting expected observation and transition probabilities.

Furthermore, these results indicate that the POMDP framework is suitable for decision making in service missions of limited size at least. Therefore, POMDP models are a suitable representation, expressive and versatile – far more so than, for instance, an FSM – to be used as a target model type for PbD of mission-level decision making.

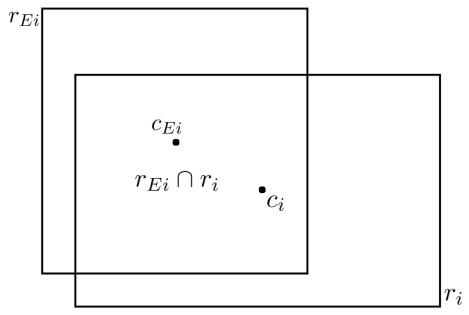
5.3. Evaluation of Feature-State Generation Based on Demonstrations

Feature mapping generation, described in detail in Section 4.2, was evaluated in experiments, based on real, physical demonstrations. Evaluation was focussed on stage-specific aspects within the scope of CESMs [147], which is therefore of type: ExperimentType(2, 4, CESM).

Automatically generated feature mapping of continuous environment properties onto symbolic, abstract states was to be compared against mapping functions previously hand-modeled by an expert. A set of multiple demonstrations FD of each CESM-1 and CESM-2 was used, with some artificial subsets SD1-SD2 for CESM-1 and SD1-SD3 for CESM-2. In the following only $f_{robot-pose}$ is discussed. Three basic stage aspects were to be evaluated:

1. Clustering performance for a given cluster number k.
2. Stability and relevance of generated regions.
3. Determining cluster number k and therefore feature state space size, autonomously.

Results in Table 5.3 show that unsupervised clustering of pose information, in the shown case robot role pose, delivers results corresponding well with manual expert partitioning. Typical potential errors are shown in Figure 5.7, but their occurrence frequencies are low in the experiments, as indicated by the results in Table 5.4. Results of EM tail significantly behind k-means and DBscan for fixed k. Automatically determining k for k-means and DBscan is discussed below. Automatically generated regions representing discretization functions f_{g1} autonomously generated from demonstrations were evaluated using common covered area metrics as illustrated in Figure 5.6. High area similarity for regions generated from different demonstration data sets indicate stability and therefore high-quality of abstract spatial feature states as shown in Table 5.5.



$$U(r_{Ei}, r_i) = \frac{A(r_{Ei} \cap r_i)}{A(r_{Ei} \cup r_i)} \quad [5.1]$$

$$U_i(r_{Ei}, r_i) = \frac{A(r_{Ei} \cap r_i)}{A(r_{Ei})} \quad [5.2]$$

$$D(r_{Ei}, r_i) = A(r_{Ei}) - A(r_i) \quad [5.3]$$

$$D_E(r_{Ei}, r_i) = \frac{A(r_{Ei}) - A(r_i)}{f(r_{Ei})} \quad [5.4]$$

Figure 5.6.: Illustration of region similarity measures used to compare automatically generated regions. - [147]

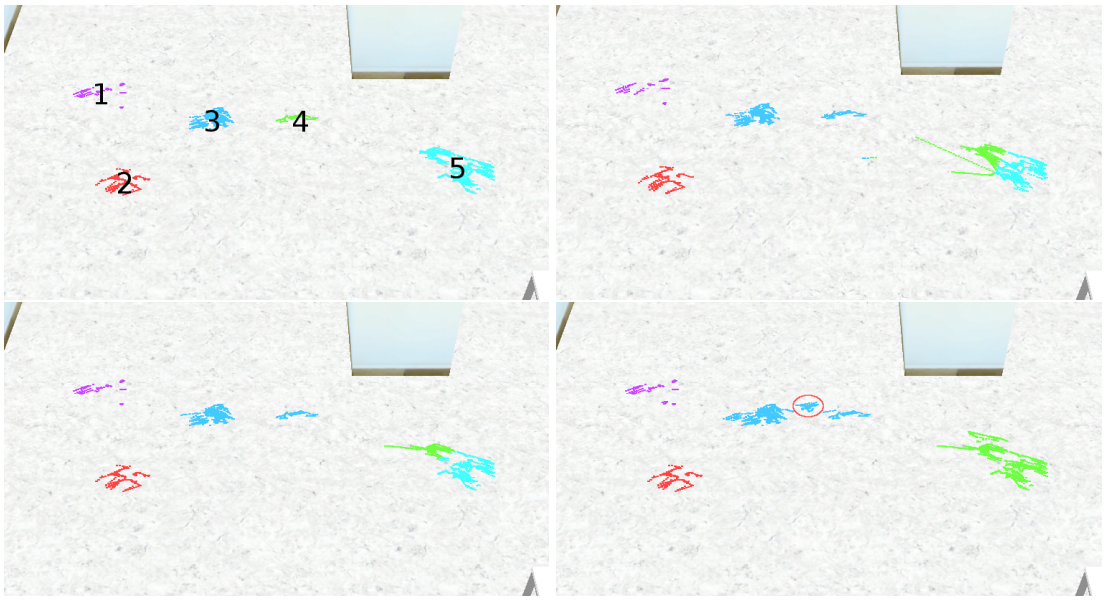
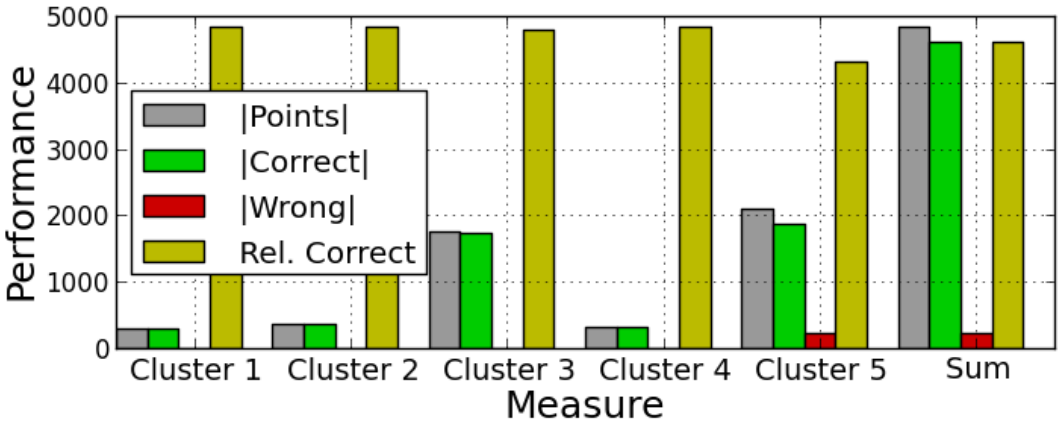
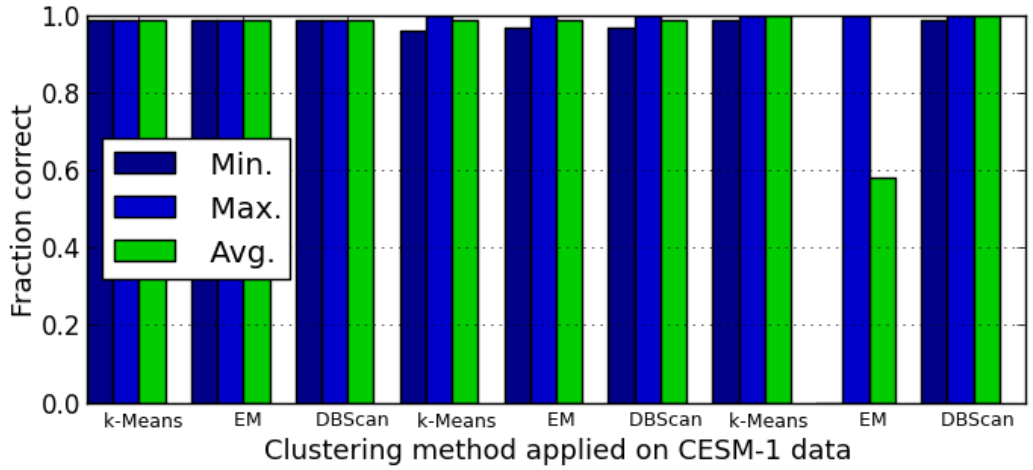


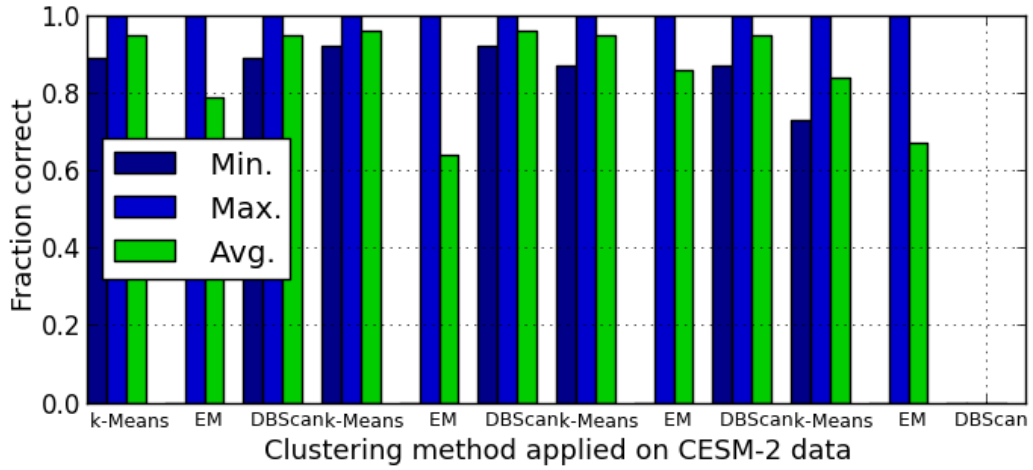
Figure 5.7.: Illustration of some typical errors occurring when clustering CESM-2 demonstrations. Clockwise from top left: expert labels, k-means, DBscan, EM. - [147]



Cluster \ Measure	|Points|	|Correct|	|Wrong|	Rel. Correct
Cluster 1	299	299	0	1.00
Cluster 2	374	374	0	1.00
Cluster 3	1751	1745	6	0.99
Cluster 4	321	321	0	1.00
Cluster 5	2099	1877	222	0.89
Sum	4844	4616	228	0.95

Table 5.3.: *Results* - Detailed error percentages when clustering mission CESM-2 with motion filtering and k-means for $f_{robot-pose}$ with $k = 5$ on the full data set FD.





k	Data	Method	Min.	Max.	Avg.
3	FD	k-Means	0.99	0.99	0.99
		EM	0.99	0.99	0.99
		DBScan	0.99	0.99	0.99
	SD1	k-Means	0.96	1.00	0.99
		EM	0.97	1.00	0.99
		DBScan	0.97	1.00	0.99
	SD2	k-Means	0.99	1.00	1.00
		EM	0.00	1.00	0.58
		DBScan	0.99	1.00	1.00

(a) CESM-1

k	Data	Method	Min.	Max.	Avg.
5	FD	k-Means	0.89	1.00	0.95
		EM	0.00	1.00	0.79
		DBScan	0.89	1.00	0.95
	SD1	k-Means	0.92	1.00	0.96
		EM	0.00	1.00	0.64
		DBScan	0.92	1.00	0.96
	SD2	k-Means	0.87	1.00	0.95
		EM	0.00	1.00	0.86
		DBScan	0.87	1.00	0.95
	SD3	k-Means	0.73	1.00	0.84
		EM	0.00	1.00	0.67
		DBScan	-	-	-

(b) CESM-2

Table 5.4.: *Results* - Relative matching of all clustering tests on each data set for $f_{robot-pose}$, relative to expert ground truth.

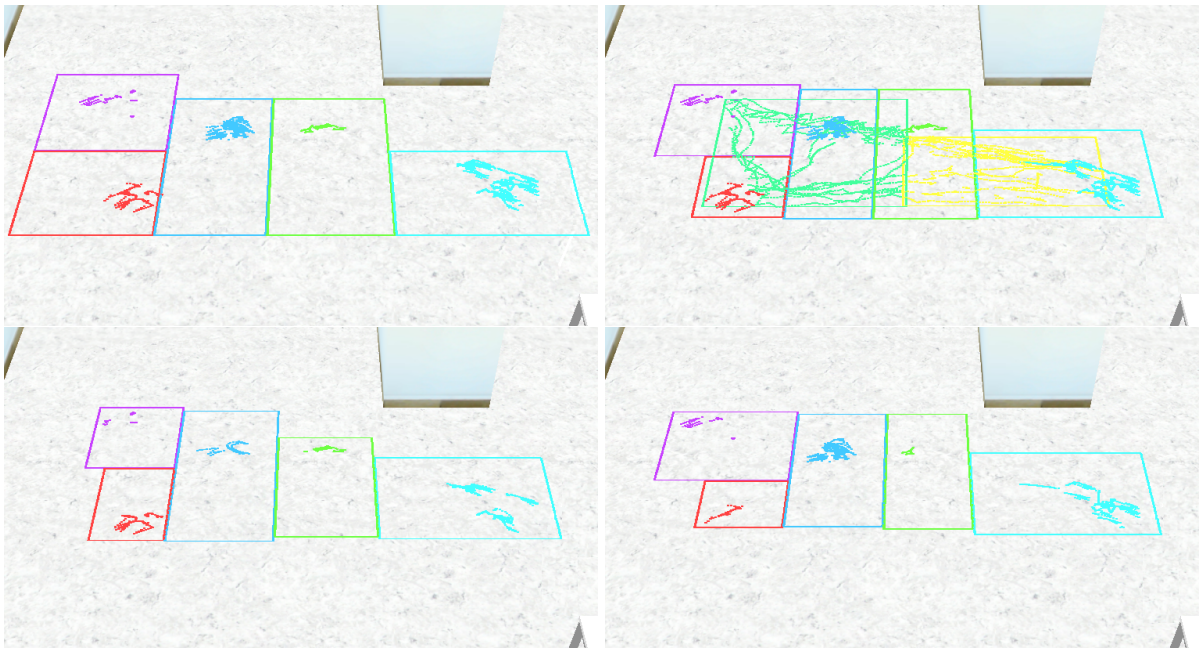


Figure 5.8.: Illustration of generated regions $f_{robot-pose}$ for data sets of CESM-2. Clockwise from top left: human expert, FD with transition regions & raw points, SD1, SD2. - [147]

i \ j	FD	SD1	SD2
FD	1.00	0.68	0.64
SD1	0.68	1.00	0.63
SD2	0.64	0.63	1.00

(a) $U(R_i,R_j)$ CESM-1

i \ j	FD	SD1	SD2
FD	1.00	0.70	0.75
SD1	0.96	1.00	0.88
SD2	0.80	0.68	1.00

(b) $U_j(R_i,R_j)$ CESM-1

i \ j	FD	SD1	SD2
FD	1.00	0.76	0.84
SD1	0.76	1.00	0.69
SD2	0.84	0.69	1.00

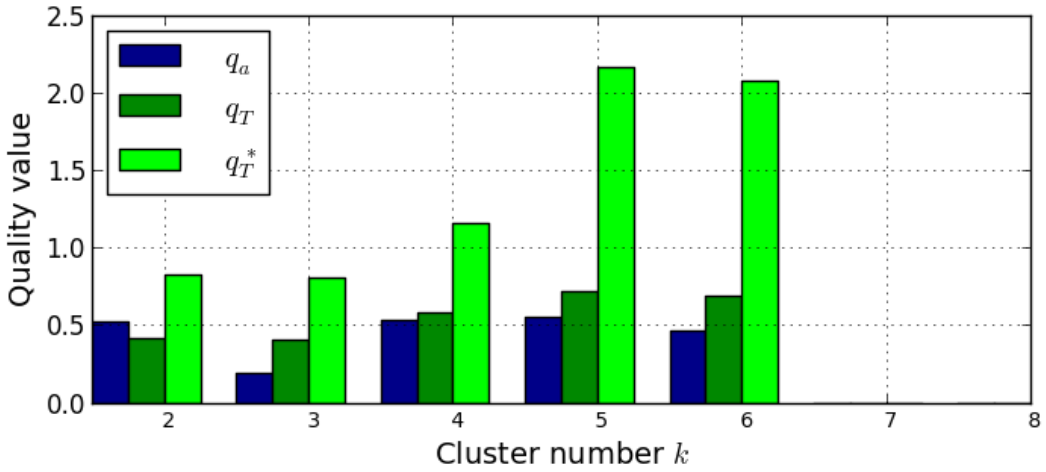
(c) $U(R_i,R_j)$ CESM-2

i \ j	FD	SD1	SD2
FD	1.00	0.81	0.87
SD1	0.92	1.00	0.83
SD2	0.96	0.81	1.00

(d) $U_j(R_i,R_j)$ CESM-2

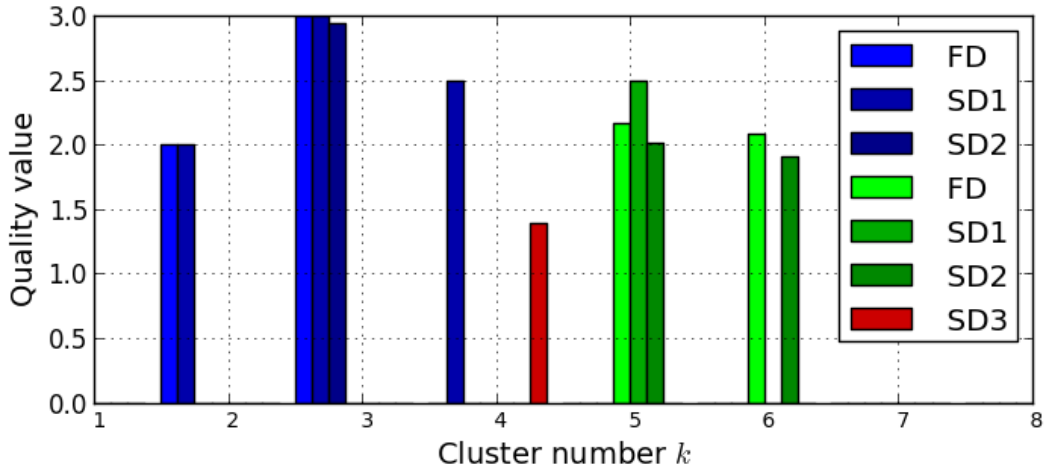
Table 5.5.: *Results* - Shared region area metrics $U(R_i,R_j)$ and $U_j(R_i,R_j)$ for regions generated from different datasets for $f_{robot-pose}$, indicating stability of automatically generated f_{g1}.

Finally, evaluation had to determine if unsupervised identification of cluster number k, and therefore feature state (category) number led to results representing the mission correctly. It is the most critical stage in the process. Results in Table 5.6 and Table 5.7 show that this is the case in these experiments for both k-means and DBscan for all but one data set. Ground truth - as modeled by an expert and without considering the catch-all feature state "Outer" is as follows: CESM-1 contains 3 regions and CESM-2 contains 5 regions.



Measure \ k	2	3	4	**5**	6	7	8
V-sv	5	7	3	1	2	4	6
SD	6	5	2	1	3	4	7
XB	5	7	1	2	3	4	6
DB	2	7	3	1	4	5	6
Vote	3	7	2	1	4	5	6
q_a	0.52	0.19	0.53	0.55	0.46	-	-
q_T	0.42	0.41	0.58	0.72	0.69	-	-
q_T^*	0.83	0.81	1.16	**2.17**	2.08	-	-

Table 5.6.: *Results* - Quality measures used to infer cluster k automatically, resulting from k-means clustering based on FD of CESM-2. Cluster number 5 is best, which is the ground truth.



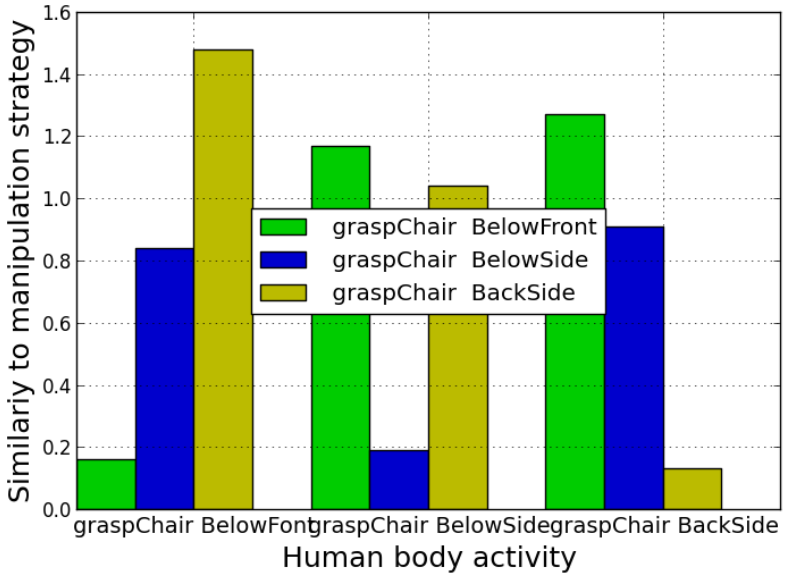
Mission	Data \ k	1	2	3	4	5	6	7	8
CESM-1	FD	-	2.00	**3.00**	-	-	-	-	-
	SD1	-	2.00	**3.00**	2.50	-	-	-	-
	SD2	-	-	**2.94**	-	-	-	-	-
CESM-2	FD	-	-	-	-	**2.17**	2.08	-	-
	SD1	-	-	-	-	**2.50**	-	-	-
	SD2	-	-	-	-	**2.01**	1.91	-	-
	SD3	-	-	-	1.39	**-**	-	-	-

Table 5.7.: *Results* - Quality measure q_T^* used to infer cluster k automatically, resulting from DBscan. Only for CESM-2:SD3, resulting k does not match ground truth.

This means that the system is able to create a state space and corresponding environment perception mapping functions from demonstrations usable for abstract mission-level planning, in an unsupervised manner. The combination performing best for this purpose is a motion based filtering in combination with DBScan and multiple cluster number k-means clustering. The optimal cluster number k selected in turn by clustering quality metrics. EM-clustering shows inferior results. The output is a set of functions tailored for a specific mission mapping continuous pose spaces of robot, humans and objects onto a discrete POMDP state space. Distinct poses corresponding to the same abstract situation are clustered in a single state. More data and further discussion regarding these state-mapping experiments can be found in [147].

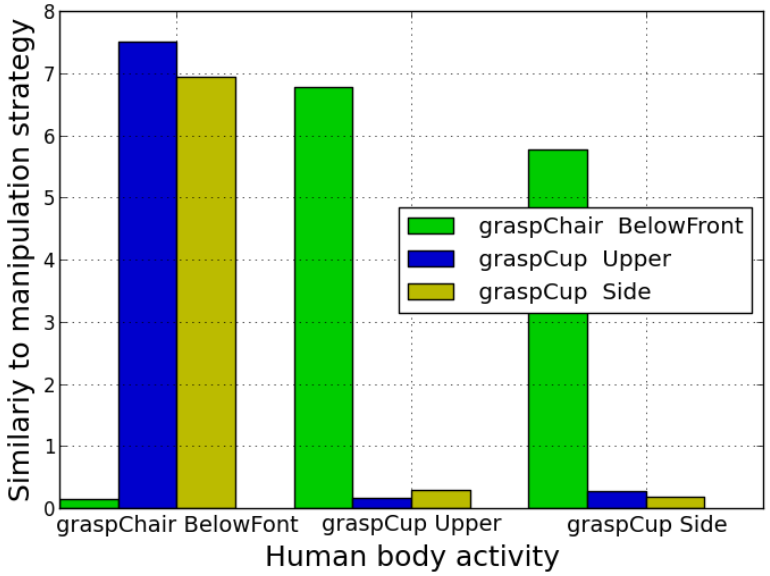
5.4. Evaluation of Manipulation Action Mapping

Action mapping, as discussed in Section 4.3 was evaluated on manipulation actions a_{mp} as occurring in CESMs. Real, physical mission demonstrations were the input on the one hand and real, physical manipulation strategy demonstrations on the other. Similarity measures and correct mapping was the focus of evaluation [42]. Therefore, these experiments are of type: ExperimentType(2, 4, CESM). Both CESM-1 and CESM-2 contain three different manipulation activities. In the correspondence similarity metric, lower values indicate greater similarity. In tables below, both activities and strategies are labeled with symbols referencing the intended strategy. It should be noted, however, that in practice symbols are arbitrary and the system has to find the correspondence in an unsupervised manner. Tables 5.8 and 5.9 show distance- and covariance-weighted similarity measures, which are the lower the more similar they are. Strategy data of CESM-1 is shown in Figure 4.10.



Activity \ Strategy	graspChair BelowFront	graspChair BelowSide	graspChair BackSide
graspChair BelowFont	0.16	0.84	1.48
graspChair BelowSide	1.17	0.19	1.04
graspChair BackSide	1.27	0.91	0.13

Table 5.8.: *Results* - Manipulation action mapping similarity measures for CESM-1. The lower the more similar.



Activity \ Strategy	graspChair BelowFront	graspCup Upper	graspCup Side
graspChair BelowFont	0.14	7.51	6.94
graspCup Upper	6.78	0.16	0.30
graspCup Side	5.77	0.27	0.18

Table 5.9.: *Results* - Manipulation action mapping similarity measures for CESM-2. The lower the more similar.

In these experiments, similarity was always greatest on the diagonal, indicating each activity mapped to the best fitting strategy. It can be seen that similarities are less between more similar motions while difference is pronounced when manipulating differing objects. It can be concluded that with these quite similar manipulation actions, differences between the correct and the second similar action are pronounced enough to guarantee robust mapping.

5.5. Evaluation of Segmentation and Preliminary Model Generation

The focussed evaluation of segmentation, presented in Section 4.4, and preliminary model generation, as discussed in Section 4.6, was performed as part of experiments conducted with some dedicated Segmentation and Preliminary model generation Evaluation Service Missions (SPESM).

SPESM-1 "Bringing snacks" is a mission where the robot brings either a pringles can or a cup of tea from a bar table to a table and a seated interacting human as shown in Figure 5.10. The human interacts with the robot by means of spoken dialog, while the robot may commute between that table and another bar table. At different positions on the bar table, a cup or can of pringles may be located. Using spoken dialog, "tea" or "an appetizer" may be requested by the human. In turn, the robot fetches the object implicitly related to the request.

This mission has the following features:

1. Robot pose $f_{robot-pose}$ encompasses some relevant locations.
2. Dialog state $f_{dialog-state}$ reflects relevant dialog interaction states.
3. Small object state $f_{small-obj-state}$ contains relevant locations and inhand of the cup and pringles can.

SPESM-2 "Putting away garbage" is a mission in which the robot throws a cup or pringles can, directly handed over, into the trash. An interacting human can approach the robot in open space and present an object to it. In turn, the robot takes the object and throws it into the trash.

This mission has the following features:

1. Robot pose $f_{robot-pose}$ encompasses some relevant locations.
2. Human body activity $f_{human-act}$ reflects wether the human reaches out or not.
3. Small object state $f_{small-obj-state}$ contains relevant locations and inhand of the cup and pringles can.

SPESM-3 "Handing over snacks" is a mission in which the robot fetches a cup or pringles can from a bar table and directly hands it over to a human, as shown in Figure 5.11. Similar to SPESM-1, an interacting human may request "tea" or "an appetizer" by means of spoken dialog. In turn, the robot fetches an implicitly associated object and returns to the human. Next, object transfer between robot and human is managed by the robot by both spoken dialog and monitoring of human body activity.

This mission has the following features:

1. Robot pose $f_{robot-pose}$ encompasses some relevant locations.
2. Human body activity $f_{human-act}$ reflects the human reaching out or not.
3. Dialog state $f_{dialog-state}$ reflects relevant dialog interaction states.

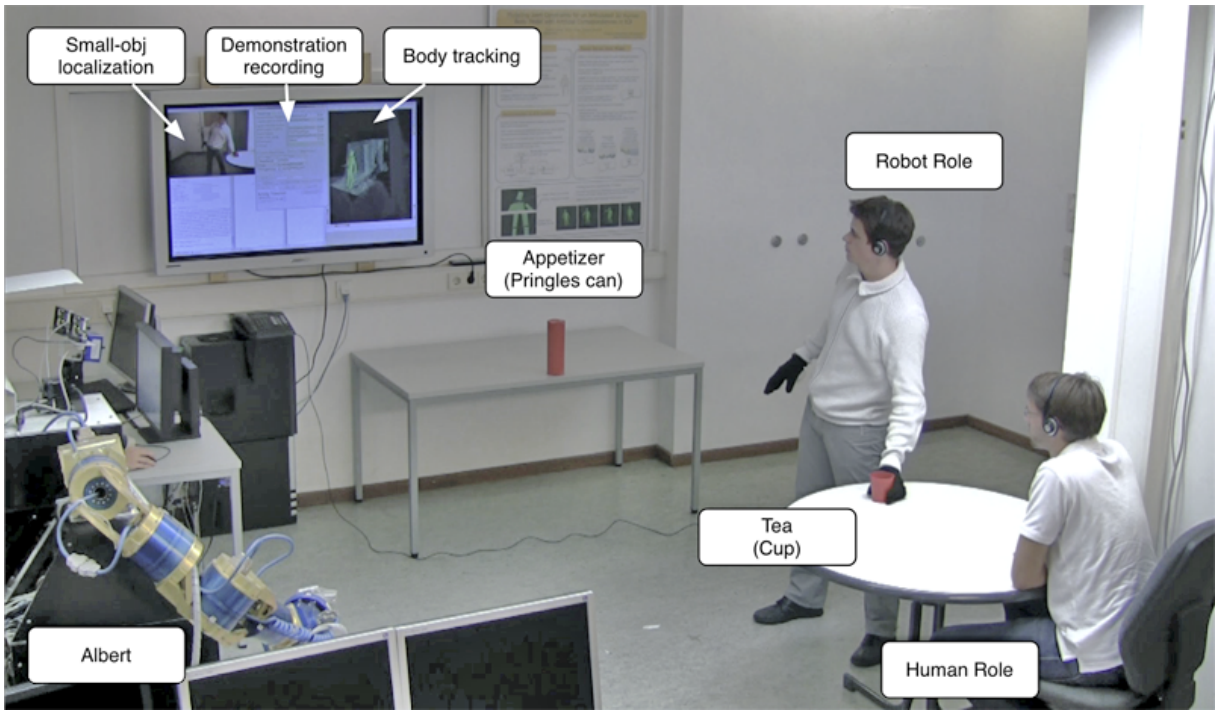


Figure 5.9.: Annotated still capture of a video of described experiments with RR and HR. It shows a demonstration recording of SPESM-1 by Albert, following RR actively with its cameras. At the time of this experiment, Kinect was not released yet and Albert used a Swissranger-3000 together with the VooDoo body tracking system. Headsets used for dialog recording can be seen, and gloves were worn to improve small red object in-hand localization. - [132]

4. Small object state $f_{small-obj-state}$ contains relevant locations and inhand of the cup and pringles can.

In SPESM-1, SPESM-2 and SPESM-3, primary stochastic action effects result from human behavior and initial object placing.

5.5.1. Experiments Focussed on Preliminary Model Generation

A set of experiments involving physical demonstrations and physical execution, focussed on preliminary model mapping [132]. State and action grounding was determined manually, no generalization or sophisticated refinement was performed and essential robot specific error states and transitions as well as action costs were added from fixed background knowledge. Consequently, these experiments can be classified as ExperimentType(3, 7, SPESM-1), ExperimentType(3, 7, SPESM-2), ExperimentType(3, 7, SPESM-3).

Ten physical demonstrations, as depicted in Figure 5.9, were performed for each mission with varying courses of events. The inertial temporal segmentation filter time utilized was set to the

default of 1 second. Primary action effect frequencies were recorded by a human supervisor to be reproduced during the robot mission-execution experiment stage.

Model components S, A, M, T, R, O were automatically generated for each mission from recorded traces, resulting in:

Mission \ Elements	$\|F_1\|$	$\|F_2\|$	$\|F_3\|$	$\|F_4\|$	|S|	|A|	|M|
SPESM-1	5	5	20	-	500	12	18
SPESM-2	5	2	5	-	50	6	12
SPESM-3	5	2	7	5	350	14	19

Because of interdependencies of feature state spaces in the transition model, a fully factored representation is not possible, leading to relatively large state spaces considering the limited scope of given missions. Some states could be combined, but combining states is not defined in a sound way, as discussed in Section 3.6, requiring further background knowledge, that does not exist in the given setup. Value function and policy Γ computation based on the largest model generated for SPESM-1 took 2 minutes to reach at least 99% optimal utility approximation: $U^* = U^{policy}(\Gamma) + \varepsilon, \varepsilon < 0.01$ with SARSOP on an Intel Q9550 CPU with 4 GB RAM. Therefore, policy computation effort was negligible compared to demonstration time in the PbD process chain.

During the execution stage of the experiment, POMDP models and policies acquired by simple PbD were evaluated against manually designed FSMs in a manner similar to the experiment described in Section 5.2. For each of the SPESM missions, an FSM was designed by an expert applying manual probability thresholds for skill measurement probabilities and basic actuation skills. Exactly the same set of basic actions was available to be selected by POMDP and FSM. Interacting humans had to behave according to primary action-effect probabilities defined during demonstrations, in fact synchronizing HR behavior frequencies between human demonstrations and robot execution.

A difference from the experiment described in Section 5.2 was that each mission had not been performed in an open-ended way. Instead, mission runs were terminated after major goals had been reached or they had failed beyond recovery, and the duration was recorded for each run. The reason was to avoid having to restart a whole 30-minute experiment after some fundamental robot-motor hardware failures, which were not relevant for evaluation but occasionally occur. Execution runs with such failures were discarded for evaluation.

For each method and mission, 10 runs were performed successively with a still shot of a run shown in Figure 5.10. The following table shows minimum, average, and maximum execution

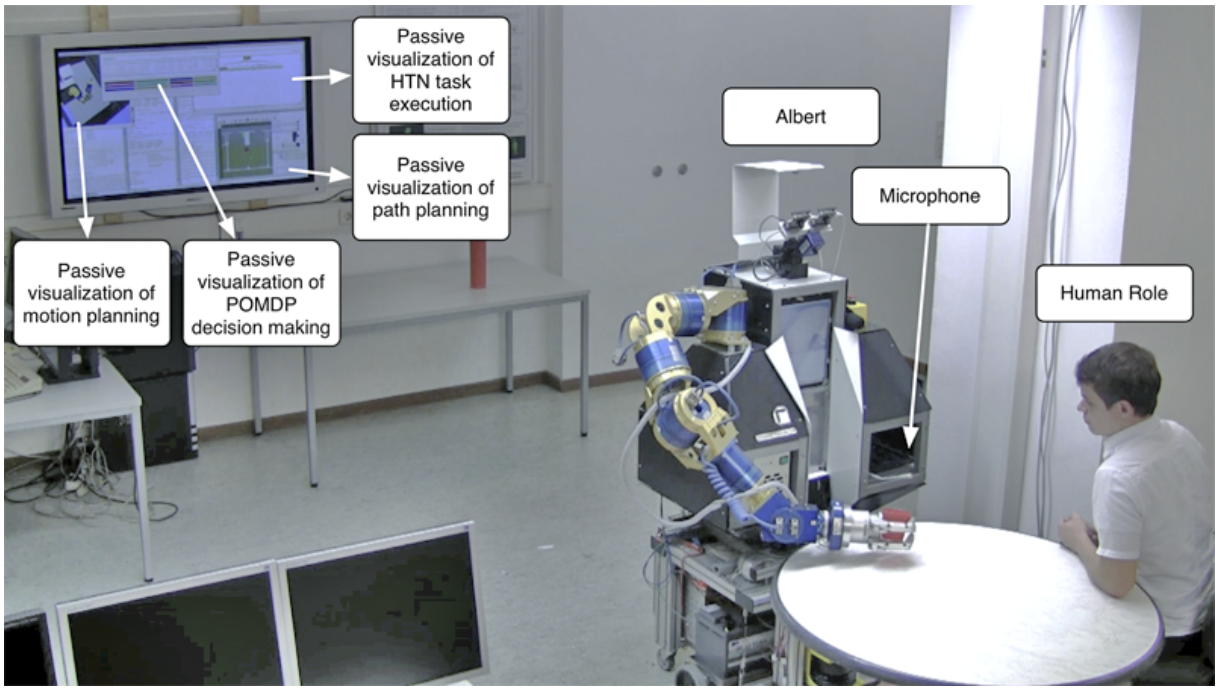
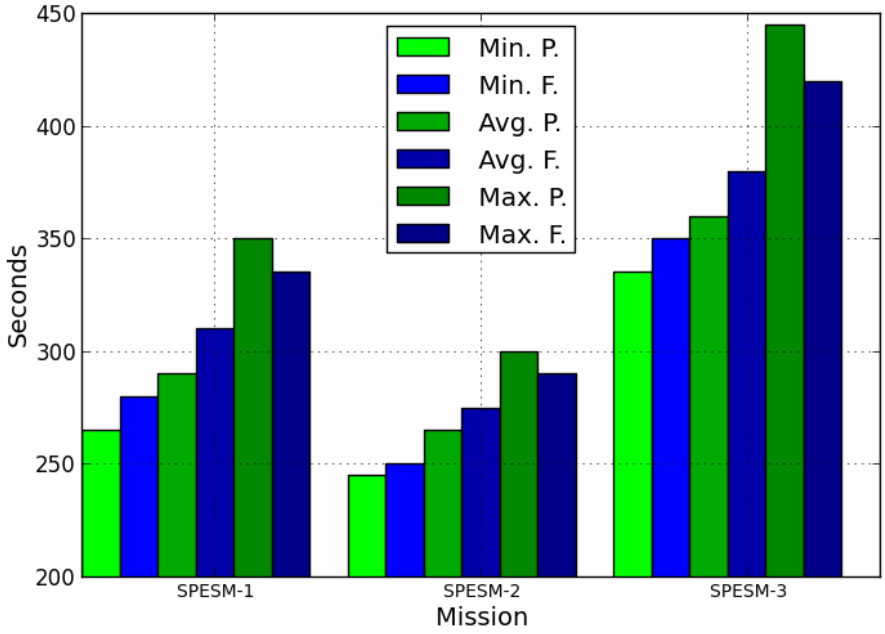


Figure 5.10.: Annotated still capture of a video of described experiments with an interacting HR. It shows an execution of SPESM-1. All planning and decision-making components ran autonomously on board of Albert and visualization of autonomous components, shown on the panel in the background is pushed unidirectionally from Albert by WLAN. - [132]

durations for both generated POMDP (P) and hand-built FSM (F) in minutes, as well as the number of major mission failures.

In these experiments, action selection based on a POMDP acquired by PbD was superior to handcrafted FSM in more complex missions 1 and 3, including spoken dialog as shown in Tables 5.10 and 5.11. Spoken dialog with a distance microphone on board of the robot was especially subject to noise and imprecise measurement in the given setting. In SPESM-2 without spoken dialog, the POMDP had at least no major disadvantage compared to the FSM.

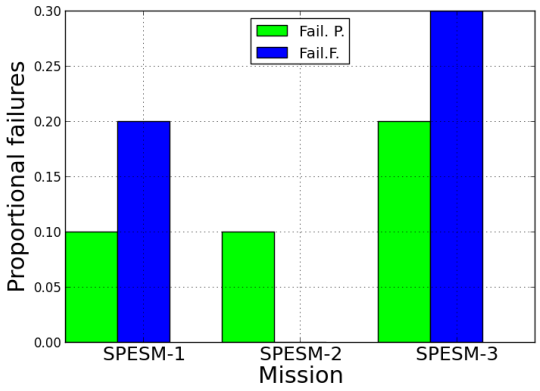
Concerning total execution durations it should be noted that the arm was driven at most at 30% of its maximum speed, while the mobile platform was not driven faster than 20% of its maximum speed for safety reasons. Therefore, human demonstrations took far less time than robot execution. A further major difference was the time taken to manually build the FSMs on the one hand and naturally learn the POMDP models from demonstrations on the other. As this aspect was thought not to be measurable easily in a sound way, only a rough estimate can be given here: an FSM building expert took more than twenty times longer to build the FSMs than was required for recording demonstrations.



Metric, Method / Mission	Min. P.	Min. F.	Avg. P.	Avg. F.	Max. P.	Max. F.
SPESM-1	4:25	4:40	4:50	5:10	5:50	5:35
SPESM-2	4:05	4:10	4:25	4:35	5:00	4:50
SPESM-3	5:35	5:50	6:00	6:20	7:25	7:00

Table 5.10.: *Results* - SPESM mission duration metrics comparing execution by POMDPs generated using simple PbD against expert-devised FSMs. The POMDP was better on average, yet had some stronger outliers.

It should be noted that apart from the obvious setting variations between differing demonstrations of a specific mission, the transition probabilities implicitly encode information about the locations where objects can generally be encountered, that they can move with the robot when picked, where they can be placed and also how a dialog can develop. The reward model encodes, for instance that the object *Cup* should be placed at a certain place, when *bring me tea* was requested during the dialog. This information is exclusively learnt by the presented process - there is no connection between the locatable object *Cup* and the utterance *bring me tea* in the background knowledge, as well as no information about where it can be found and where it should be brought.



Mission \ Metric, Method	Fail. P.	Fail. F.
SPESM-1	1/10	2/10
SPESM-2	1/10	0/10
SPESM-3	2/10	3/10

Table 5.11.: *Results* - SPESM mission failures comparing execution by POMDPs generated using simple PbD against expert-devised FSMs.

Figure 5.11.: Still capture of a video of experiments regarding SPESM-3 with demonstration on the left and execution on the right. - [131]

5.6. Evaluation of Generating and Verifying Transition Hypotheses

As part of interactive transition model space exploration, which consists of both generalization, introduced in Section 4.7 and verification, presented in Section 4.8, experiments in both physical settings and extensive analysis of tailored, more artificial models have been performed.

5.6.1. Demonstration, Generation, Request and Execution Experiments

The experiment shown in Figure 5.12, including physical demonstrations and physical execution of CESM-1, was performed with the robot Albert: ExperimentType(4, 7, CESM-1) [134]. Two different instances of the mission were investigated:

1. The first mission instance was to focus on generalizing transitions, originating in the exact human pose, location and orientation, reflecting human intention to sit down on the chair. Generalization therefore had to take place in $f_{human-pose}$.

2. In the second mission instance, generalization in $f_{furni-state}$ had to account for two options (two different a_{mp}^{mst}) to grasp the chair, depending on different positions and orientations.

In all cases, feature and action mappings were designed manually; chair location and orientation was split into two different features $F_{3.loc}$ and $F_{3.or}$. Experiment design was as follows: first, for each instance, a certain set of "real" mission dynamics concerning primary stochastic action effects was devised. Then, as in the experiment discussed in Section 5.5, twenty demonstration sequences were performed for each instance, however omitting some courses of events devised for the "real" mission dynamics. Generalization then had to find candidates covering these courses of events and generate respective demonstration requests. Further demonstrations keeping tightly to these requests could then enhance the model. Finally, a policy was computed from the resulting model and executed by the robot. Primary "real" dynamics of the mission were applied during execution. Again, hand-modeled FSMs containing handling of all "real" dynamics performing different mission instances were applied for comparison. Evaluation should show how well generalization and requests were able to enhance the model from incomplete demonstrations to cover aspects well enough to compete with the FSM, which had the advantage of manual tailoring.

Mission \ Elements	$\|F_1\|$	$\|F_2\|$	$\|F_{3.loc}\|$	$\|F_{3.or}\|$	|S|	|A|
CESM-1/1	4	9	4	3	432	8
CESM-1/2	4	5	5	5	400	8

A total of 1157 generalizations of non-observed effects (v_g) were made, with 18 considered crucial by a human-expert analysis to scale to non-demonstrated aspects of the mission. Most non-crucial v_g affected the *idle* action. A total of twenty v_g were rated above the general threshold of which seven had to be rejected as invalid. The rest could be confirmed with demonstrations, again a different number of times to correct the estimated frequencies to the desired mission values.

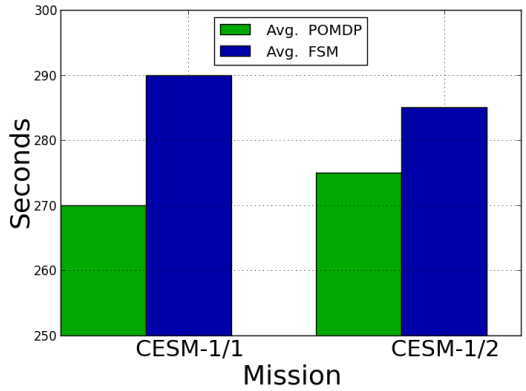
A total of 1273 generalizations of non-observed effects (v_g) were made, with 14 being considered crucial by a human expert analysis. Most of the other v_g affected the *idle* action. A total of 22 v_g were rated above the general threshold and could be confirmed. These examples show the necessity of relevance estimation, and the first one also highlights the need for the binary tree request generation to reject false hypotheses.

Execution time (average duration of one main task performance within the mission) and failure frequency can be considered the primary performance criteria of the decision-making

Figure 5.12.: Still capture of a video of experiments regarding CESM-1 with demonstration on the left and execution on the right. - [134]

system. Both missions were executed by the robot (and interacting human) and measured by a supervisor ten times for each controller. It included courses of events which were not present in the initial demonstrations, but only after generalization and request generation.

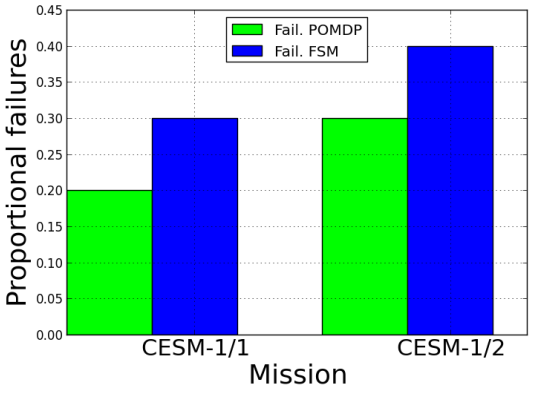
For each method and mission, ten runs were performed successively. The following table shows average execution duration for both generated POMDP (P) and hand-built FSM (F) in minutes as well as number of major mission failures.



Metric, Method / Mission	Avg. POMDP	Avg. FSM
CESM-1/1	4:30 min	4:50 min
CESM-1/2	4:35 min	4:45 min

Table 5.12.: *Results* - CESM-1 mission durations comparing execution by POMDPs generated using PbD against expert-devised FSMs.

Failures included pulling the chair when the human had no interest and failing to pull the chair. Delays occurred when taking some time to interpret the human pose correctly, looking again for the chair or trying again to pull the chair.



Metric, Method / Mission	Fail. POMDP	Fail. FSM
CESM-1/1	2/10	3/10
CESM-1/2	3/10	4/10

Table 5.13.: *Results* - CESM-1 mission failures comparing execution by POMDPs generated using PbD against expert-devised FSMs.

The results show that the learned POMDP was able to match and even slightly surpass the performance of the hand-tailored FSM. The learned POMDP can achieve good performance from a limited number of natural demonstrations of non-technical experts.

5.6.2. Focussed Generalization Evaluation

Focussed evaluation of transition generalization was performed with simulated demonstration observations [117] and is of type ExperimentType(2, 1+2, FGESM). These simulated demonstration sequences had exactly the same structure as real demonstrations after segmentation. Simulation was chosen, as there is no fundamental difference between taking real demonstrations and simulated ones in exactly the same representation for focussed evaluation on the abstract level. Focussed evaluation had to assess if transition generalization can indeed improve policy performance by estimating omitted demonstration aspects.

To achieve this, two sets of demonstrations were taken: a "perfect" reference set representing ground truth transitions and an imperfect set, with some demonstrations omitted. First, a model and policy was computed from the set of reference demonstrations, resulting in a reference policy. Then, a model and policy was computed from the imperfect set without any generalization. Next, models were generated from the imperfect set with subsequent generalization using different parameters. For all policies, multiple simulated executions were performed and accumulated rewards assessed. Detailed results are discussed below. A special mission closely related to missions used in real experiments was used for this purpose.

FGESM-1 "Bringing tea" is a mission where the robot can bring a red cup of tea, but not a green one, after being requested by dialog. It is very similar to SPESM-1, having the same features. There are three $f_{robot-pose}$ regions where the robot may encounter a cup of tea. It can fetch a red cup, but not a green cup. An interacting human can request the red cup.

This mission has the following features:

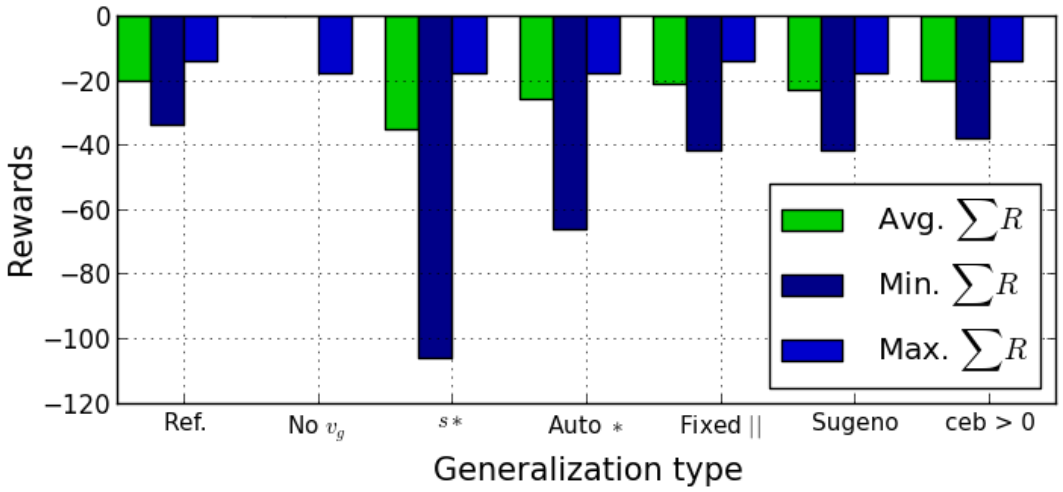
1. Robot pose $f_{robot-pose}$ contains some relevant locations.
2. Dialog state $f_{dialog-state}$ contains dialog interaction states.
3. Small object state $f_{small-obj-state}$ contains relevant locations and inhand of the cups.

In the given evaluation, all demonstrations led to the following set:

Mission \ Elements	$\|F_1\|$	$\|F_2\|$	$\|F_3\|$	|S|	|A|
FGESM-1	5	3	4	60	7

Given different parameters for generalization, Table 5.14 shows performance of the reference policy, the original policy with omitted demonstrations and policies resulting from generalized models using different parameter sets. The following different parameter sets are shown:

1. "Ref.": a policy generated from the reference demonstrations and respective model.
2. "No v_g": a policy generated from demonstration sets with omitted demonstrations and resulting model. All following models are based on this demonstration set.
3. "$s*$": generalization mask κ_{scope} as described in Section 4.7.2 is defined over all of S with origin $s = *$.
4. "Auto $*$": generalization mask κ_{scope} is determined by "by analyzing onto which features actions have an effect" as explained in Section 4.7.2. All following models use this mask.
5. "Fixed $\|\|\|$": not encountered feature transitions are assigned a fixed likelihood "Trans. conf.", no relationship metric is computed for these.
6. "Sugeno": a confidence based on transition relationships using a scale factor α_{gc} as discussed in Section 4.7.3, is used.
7. "ceb > 0: the same as before, except certainty bias is considered with scale factor γ_{gc}, as explained in Section 4.7.3.



Value / Type	κ $*$	Trans. conf.	α_{gc}	γ_{gc}	$\lvert v_g \rvert$	Avg. $\lvert Q \rvert$	Min $\lvert Q \rvert$	Max. $\lvert Q \rvert$	Avg. $\sum R$	Min. $\sum R$	Max. $\sum R$
Ref.	Reference model				-	8.53	7	12	-20.13	-34	-14
No v_g	No generalization				-	∞	8	∞	-∞	-∞	-18
$s*$	s	-	-	-	59	12.27	8	30	-35.07	-106	-18
Auto $*$	s,s'	-	-	-	137	10	8	20	-26	-66	-18
Fixed $\lvert\lvert\lvert$	s,s'	0.02	-	-	137	8.8	7	14	-21.33	-42	-14
Sugeno	s,s'	-	-0.95	-	137	9.2	8	14	-22.8	-42	-18
ceb > 0	s,s'	-	-0.95	0.5	137	8.53	7	13	-20.13	-38	-14

Table 5.14.: *Results* - FGESM-1 simulated sequences and corresponding accumulated rewards for different generalized models.

It can be seen that generalization is effective in generating robust policies by estimating omitted transitions even without verification by further demonstrated requests. Policies resulting from generalized models accumulate reward more closely approximating the policy resulting from the reference model. Nonetheless, using requests is safer and the preferred approach. Further data concerning actual transition probabilities generated for the mission discussed, more tested parameter combinations and evaluation concerning further simulated missions can be found in [117].

5.7. Evaluation of Model Refinement by Geometric Analysis

Geometric analysis for refinement of transition models as presented in Section 4.11 was evaluated using setups CESM-1 and CESM-2. In-depth analysis of single path probabilities can be found in [163]. To compute estimates of mobility transitions, 100 point pairs of a chair and robot pose were sampled from f_{g1}, derived from demonstrations and subsequent state mapping as shown in Figure 5.13. The path segment deviation parameter ε was set to 1 cm, empirically derived from tests with the real robot. Multiple target-pose candidates were evaluated for each sampled pairs and path computed and probabilities computed for each pair as described in Section 4.11 using the "grasp Chair Below Front" *mst*. Different target areas were considered:

1. Target voxel (1): Certainly reaching the chosen target voxel, without colliding, modeling minimal deviations.

2. Neighbors (4+1) : Reaching the target or directly adjoint x-y neighbouring voxels, without colliding, modeling medium target deviations.

3. Box (8+1): Reaching the target or all adjoint x-y neighboring voxels, including diagonally adjoint, without colliding, modeling higher target deviations.

Additional obstacles were introduced into CESM-1 and CESM-2 settings, with the CESM-1 setting having a higher difficulty level (constraint space) and CESM-2 having a lower difficulty level, as illustrated in Figure 5.13. Results are shown in Table 5.15 (CESM-1) and Table 5.16 (CESM-2). It has to be noted that computational effort was highly non-optimized in these tests, with a path roadmap regenerated for each test and being a major factor. Thus, optimized effort could be reduced by at least an order of a magnitude. Results show probabilities intuitively expected for the given settings. Evaluation, including (near) collisions with the real robot was not performed, however, for safety reasons.

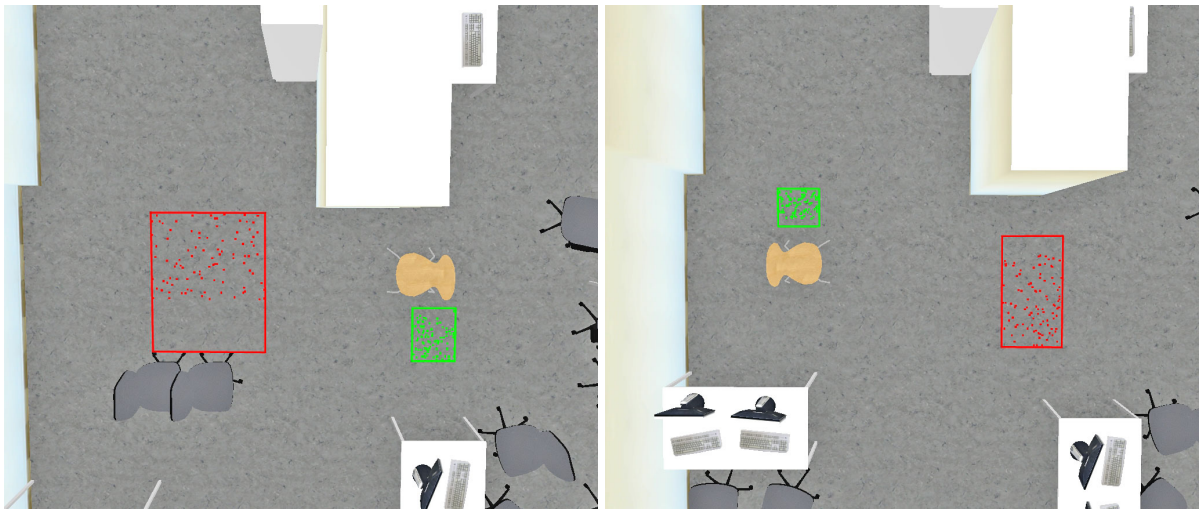
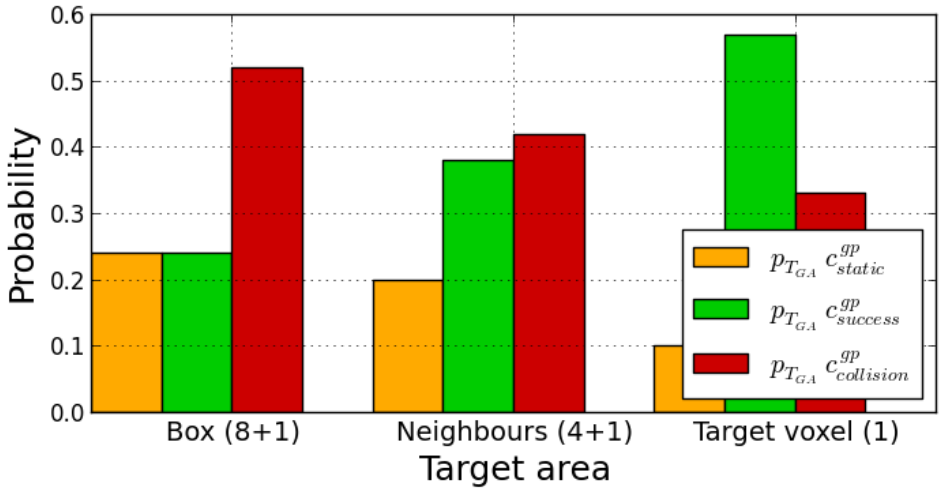
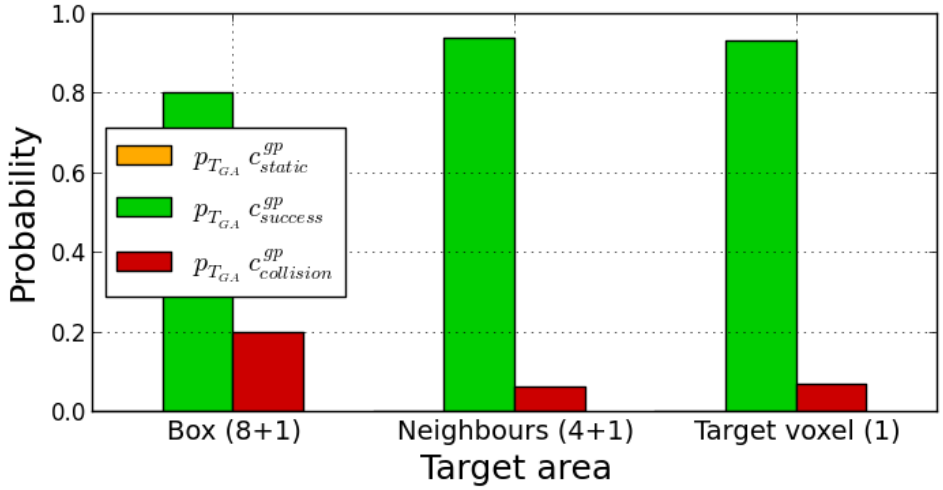


Figure 5.13.: Points indicate pose of robot (red) and chair (green) sampled for path analysis used for PbD model refinement. Additional obstacles are shown graphically. An analysis for CESM-1 is shown on the left with chair orientation towards the right and one for CESM-2 on the right with chair orientation towards the bottom (towards the white table). - [163]



Target area \ $p_{T_{GA}}$,Effort	$p_{T_{GA}}$ c^{gp}_{static}	$p_{T_{GA}}$ $c^{gp}_{success}$	$p_{T_{GA}}$ $c^{gp}_{collision}$	Tested poses	Duration sec.
Box (8+1)	0.24	0.24	0.52	1288	4601
Neighbours (4+1)	0.2	0.38	0.42	1105	3607
Target voxel (1)	0.1	0.57	0.33	1144	4324

Table 5.15.: *Results* - CESM-1 model refinement for goto chair and "grasp Chair Below Front".



Target area \ $p_{T_{GA}}$,Effort	$p_{T_{GA}}\ c^{gp}_{static}$	$p_{T_{GA}}\ c^{gp}_{success}$	$p_{T_{GA}}\ c^{gp}_{collision}$	Tested poses	Duration sec.
Box (8+1)	0	0.80	0.20	3055	3459
Neighbours (4+1)	0	0.94	0.06	2940	3757
Target voxel (1)	0	0.93	0.07	2211	3055

Table 5.16.: *Results* - CESM-2 model refinement for goto chair and "grasp Chair Below Front".

5.8. Evaluation of Learning from Experience in Dynamics Simulation

Suitability of learning transition model probabilities from experience gathered in execution trials in physical dynamics simulation was to be evaluated in several experiments. The goal was to investigate if the approach, described in Section 4.12, is able to improve action selection, choosing more robust actions in the face of uncertainty, while keeping computational time required to run dynamics simulations reasonable. Although computation in this process stage takes place offline, temporal requirements may dominate the duration of the whole PMPM-PbD process as discussed in Section 5.9. Physical dynamics simulation trials cannot be executed much faster than realtime and a large number of samples is required. Experiments have to investigate wether priming the simulation setup by feature states, learned from demonstrations, is sufficient to reduce effort to a reasonable amount. First, however, the suitability of the approach in the context of generating POMDP models is evaluated, as discussed next.

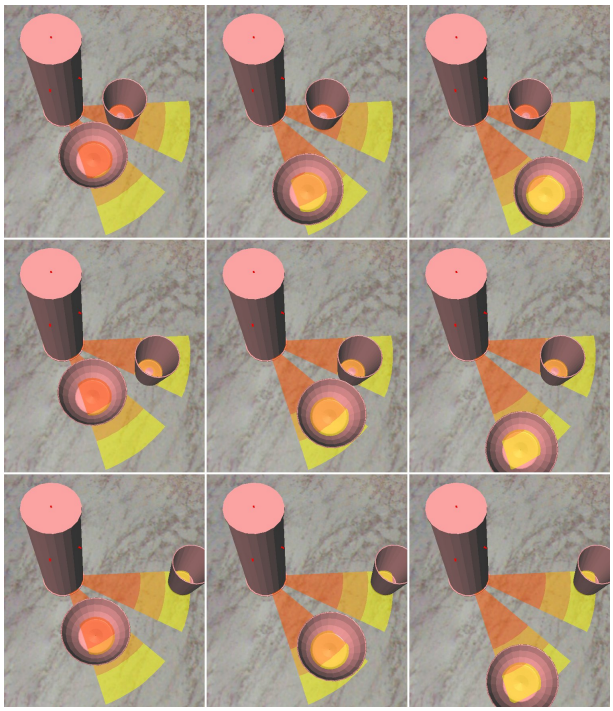


Figure 5.14.: Potential origin states in DSET-1 from s_1^{or} upper left to lower right s_9^{or}. - [12]

5.8.1. Focussed Evaluation of Learning in Simulation Without PbD

In a first experiment set, action selection policies resulting from POMDP transition models, generated by means of learning from experience in simulated trials were to be evaluated against MDP policies and simple heuristics [12]. This was done to assess if the chosen approach is viable in principle. Two aspects were specific to this non-PbD experiment stage:

- Temporal and complexity aspects were less relevant as state space composition was unfocussed in contrast to focussed feature state-action pairs tested in PbD refinement.
- A distinct feature state layout was chosen.

Based on a scene with three objects on a table: the cup, the pringles can, and the bowl, mentioned in Section 5.1, a simple manipulation task was to be performed. Those objects were closely arranged on a table with slightly differing layouts as shown in Figure 5.14. In one experiment *dynamics simulation evaluation task* (DSET-1), the pringles can was to be grasped by three different manipulation action options, in the other experiment (DSET-2), the cup was to be grasped. The other two objects served as obstacles that were arranged in certain relative sectors at a certain distance. Feature states are defined by certain sectors and the combination of layout options creates the potential origin state space S^{or}as shown in Figure 5.14. Based on small-object localization uncertainty, $\rho_{obs}(\xi_i)$ could be defined, from which the observation model resulted for the potential origin state space in DSET-1:

0.8140	0.1260	0	0.0580	0.0020	0	0	0	0
0.0640	**0.8200**	0.0900	0.0020	0.0240	0	0	0	0
0	0.0940	**0.8640**	0	0.0040	0.0380	0	0	0
0.0320	0.0080	0	**0.8040**	0.0900	0	0.0580	0.0080	0
0.0060	0.0400	0.0020	0.0760	**0.7460**	0.0780	0.0020	0.0400	0.0100
0	0.0040	0.0580	0	0.0620	**0.8180**	0	0	0.0580
0	0	0	0.0420	0.0080	0	**0.8640**	0.0860	0
0	0	0	0.0100	0.0500	0.0040	0.0780	**0.7760**	0.0820
0	0	0	0	0.0100	0.0380	0	0.0700	**0.8820**

As can be seen, there is relevant but not excessive uncertainty. The total state space is much larger, being composed of several more potential resulting feature states. A feature F_i is defined for each object:

1. $F_{manipobj}$:= *standing, fallen over, on floor, stable grasp, unstable grasp*
2. $F_{obstacle1}$:= *sectorA1, sectorA2, sectorA3, outside, fallen over, on floor*
3. $F_{obstacle2}$:= *sectorB1, sectorB2, sectorB3, outside, fallen over, on floor*

This leads to a total of 180 states $S := F_{manipobj} \times F_{obstacle1} \times F_{obstacle2}$ and $S^{or} := F_{obstacle1}[1,3] \times F_{obstacle2}[1,3]$. A reward of +5 was given to the state of *stable grasp* $\wedge c'^{obstacle1} = c^{obstacle1} \wedge c'^{obstacle2} = c^{obstacle2}$. In turn, the transition model was computed by taking 10 samples ξ_i from each origin state with 10 observations ξ_i^{obs} per intrinsic configuration sample ξ, repeated for each of the 3 actions, resulting in a total of $10 * 10 * 9 * 3 = 2700$ trials for both DSET. Based on the resulting transition model, policies were computed by SARSOP [80]. In turn, the resulting policy was tested in simulation against an MDP based on the same transition model and a manually defined expert heuristic assigning a most robust action to each state. The same ρ_{obs} as during model generation was applied in each case.

Task DSET-1 Pringles Three different action options a_{mp}^{sgrasp}, representing different power grasps on the pringles can, shown in Figure 5.15, were available in DSET-1. First, the policy was tested for average reward, taking 10 samples ξ per origin state s^{or}, followed by belief computation, action selection and execution in dynamics simulation. It resulted in the following average rewards:

State / Measure	s_1^{or}	s_2^{or}	s_3^{or}	s_4^{or}	s_5^{or}	s_6^{or}	s_7^{or}	s_8^{or}	s_9^{or}
Avg. reward	4.5	4.3	4.4	3.0	3.1	3.0	5.0	4.0	4.5

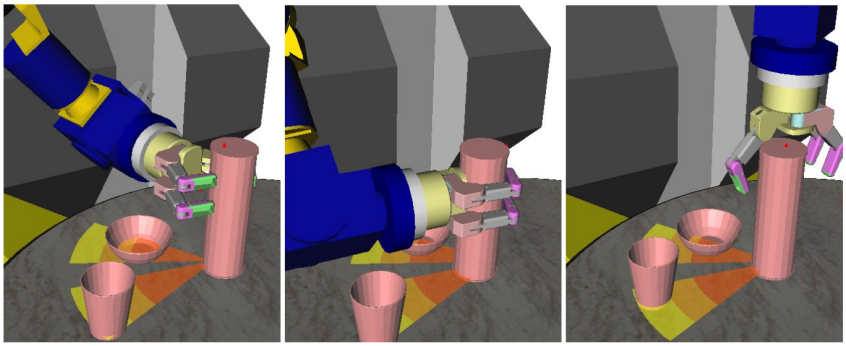
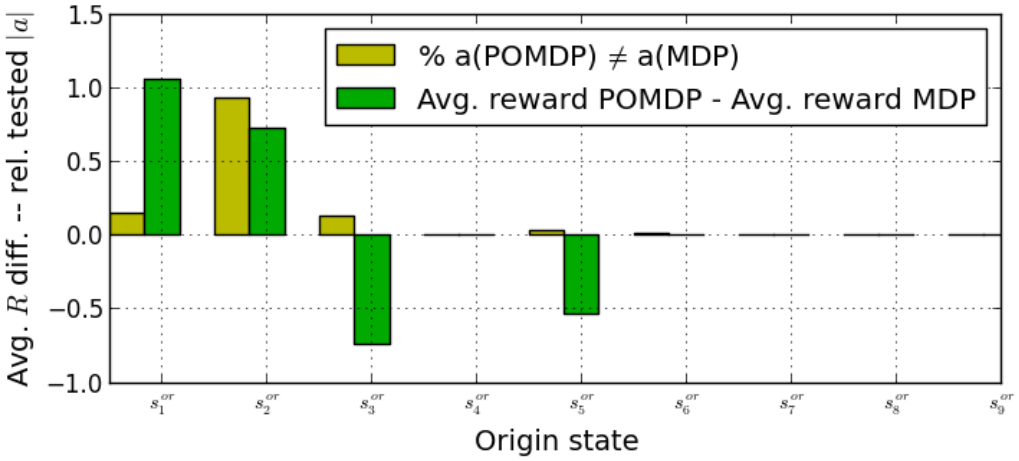


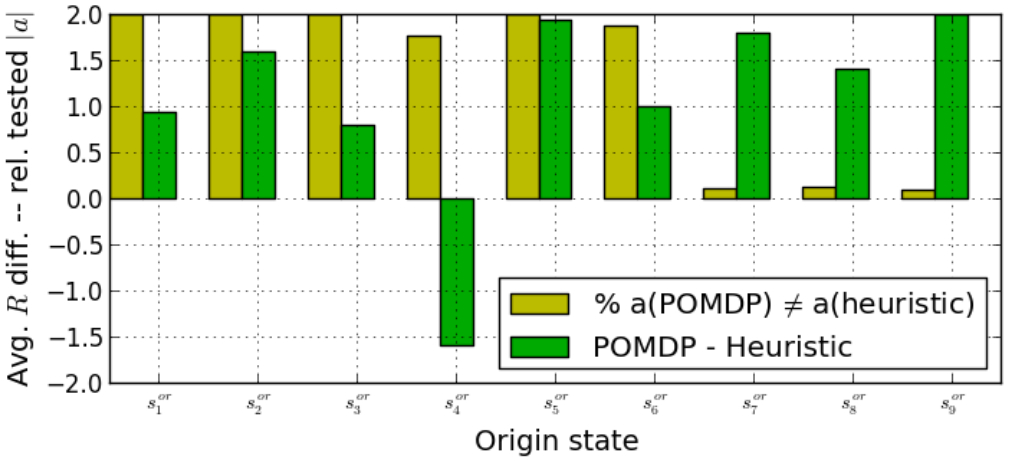
Figure 5.15.: Illustration of actions in DSET-1: a_1^{sgrasp} (left), a_2^{sgrasp} (center), a_3^{sgrasp} (right). - [12]

Subsequently, for each origin state s^{or}, intrinsic configuration and observation pairs (ξ_i, ξ_i^{obs}) were sampled at most 500 times or until 25 different action choices between both active action selection methods had been made. Different action methods were instantiated into trajectories by motion planning on the given ξ_i^{obs}) executed in dynamics simulation given ξ_i. The results, shown in Table 5.17 were obtained in a comparison of POMDP and MDP. It can be noted that with states where obstacles are close to the object to be manipulated, the POMDP is superior, because it is more risk averse, considering its own perception uncertainty. Action choices are nearly the same in cases of more distant obstacles. Subsequently, results were obtained for a comparison of POMDP and manual heuristic, shown in Table 5.18. In this case, the difference is more pronounced.



Measure \ State	s_1^{or}	s_2^{or}	s_3^{or}	s_4^{or}	s_5^{or}	s_6^{or}	s_7^{or}	s_8^{or}	s_9^{or}
% a(POMDP) $\neq$ a(MDP)	14.4	88	12.3	0.4	2.6	1.2	0	0	0
Avg. reward POMDP	**3.46**	**4.53**	4.26	5.00	4.46	5.00	-	-	-
Avg. reward MDP	2.40	3.80	**5.00**	5.00	**5.00**	5.00	-	-	-

Table 5.17.: *Results* - Comparison of POMDP vs MDP in simulated execution of DSET-1.



Measure \ State	s_1^{or}	s_2^{or}	s_3^{or}	s_4^{or}	s_5^{or}	s_6^{or}	s_7^{or}	s_8^{or}	s_9^{or}
% a(POMDP) $\neq$ a(heuristic)	100	100	100	88	100	94	5.3	6.09	4.76
POMDP	**5.00**	**4.86**	**4.60**	3.40	**4.13**	**4.53**	**5.00**	**4.20**	**4.33**
Heuristic	4.06	3.26	3.80	**5.00**	2.20	3.53	3.20	2.80	2.33

Table 5.18.: *Results* - Comparison of POMDP vs heuristic in simulated execution of DSET-1.

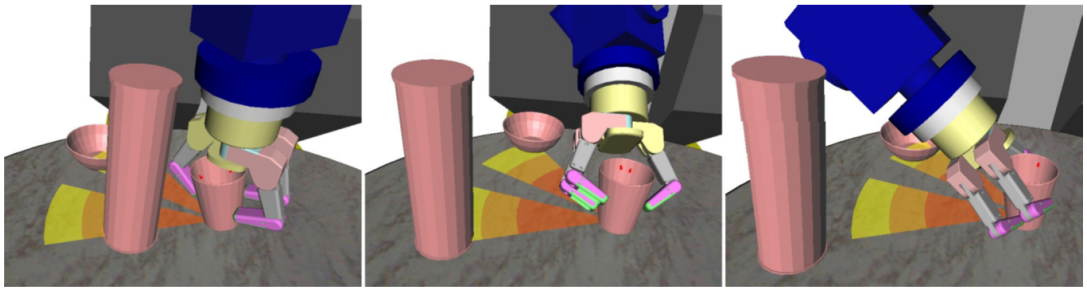


Figure 5.16.: Illustration of actions in DSET-2: a_1^{sgrasp} (left), a_2^{sgrasp} (center), a_3^{sgrasp} (right). - [12]

Task DSET-2 Cup Again, three different action options a_{mp}^{sgrasp}, representing different approach options to grasp the cup, shown in Figure 5.16, were available for selection in DSET-2. Initial average reward estimation simulations show that this task is much more error prone than the previous one.

Measure \ State	s_1^{or}	s_2^{or}	s_3^{or}	s_4^{or}	s_5^{or}	s_6^{or}	s_7^{or}	s_8^{or}	s_9^{or}
Avg. reward	0.3	0.4	0.6	0.6	2.1	1.4	2.3	2.2	3.2

This is easy to explain when comparing Figures 5.15 and 5.16: The pringles can is much more of an obstacle when grasping the cup than the other way around. In this more complex setting, POMDP performance superiority gets more distinct as shown in Tables 5.19 and 5.20. In general it can be noted that the POMDP performs worse in intrinsic s_5^{or} which results from problems with one action and resulting errors, shown in Figure 5.17.

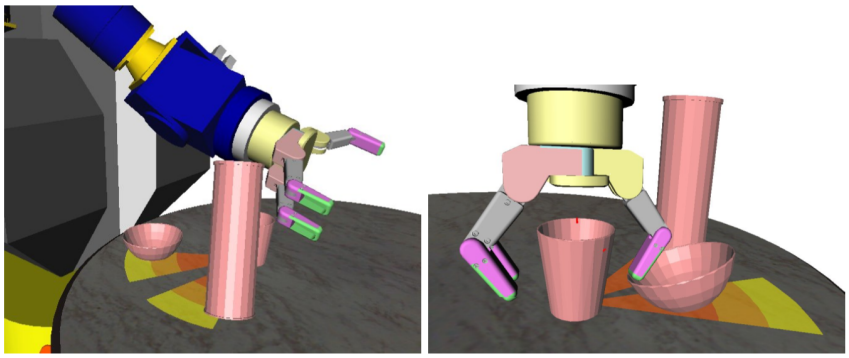
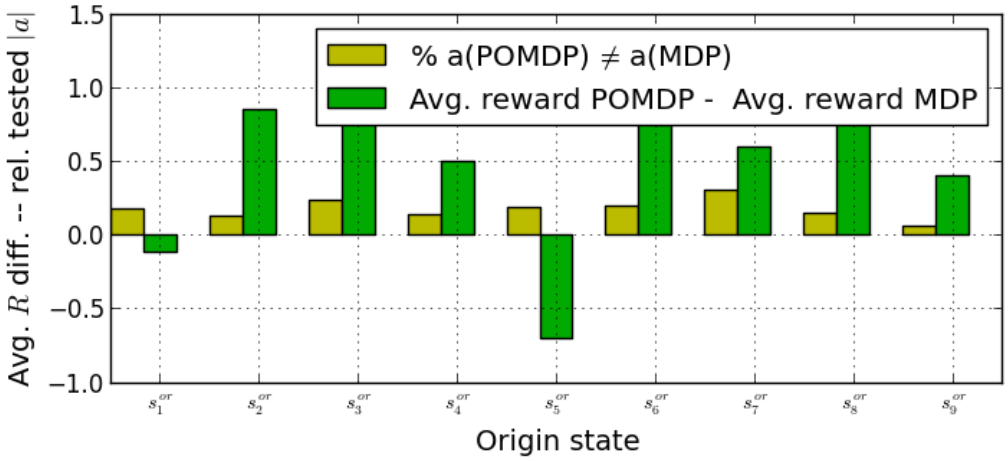
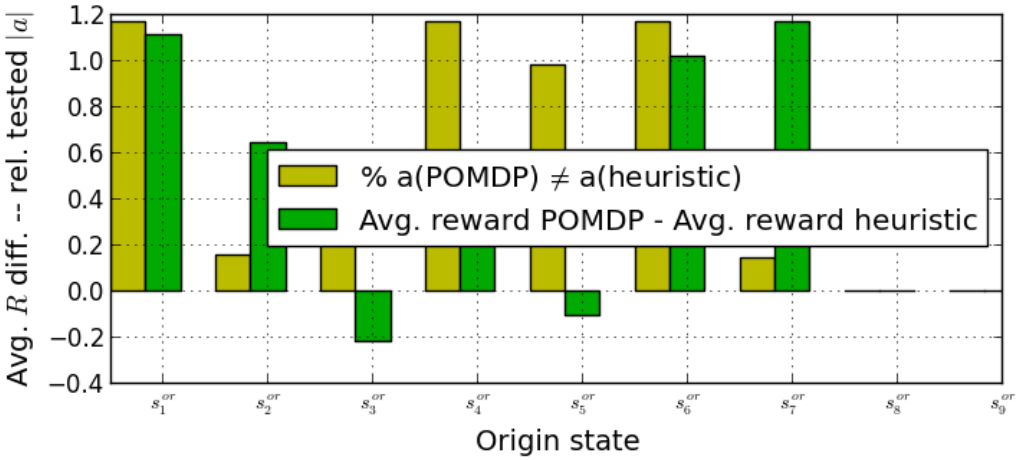


Figure 5.17.: Occurrence of collisions in DSET-2 with pringles can (left) and bowl (right). - [12]



Measure \ State	s_1^{or}	s_2^{or}	s_3^{or}	s_4^{or}	s_5^{or}	s_6^{or}	s_7^{or}	s_8^{or}	s_9^{or}
% a(POMDP) $\neq$ a(MDP)	17	12	23	13	18	19	29	14	6
Avg. reward POMDP	0.20	**0.95**	**1.40**	**0.30**	1.15	**2.60**	**1.20**	**3.50**	**4.60**
Avg. reward MDP	**0.31**	0.10	0.35	-0.20	**1.85**	1.80	0.60	2.70	4.20

Table 5.19.: *Results* - Comparison of POMDP vs MDP in simulated execution of DSET-2.



Measure \ State	s_1^{or}	s_2^{or}	s_3^{or}	s_4^{or}	s_5^{or}	s_6^{or}	s_7^{or}	s_8^{or}	s_9^{or}
% a(POMDP) $\neq$ a(heuristic)	100	13	26	100	84	100	12	0	0
Avg. reward POMDP	**1.23**	**0.64**	0.91	**0.44**	1.34	**2.12**	**2.50**	-	-
Avg. reward heuristic	0.12	0.00	**1.13**	0.21	**1.45**	1.10	1.33	-	-

Table 5.20.: *Results* - Comparison of POMDP vs heuristic in simulated execution of DSET-2.

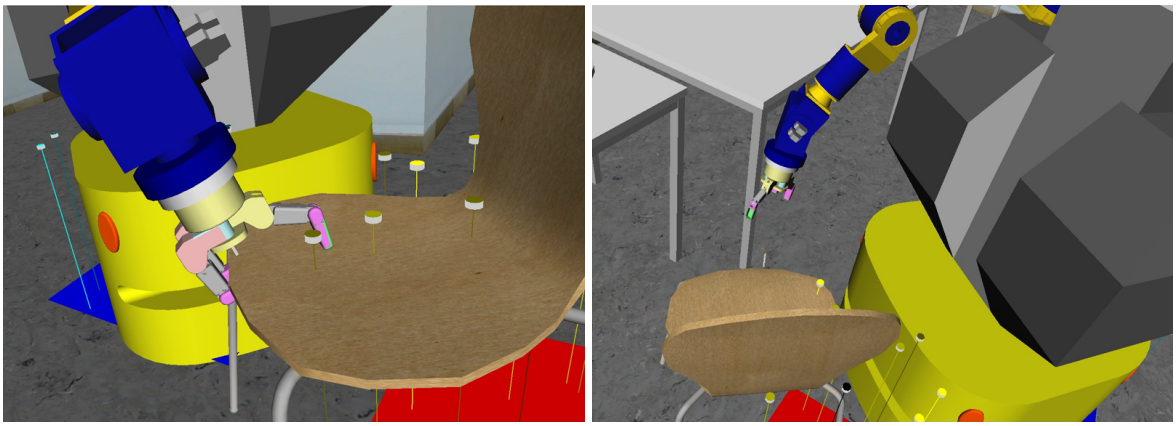


Figure 5.18.: Exemplary view of PbD based simulation trial 1 (left) and trial 2 (right). - [135]

Discussion and limitations This evaluation shows that generating POMDP models based on trials in simulation for some manipulation actions, including qualitatively different manipulation strategies focused on certain target objects, can lead to robust decision-making models.

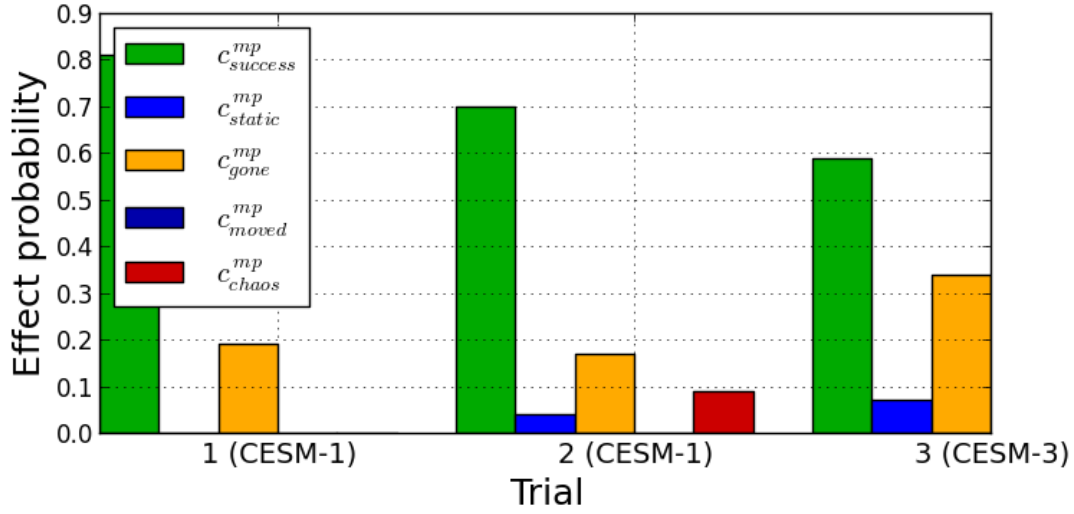
However, without the integration into PbD, severe limitations remain: the state space is overly specific and very large even for simple settings. It is utterly impossible to compute models for any potential setting, to be encountered in any mission. Especially determining which model to apply in a certain mission is not straightforward when assuming partially observable environments.

Yet, these problems can be overcome when only learning from experience in situations, observed during demonstrations and thus relevant for a mission.

5.8.2. Evaluation of Trials in Simulation Based on PbD

To evaluate effect probabilities generated by trials in simulation based on feature categories c^{mp}_{origin} and manipulation actions a^{sgrasp}_{mp}, as discussed in Section 4.12.3, state-action pairs generated by PbD in evaluation missions were assessed [135]. Of several state-action pairs occurring in demonstrations, some state-action pairs were manually selected for further evaluation. In each trial, 144 intrinsic configurations ξ_i were sampled from feature category regions, generated from PbD recordings. Observation deviations were sampled from normal distributions in dimensions x, y and orientation θ independently. The mean was always assumed to be the true pose value, while standard deviations were 10mm for x and y and 1 degree for θ. For each sampled intrinsic pose, a single observed pose ξ^{obs}_i was sampled and used for planning. Only configurations ξ^{ex}_i within the overall reachability of the robot were used for planning and simulation. Selected state-actions pairs were the following:

1. Grasping the chair from the front in CESM-1, as shown in Figure 5.18.



Trial \ Result	$\lvert\xi_i\rvert$	$\lvert\xi_i^{ex}\rvert$	$c^{mp}_{success}$	c^{mp}_{static}	c^{mp}_{gone}	c^{mp}_{moved}	c^{mp}_{chaos}
1 (CESM-1)	144	32	0.81	0	0.19	0	0
2 (CESM-1)	144	23	0.7	0.04	0.17	0	0.09
3 (CESM-3)	144	29	0.59	0.07	0.34	0	0

Table 5.21.: *Results* - Transition probabilities for selected state-action pairs in CESM missions, based on simulated trials.

2. The same state and action as in the previous trial, but another table as obstacle nearby was introduced, as shown in Figure 5.18.

3. Grasping the table in CESM-3.

The number of actually tested configurations ξ_i^{ex} as well as effect probabilities resulting from each trial are shown in Table 5.21. Because the effect $c^{\prime mp}_{jammed}$ was not evaluated it is not reflected in the table. The execution time for the dynamics simulation (without motion planning) took around 2400 seconds for each trial. Between trials 1 and 2, it can be clearly seen that the additional obstacle reduces the success probability as expected. However, the high probability of $c^{\prime mp}_{gone}$ in each trial, resulting from the chair or table having fallen over, does not reflect real world behavior. In real experiments, $c^{\prime mp}_{static}$ or, rarely, $c^{\prime mp}_{moved}$ would result in those situations where the action fails and an unintended momentum is given to the furniture object. It can be concluded that better physical dynamics simulation is necessary to generate realistic distinctions between different failure categories.

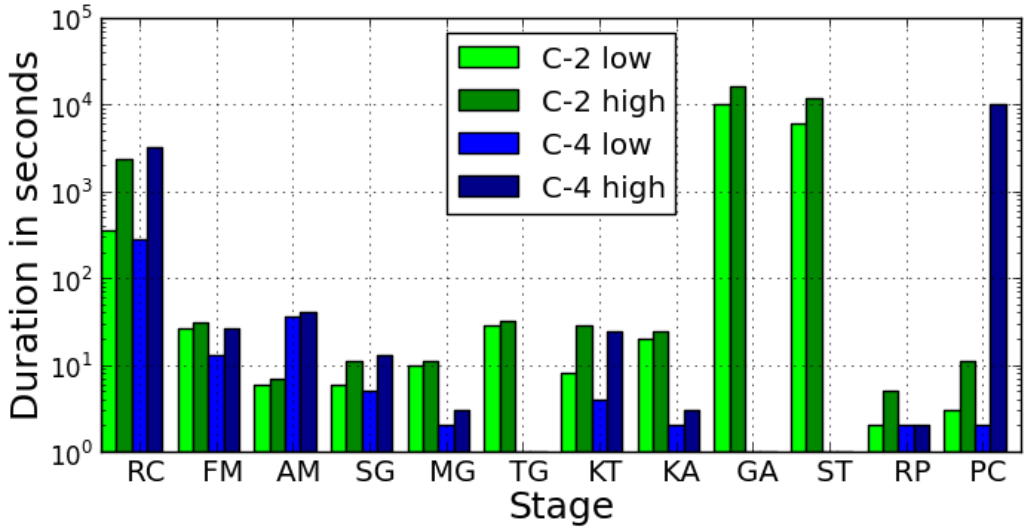
5.9. Evaluation of Process Stage Lead Times

To assess bottlenecks concerning computational complexity in the process chain, actual processing lead times were measured for different sets of demonstrations of CESM-2 and CESM-4. The following process stages were assessed, based on shared sets of demonstrations:

1. Recording of demonstrations (RC)
2. Generation of feature mappings (FM)
3. Generation of action mappings (AM)
4. Segmentation of recordings (SG)
5. Preliminary model functional expression generation (MG)
6. Generalization of transitions (TG)
7. Adding information to the knowledge base: DL "tell" (KT)
8. Inferring information from the knowledge base for model completion: DL "ask" (KA)
9. Model refinement based on geometric analysis (GA)
10. Model refinement based on trials in dynamics simulation (ST)
11. POMDP model generation from rules, using function expressions (RP)
12. Policy computation based on the POMDP model (PC)

For each mission, different sets of physical demonstrations, ranging from three to ten demonstrations were evaluated. Refinement steps GA and ST were only evaluated on selected state-action pairs individually and total times are just an extrapolation as if computing refinement for all relevant pairs in one single run. Lead time measurements were performed on personal computers with similar speed, mostly on a Intel Core i5 CPU with 4 GB RAM and some further assessments on a Core 2 Duo CPU. All process stages only utilized a single CPU for their primary computation. As lead times can vary considerably between different sets of demonstrations and vary by many orders of magnitudes between different process stages, only rough lower and upper bounds of taken measurements are shown in Table 5.22.

As can be seen, demonstration times dominate the pure PbD stages, thus those are unproblematic. However, both geometric and physical dynamics simulation refinement stages need excessive computation time in their current realization. Therefore, further optimisations are



Stage / Mission	RC	FM	AM	SG	MG	TG	KT	KA	GA	ST	RP	PC
C-2 low	360	25	5	5	9	28	7	19	10000	6000	1	2
C-2 high	2400	30	6	10	10	31	28	23	16000	12000	4	10
C-4 low	280	12	35	4	1	-	3	1	-	-	1	1
C-4 high	3200	25	40	12	2	-	23	2	-	-	1	10000

Table 5.22.: *Results* - Exemplary process stage lead times for CESM-2 (C-2) and CESM-4 (C-4).

necessary there of at least one order of magnitude to allow practical application. Yet, all process stages except policy computation show highly linear effort and can thus be handled in a predictable manner. However, policy computation effort scales in a highly non-linear fashion, as shown in the last column in Table 5.22. Policies were computed for CESM-4 with differing transition and observation models – but exactly the same state and action set – leading to differences of five orders of magnitude in computation time.

This leads to the conclusion that controlling the complexity of transition and observation models, represented by the number of non-zero probability entries, is the main hurdle for practical application of the presented concept in missions of intermediate size.

5.10. Evaluation Conclusions and Limitations

In the given experiments, process stages were evaluated individually and in combination, in simulation and on real hardware based on several different missions. The technical system has shown its applicability to real-world problems.

More extensive naive-user studies or long-term experiments would be desirable but were beyond the scope of this thesis. It shall be noted explicitly at this point that extensive, truly controlled mission experiments with fully autonomous physical service robots, integrating all three major skill domains and with the focus to collect expressive data beyond "show-off demonstrations" is extremely hard given the current state of the art in service robotics. In this chapter, experiments resulting in the smallest result tables were by far the most challenging and laborious to set up and perform in a controlled manner.

In the literature, such experiments are rare. This Chapter aims to reduce that gap to some extent.

6. Summary and Conclusions

Autonomous decision making in complex domestic environments requires a robot with an elaborate model of potential situations and action effects. Models that incorporate quantification of uncertainties in both perception and courses of events while covering diverse skills domains are complex. Such complexity leads to two major challenges:

1. Organizing models in a comprehensive and yet efficient manner.
2. Acquisition of models.

6.1. Contribution Summary

As presented in the previous Chapters, this thesis outlines a concept for both modeling as well as model acquisition on complex domestic service robots. The well founded concept of POMDPs was chosen to handle autonomous decision making as it is able to consider uncertainties in both environment observation as well as action effect prediction.

To manage diverse sets of skill domains, different aspects of the world, observable by the robot, are organized in features, which ground abstract POMDP state representations of diverse types of situations. Abstract decisions are represented by actions, which are realized by compound subtasks, triggering elementary robot skill execution. Strategies to derive actual model probabilities for both observation uncertainties considering varying robot perception skills as well as action prediction are outlined. A functional expression system is presented, enabling efficient POMDP model generation for diverse missions, generating a full model from a compact rule-based description. Furthermore, a background knowledge representation system based on description logic reuses information about the world and the robot for multiple mission models. Such model organization allows to represent versatile service robot missions in the information processing system of the robot. The representation is in turn used for autonomous decision making during execution.

Acquisition of those models, thus generation of a model to be used in the information processing system of the robot during execution, is realized by interactive learning. Three main challenges were tackled to achieve full model generation from observation of human demonstrations and autonomous refinement:

- The chosen model organization needs knowledge about the grounding of abstract states and action in real-world situations. The presented approach to this challenge autonomously generates state and action grounding based on human demonstrations, using spacial clustering and discretization, as well as trajectory-oriented action mapping.

- An abstract POMDP model has to be generated while there is potentially lacking model knowledge after initial demonstrations. This challenge is tackled by generating state and action sets based on demonstrations and using previously generated state grounding. Subsequently, the preliminary abstract model is explored for potentially lacking action effect knowledge, with model portions rated for relevance and in turn verified by posting interactive request to human mission teachers.

- There is a need for information about robot-specific action effect probabilities, which cannot be inferred from human demonstrations. To derive this information, geometric analysis and robot learning from trials in dynamics simulation is performed, directed by information about the scene and mission as gathered from observation. That information refines the preliminary POMDP model into a final model.

Concept and process stages for both learning and execution have been completely implemented and integrated on two physical service robots with multiple skills domains. Experimental evaluation was conducted, both focussed on performance and peculiarities of individual process stages as well as overall process suitability. As a result of being realized as a complete system, the comprehensive concept has proven its suitability to tackle abstract-level autonomy of service robots with multiple skill domains.

6.2. Discussion and Limitations

On analyzing the approach taken and potential future extensions, several insights stand out:

- There is no way around symbols on an abstract, coordinating layer. Symbols do not necessarily have to be meaningful for humans outside communication with the robot as long as there is sufficient grounding inside the information processing system of the robot. In fact, symbols are just a way to organize information, reduce complexity and focus on relevant aspects alone. Symbols are compact references to concepts, clusters and data structures. These references both reduce information complexity and bring information of highly diverse resources into coherent forms where algorithms have to consider them equally. Yet, a reasoning and planning system using symbols is only as powerful as the methods that map situations and processes onto symbols as well as further information –

like probabilities – attached to the symbols. In the given system, especially state mappings are still limited as the focus was on learning those from observations. More sophisticated methods could be considered as discussed in the next Section.

- Any autonomous reasoning and planning system has to make a trade-off between the model expressiveness and the complexity implied by the model. Expressiveness describes the scope of information that can be contained in a model and then used by planning algorithms. Furthermore it may also describe how much information is acquired when creating such a model and how well it approximates real-world behavior. However, with growing expressiveness complexity challenges arise quickly, for instance, in the given system POMDPs have the ability to model uncertainty in both perception and potential courses of events. The system as presented still uses non-hierarchical POMDPs, which limits mission size, and the learning process does not exploit life-long learning extensively. Different levels of model expressiveness - for example for short-term and long -erm planning in a mission or the learning process - could improve the balance between expressiveness and complexity.

- Probably the most important contribution is the insight how extensively a tight combination of highly diverse paradigms can enable abstract learning in ways impossible otherwise. The proposed system shows in depth how greatly different techniques like PbD, POMDPs, description logic, motion planning and physical simulation can work together in model acquisition. Most of these paradigms applied alone for model acquisition would be just infeasible, as shown in learning from simulated trials alone without a preliminary model skeleton acquired by PbD. It should be clearly expressed that it came as a surprise *how well* some problems can be overcome by combining highly differing paradigms in the right way during development of the presented system. Exploiting these opportunities has to go beyond just collecting groups of individually strong methods or sets of skills – though these are also exploited in the system which is based on such sets of skills – but *directed, complementary* use of paradigms as shown in the model acquisition process.

- A key to future more capable systems will be huge amounts of knowledge. Examples discussed in this thesis are state and action mappings, state sets, primary action effects, secondary action effects and observation uncertainty. Various different representations for these knowledge aspects could be utilized. Persistently storing such knowledge, adding new knowledge, reprocessing existing information when introducing new data as well as efficient search and retrieval are non-trivial on large amounts of data. In this thesis, large

amounts of data were not investigated, but the topic will be crucial when tackling more complex missions.

- There is no online planning and reasoning applied in the given system. All model aspects are compiled offline and finalized by policy computation. This design choice was made to focus exclusively on modeling and model acquisition – complex challenges on their own. However, there is no fundamental limit to online model improvement and policy computation as discussed in the next Section. It is reasonable to assume that model acquisition process stages are compatible with online planning and reasoning techniques.

6.3. Outlook

Many of the current limitations in the system may be overcome by including new methods currently being investigated in research. Crucial limitations include model and planning complexity, online-time planning, reasoning and learning as well as large amounts of knowledge.

- Relational stochastic models [83] are a promising decision-making model representation, similar to MDPs and able to consider uncertainties. However, states can be expressed in manners more suitable for service robot settings, reducing model and planning complexity.

- Future approaches in hierarchical representations may be able to tackle the complexity problem by making more compromises between precision and model scope.

- Flexible state spaces for online planning may be another way to tackle the problem of complexity. States and action can be added to the model during execution time with computation of new policies . Some process stages of the presented system are highly suitable to such an approach, adding states, actions, transitions and rewards on the fly.

- POMDPs are special forms of DBNs and as such the given model acquisition process could be extended to generating DBNs. Inference techniques on such DBNs may then be suitable for solving decision-making problems.

- For practical purposes, state mapping currently uses quite crude representations. While sufficient in abstract missions investigated, more complex missions might require more sophisticated state mappings. A contender for such a mapping is GMMs, which are well suited to be probabilistic representatives of situation clusters defined on any continuous domain and in turn applied for probabilistic belief computation. Learning such GMMs

from observations could directly use the existing clustering process and sampling from such GMMs in refinement stages ist straightforward. However, such a GMM mapping is less easy to understand for human engineers who debug learned missions and thus requires an otherwise mature system.

- In combination with online model compilation and policy computation, further online DL inference reasoning or action-effect likelihood estimation could be performed. Yet, temporal constraints that are currently not important, would then be crucial, leading to new challenges.

- Interactive learning could be integrated into mission execution. With the robot being able to improve action-effect and situation knowledge from encountered courses of events, learning from experience could be enhanced. Process stages and knowledge representations, suitable for learning from experience exist and therefore introduction of online learning should be straightforward.

- In general, more extensive lifelong learning and accumulation of massive amounts of data is necessary for a robot to be able to execute large numbers of complex missions. It is clear that including such capabilities in the presented system would require massive further research and extensions and is not trivial.

6.4. Conclusions

In general, moving towards truly autonomous domestic robot servants will require broad progress on the following fronts:

1. Improved robust sensors, especially fast 3D point cloud vision and compliant actuators.

2. Improved perception and motion planning skill algorithms, considering real-world uncertainty and utilizing such hardware.

3. Highly integrated situation assessment, reasoning, planning and learning compounds, considering real-world uncertainty, on top of the perception and actuation skills.

The present thesis makes a contribution to the latter point, giving insight into highly integrated processes, that can lead towards working systems. Actually implemented as a prototype for experiments and analysis it also gives further insight into practical software and handling challenges of such systems.

However, architects and engineers always need to be conscious about the fact that real-world complexity will always require tradeoffs between model expressiveness and computational complexity. Also, there is no way to prove model correctness in the face of a real-world setting.

Humans can only overcome the curse of complexity because they conceptualize, focus, utilize heuristics and apply learned, static patterns. Consequently, there are always limits to reasoning and decision making, leading to errors and suboptimal action choices. Perfection is impossible in real-world settings, thus not only is it true that "to err is human", but more generally "autonomy means to err". When that fact is accepted, robot servants may at some point become great tools for civilization.

A. Glossary and Notation

Term	Description	Reference
Access mask	Identifier to select multiple entries in a matrix	
POMDP functional expression access mask	Identifier to modify multiple entries in POMDP model components at once	Sec. 3.6
Generalization mask	Identifier to select multiple transitions	Sec. 4.7.1
Action a	A symbolic representation of a self-contained robot task, executable in one chunk.	Sec. 2.1.2
Action space A	Set of all actions in a model	Sec. 2.2.2
Agent activity G_a	Recording components corresponding to human actions	Sec. 4.1
Autonomy	Flexible agent behavior in complex environments	Sec. 2.1
Background knowledge	Persistent information valid for multiple missions	Sec. 3.7
Bayesian probability	Information based value estimate	Sec. 2.2.1
Bayesian network (BN)	Graph of probabilistic conditional dependence	[122]
Dynamic Bayesian network (DBN)		[122]
Belief (state) b	Subjective Bayesian estimate of a world state by an agent	Sec. 2.2.3
Reachable belief	Belief states which may occur in a certain POMDP model	Sec. 2.2.5
Binary space partitioning		Sec. 4.2.3
Category $c^i_{x_i}$	(= feature state), state element in a feature	Sec. 3.2
Clustering	Grouping data point into a common class	Sec. D.2
k-means clustering	Clustering method	Sec. D.2
EM clustering	Clustering method	Sec. D.2
DBScan clustering	Clustering method	Sec. D.2
DB index	Optimal cluster number measure	Sec. D.2
SD index	Optimal cluster number measure	Sec. D.2
XD index	Optimal cluster number measure	Sec. D.2

Term	Description	Reference
Cluster enclosing box (CEB)	Rectangle fully enclosing a point cluster	Sec. 4.2.3
Concept, DL	Description logic class described by axioms	Sec. 3.7.4
Description logic (DL)		Sec. 2.6
DL axiom	TBox, class information, stored peristently	Sec. 3.7
DL fact	ABox, instance information, generated on the fly	Sec. 3.7
Discretization	Mapping a continuous domain onto a finite set	Sec. 3.2
Dynamic motion primitive (DMP)	Skill representation for manipulation imitation learning	Sec. 2.4.1
Elementary operator	Atomic skill operation in a task scheme	Sec. 2.4.2
Execution(-time)	Robot performs a role autonomously	Sec. 3.1
Feature f_i	Model aspect representing perceivable environment properties of a certain kind	Sec. 3.2
Feature mapping f_i	Function, mapping perception skill domains onto discrete category symbols	Sec. 3.2
Feature state $c^i_{x_i}$	(= category) a concept, representing a class of similar environment situations in a domain	Sec. 3.2
Feature state space F_i	Set of all feature states in a certain model	Sec. 3.2
Feature-discretized mapping f_{g1}	Function mapping input values onto categories	Sec. 3.2
Feature-state mapping f_{g2}	Function mapping feature states onto (PO)MDP states	Sec. 3.2
Feature state description tree (FSDT)	Quality assessment tree for alternative discretization assessment in PbD state mapping	Sec. 4.2.5
Separation choice node (SCN)	Node in an FSDT, representing alternatives	Sec. 4.2.5
Feature state separation node (FSSN)	Node in an FSDT, representing a split into further multiple categories	Sec. 4.2.5
Gaussian mixture model (GMM)	Probabilistic representation of data membership	Sec. D.3
Gaussian mixture regression (GMR)	Representation, computed from GMMs, better suited for distance computations	Sec. D.3

Term	Description	Reference
Generalization of transitions	Estimating new, non-zero transitions from existing non-zero ones	Sec. 4.7.1
Generalization confidence (gc)	Basic estimate confidence for a generalized transition	Sec. 4.7.3
Non-observation bias (nob)	Factor, taking likelihood for demonstrations being omitted, based on frequencies, into account	Sec. 4.7.3
Certainty bias (ceb)	Factor, limiting increased stochasticity	Sec. 4.7.3
Grasp Wrench Space	Metric for stability of a grasp using forces	Sec. 4.12.2
Hidden Markov model (HMM)	Model for observing hidden state sequences	Sec. 4.5
Human interaction role (HR)	Human interaction role demonstrating human teacher	Sec. 4.1
Human-Robot interaction (HRI)	Interaction between humans and robots by means of natural speech, gestures and touch	Sec. 1
Human fully body motion activity	Temporal segment of human body configuration movement, classified by a symbolic label	Sec. 2.7.1
Hough ellipse detection	Computer vision technique to detect ellipses in pixel images	Sec. 3.3.4
Imitation learning	Usually a notion for skill-level manipulation PbD	Sec. 2.4.1
Interactive learning	Robot learning from humans or active environment exploration	Sec. 1.1
Invariance		
Agent invariance	Mission properties, valid when performed by different physical agents	Sec. 3.7.1
Environment invariance	Mission properties valid in varying settings	Sec. 3.7.1
Learning by exploration	Learning from executing focussed actions and evaluating results to refine action knowledge	Sec. 2.5
Macro operator	Abstract task reference in sequence PbD	Sec. 2.4.2
Manipulation strategy (mst)	Powerful constraint-based motion planning manipulation skill and task representation	Sec. 2.7.2

Term	Description	Reference
Markov Decision Process (MDP)	Decision making framework based on the Markov property and considering stochastic action effects	Sec. 2.2.2
Partially Observable MDP (POMDP)	MDP considering imperfect state knowledge	Sec. 2.2.3
Mixed Observability MDP (MOMDP)	Mixed fully observable MDP and POMDP representation	Sec. 2.2.4
Measurement m	Abstract observation, received by an agent	Sec. 2.2.3
Mission	Global, self contained objective of an autonomous robot	Sec. 1
Mobility	Self-localization and navigation skill domain	Sec. 1
Monte-Carlo method	Deriving process parameters by sampled simulations	[122]
Motion state	Data point type attribute in PbD state mapping	Sec. 4.2.1
Object manipulation	Interaction with the physical world using grippers	Sec. 1
Observation	1) Execution-time: abstract world state perception 2) Recording of human demonstrations	Sec. 2.2.3
ODE	Open Dynamics Engine physics simulation	Sec. 2.5
OpenRAVE	Robot simulation framework	Sec. 4.12.1
OPPL	Rule language to work on DL expressions	Sec. 3.7.2
OWL	Ontology language, implementing DL subsets	Sec. 3.7.2
Partially Observable	Intrinsic state of the world being uncertain (hidden) to an agent	Sec. 2.2.3
Physical dynamics simulation	Computationally simulating Newton body dynamics	Sec. 2.5
Planning	Reasoning of an agent to find suitable action choices	Sec. 2.1.2
Motion planning	Search for arm/hand motion trajectories	Sec. 2.7.2
Logic-based planning	Search for abstract action sequences	Sec. 2.1.2
Probabilistic planning	Search for robust action choices under assumptions of stochastic courses of events	Sec. 2.2
PMPM-PbD	PbD framework presented in this thesis	Sec. 1.4
Policy	Action choice function for decision making	Sec. 2.2.5
Programming by demonstration (PbD)	Teaching robot skill/task/mission knowledge by natural demonstrations of human domain experts	Sec. 2.4

Term	Description	Reference
RANSAC plane fitting	Algorithm for optimally fitting planes into point clouds	Sec. 3.3.4
Rational agent	Autonomously reasoning entity, embedded into a world	Sec. 2.1
Refinement of a model	Adapting mission model parameters to better reflect real world properties	Sec. 1.4
Reward r	Positive signal to an agent, feeding motivation	Sec. 2.2.2
Reward model R	Model of all reward signal values in relation to states and actions	Sec. 2.2.2
Robot role (RR)	Robot role demonstrating human teacher	Sec. 4.1
Segmentation (of a demonstration)	Generation of an abstract state-action sequence from continuous demonstration recordings	Sec. 4.4
Similarity metric	Measure of similarity between two potential abstract measurements	Sec. 3.3
Utterance similarity	Similarity between speech sentences	Sec. 3.3.1
Human body activity similarity	Similarity between classified human body configurations	Sec. 3.3.2
Situation E_s	A unique configuration of the world in its entirety	Sec. 3.2
Skill	Control-level capability of a robot	
Skill domain	Major ability aspect of a robot	Sec. 3.1.1
Perception skill (component)	Skill primarily delivering information about environment aspects for the abstract level	Sec. 3.1.1
Actuation skill (component)	Skill primarily controlling actuators	Sec. 3.1.1
State s	Abstract class of similar situations	Sec. 2.1.2
State space S	Set of all states in a model	Sec. 2.2.2
Stochastic courses of events	Courses of events with action effects not deterministically predictable	Sec. 2.2.2
Primary stochastic action effect	Effects mostly independent from specific robot skill characteristics	Sec. 3.4.2
Secondary stochastic action effect	Effects mostly defined by robot skill characteristics	Sec. 3.4.2
SWRL	Rule language to work on DL expressions	Sec. 3.7.2

Term	Description	Reference
Task	Compound of actions (skill execution), yet not self-contained autonomy as a mission	Sec. 3.4
Tool center point (TCP)	End-effector spot definition of a robot arm	Sec. 2.7.2
Transition v	Transfer of a specific state into another by a certain action	Sec. 2.2.2
Transition model T	Probabilities of all transitions in a model	Sec. 2.2.2
Meta transition model	Causally supported transition model for excluding impossible transitions	Sec. 4.5
Transition frequency model TF	Unnormalized model of transition counts	Sec. 4.6
Transition hypotheses v_g	Unobserved transitions, estimated based on observed ones	Sec. 4.7.1
Transition mask κ	Identifier to select groups of transitions	Sec. 4.7.1
Trial	An execution trial, potentially in simulation to learn experience from	Sec. 4.12.1
Utility U	Projected accumulated rewards	Sec. 2.2.2
Value function Γ	Utility for any possible (belief) state	Sec. 2.2.3
Value function plane α	Element of value functions, resulting from value iteration algorithms	Sec. 2.2.3

Symbol	Description
S	State space
s	State
S_p	Primary effects state space
S_D	Demonstration state space
S_E	Secondary effects state space
A	Action set
a	Action
A_p	Primary effects action set
A_D	Demonstration action set
A_E	Secondary effects recovery action set
A_I	Information gain action set
M	Measurement set
m	Measurement
T	Transition Model
T_D	Demonstration transitions
T_{GA}	Transitions derived from geometric analysis (GA)
T_{D+GA}	Transitions derived from demonstrations and GA
T_{Sim}	Transitions derived from trials in dynamics simulation
$T_{D+GA+Sim}$	Merged transition model
T_E	Unspecific secondary transition probabilities
TF_D	Demonstration transition frequency model
O	Observation model
R	Reward model
b	Belief (state)
$R(b_0)$	Reachable belief
Γ	Value Function
α	Value function plane
D	(Skill) Domain

Symbol	Description
E_s	Environment situation
G_a	Agent activity, distinguished from situation in PbD
$f_g(E_s)$	Overall, general feature mapping
f_{g1}	Feature discretization: situation to feature state mapping
f_{g2}	Feature state tuple to state mapping
$f_{feat_i}(E_s)$	Specific feature mapping
F_i	Feature state space
$c^i_{x_i}$	Category = feature state in feature i
FS	Feature state tuple space
FM	Feature measurement tuple set
$\Psi(FS)$	Feature state tuple space to state space mapping
ω_{fs}	Weights for merging transition probabilities in combine states
$\langle m_i, m_j \rangle$	Utterance similarity metric
$\langle a_i, a_j \rangle$	Human body activity similarity metric
mst	Manipulation strategy [graph]
k	Cluster number
$\vec{\omega}$	GMM weights
$p_{Hough}(o_x, o_y, \alpha, \beta, \theta)$	Hough ellipse correctness confidence
$\top$	DL ontology root: thing
Obs	A single demonstration observation trace
$obs(t)$	Single demonstration recording data point
Ξ_{Demo}	Set of multiple abstract demonstration sequences Obs
CEB	Cluster enclosing box
ω_{sim}	CEB similary measure
$q(SN_i)$	FSSN quality
$X^{tcp,act}_m$	Human body activity classified hand trajectory segment
$X^{tcp}_{j,i}$	Manipulation strategy j, training data trajectory i
$\Phi(mst)$	GMR of mst training trajectories
ω_t	Trajectory data point weights
Q_s	Abstract state-action sequence of a course of events
ε_t	Temporal demonstration segmentation threshold

Symbol	Description
$v : T(v) := T(s_i, a_k, s'_j)$	Transition model entry - a single transition
v_g	Generalized transition hypotheses
$\kappa(s^*, a^*, s'^*)$	Transition mask
V_{mask}	Set of transitions defined by a mask κ
$gc(v_g)$	Generalization confidence of a generalized transition
α_{gc}	Generalization confidence similarity weight
β_{gc}	Generalization confidence non-observation bias weight
γ_{gc}	Generalization confidence certainty bias weight
$\rho^s(s)$	State relevance measure for v_g ranking
$\vec{\omega}^s$	State relevance component weights
$\rho^v(v)$	Transition relevance measure
$\{\sigma_i^m\}$	Simulation trial virtual observation deviations
$\{\sigma_j^a\}$	Simulation trial virtual actuation deviations
ρ_k	Simulation trial situation and observation sample densities
ξ	Simulation trial situation (world configuration)
W_r	Grasp Wrench Space
$\varepsilon_{grasp}^{wrench}$	Grasp wrench space threshold to be considered a stable grasp

B. Processing Schemes and Algorithms

In the following, algorithms, developed in the scope of the presented system, are depicted schematically or in pseudocode.

Algorithm 1 Generate request sequences

Input: T_D, V_g^{verify} sorted by $\rho^v(v_g)$, S_1, max_l, max_g

Output: $\Xi_{requests}$

$V_g^{request} = \emptyset$
$V_g^{discarded} = \emptyset$
while $v_g \in V_g^{verify} \wedge v_g \notin V_g^{request} \wedge V_g^{discarded} \neq V_g^{verify} - V_g^{request}$ **do** {Check hypotheses for paths}
  if $Q_{path}^{min}(s_1 \in S_1, s \in v_g) \neq \emptyset$ **then** {Compose a request sequence}
    $Q_{path}^{request} = Q_{path}^{min}(s_1 \in S_1, s \in v_g) + v_g$
    $app_g = max_g$
    $V_g^{requests} \leftarrow v_g$
    while $\exists v_g^{minpath} \in V_g^{verify}, v_g^{minpath} \notin V_g^{request}, |Q_{path}^{min}(s' \in v_g, s \in v_g^{minpath})| < max_l \wedge app_g \neq 0$ **do**
      $Q_{path}^{request} = Q_{path}^{request} + Q_{path}^{min}(s' \in v_g, s \in v_g^{minpath}) + v_g^{minpath}$
      $app_g = app_g - 1$
      $V_g^{request} \leftarrow v_g^{minpath}$
      if $v_g^{minpath} \in V_g^{discarded}$ **then**
        remove $v_g^{minpath}$ from $V_g^{discarded}$
      end if
    end while
    $\Xi_{requests} \leftarrow Q_{path}^{request}$
  else
    $V_g^{discarded} \leftarrow v_g$
  end if
end while

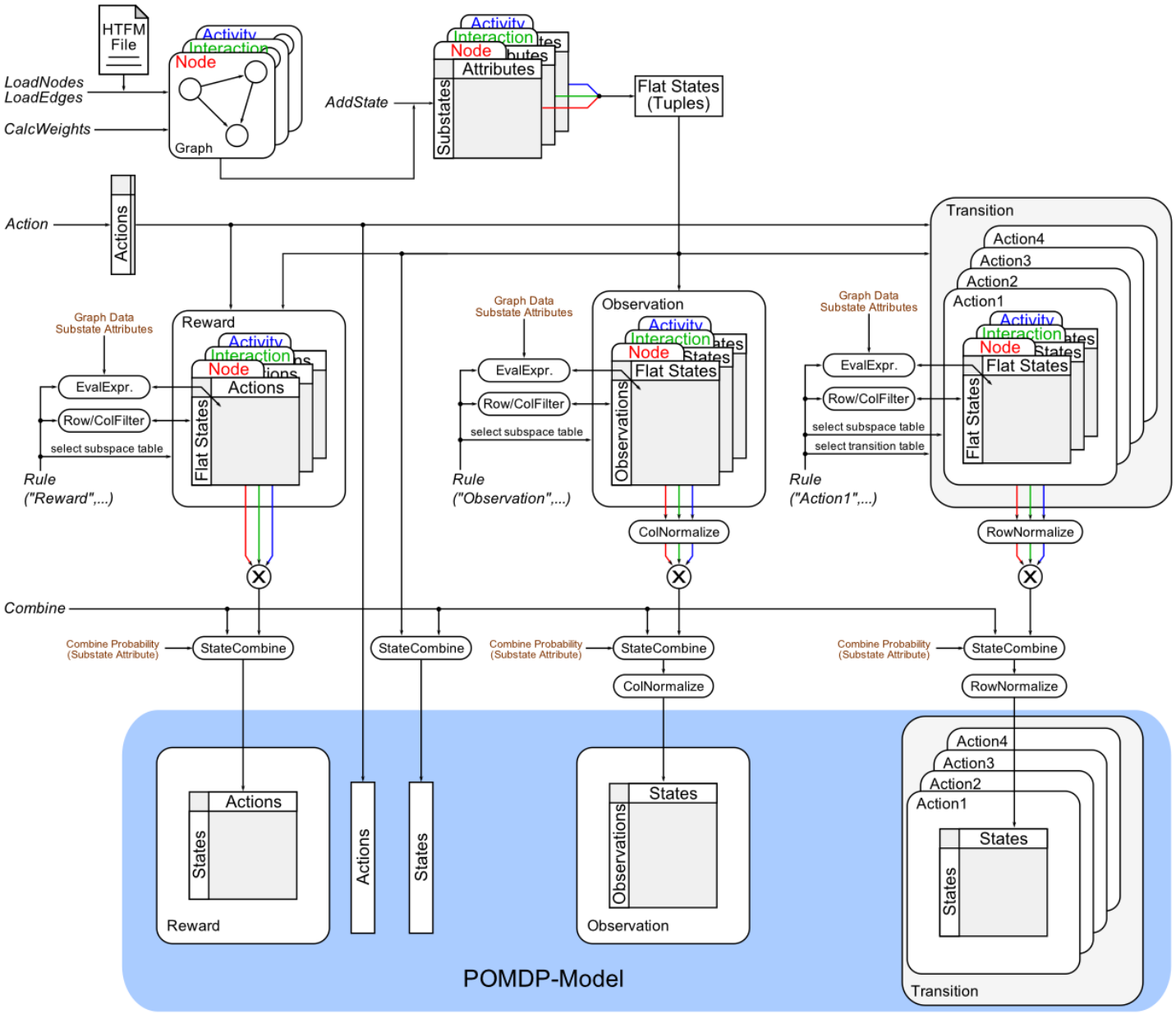


Figure B.1.: Schematic view of processing in the functional expression toolbox, presented in Section 3.6, showing exemplary features "Node", "Interaction" and "Activity". - [37]

Algorithm 2 Compute Separators for Feature State Regions

Input: Limit box l_j, $B_l := \{b_1, \ldots, b_l\}$ *inside* l_j

for all b_i **do** {Neighbor computation}

$\{b_{gt}[x_k] \in B_{gt_i} |\ b_{gt}[x_k] > b_i[x_k]\} :\ un_{x_k}(b_i) \leftarrow min(b_{gt}[x_k] \in B_{gt_i})$

$\{b_{lt}[x_k] \in B_{lt_i} |\ b_{lt}[x_k] < b_i[x_k]\} :\ ln_{x_k}(b_i) \leftarrow max(b_{lt}[x_k] \in B_{lt_i})$

end for

for all b_i **do** {Computing separator candidates $p_{x_k,i}$ with upper neighbors}

for all x_k **do** {$\sigma_{cluster}$ is defined on cluster $i \sim b_i$, $c_{cluster}$ is the center of a cluster i}

$p_{x_k,i} = \frac{\sigma_i}{(\sigma_i + \sigma_{un_{x_k}(i)})} * (c_{un_{x_k}(i)} - c_i)$

$P_{cand} \leftarrow p_{x_k,i}$

end for

end for

for all $p_{x_k,i} \in P_{cand}$ **do** {Checking candidate intersection with any other CEB}

for all b_i **do**

if $p_{x_k,i} \bigcap b_i$ **then**

remove $p_{x_k,i}$

end if

end for

end for

if $P_{cand} = \emptyset$ **then** {Checking empty candidate set}

shrink all b_i by Δ_X

Compute Separators(l_j)

end if

for all $p_{x_k,i} \in P_{cand}$ **do** {Computing separator candidate qualities}

$\omega_h =$

$sim(i,g) = \sum_k \frac{\|\sigma_{x_k,i} - \sigma_{x_k,g}\|}{\max(\sigma_{x_k,g}, \sigma_{x_k,g})} + \|shape(i) - shape(g)\|$

$Sim(B_g) = \frac{1}{|B_g|^2} \sum_{i \in B_g} \sum_{g \in B_g, g > i} sim(i,g)$

$\omega_{sim} = \min(Sim(B_{up}), Sim(B_{low}))$

$q(p_{x_k,i}) = \omega_h + \frac{\alpha}{\omega_{sim}}$

end for

$p^{opt}_{x_k,i} = max_{p_{x_k,i} \in P_{cand}} (\, q(p_{x_k,i}) \,)$

[continued]

[continued]

for all b_i **do** {Compute next tree children for recursion}

  if $b_i[x_k] < p^{opt}_{x_k,i}$ **then**

    $l_{lower} \leftarrow b_i$

  else

    $l_{upper} \leftarrow b_i$

  end if

end for

Compute Separators(l_{lower})

Compute Separators(l_{upper})

C. Examples

Examples, illustrating techniques, discussed in Chapters 3 and 4 are given in the following.

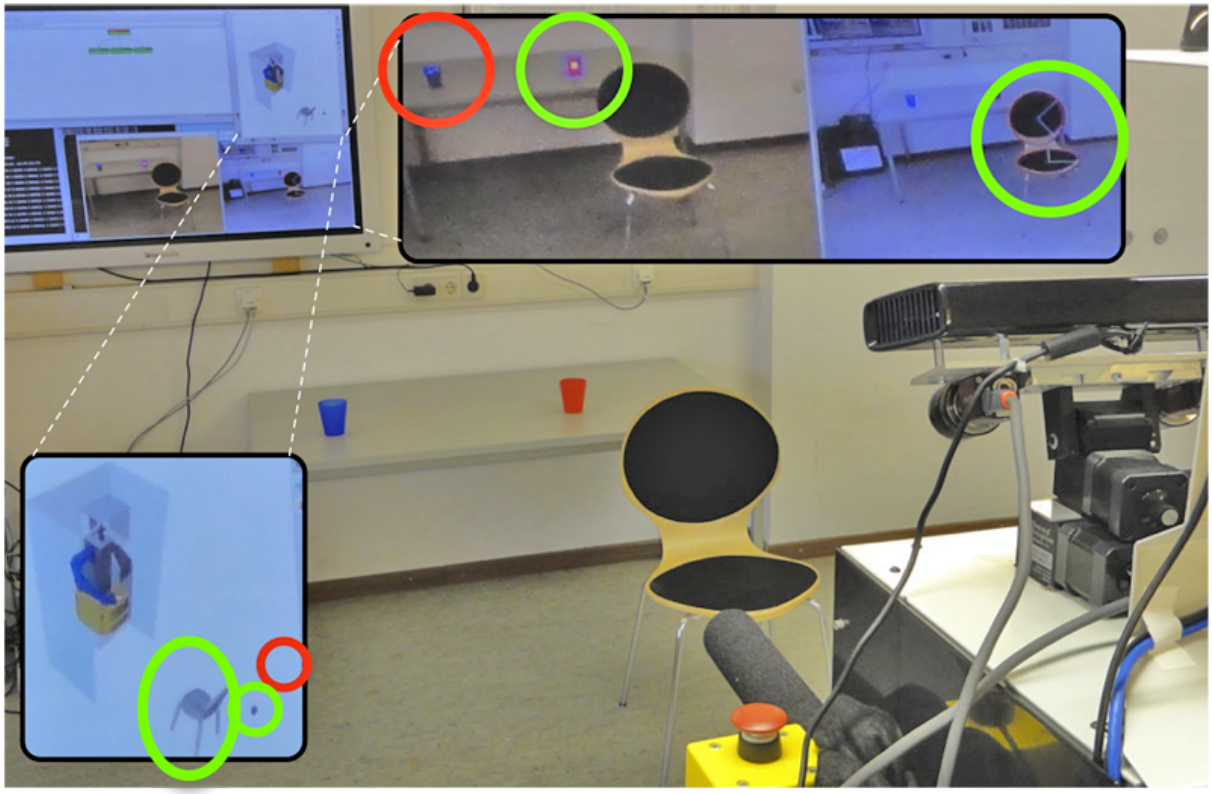


Figure C.1.: False negative among true positives during localization of various objects as discussed in Section 3.3.3. Note: the red cup is also shown blue in the motion planning visualization. - [125]

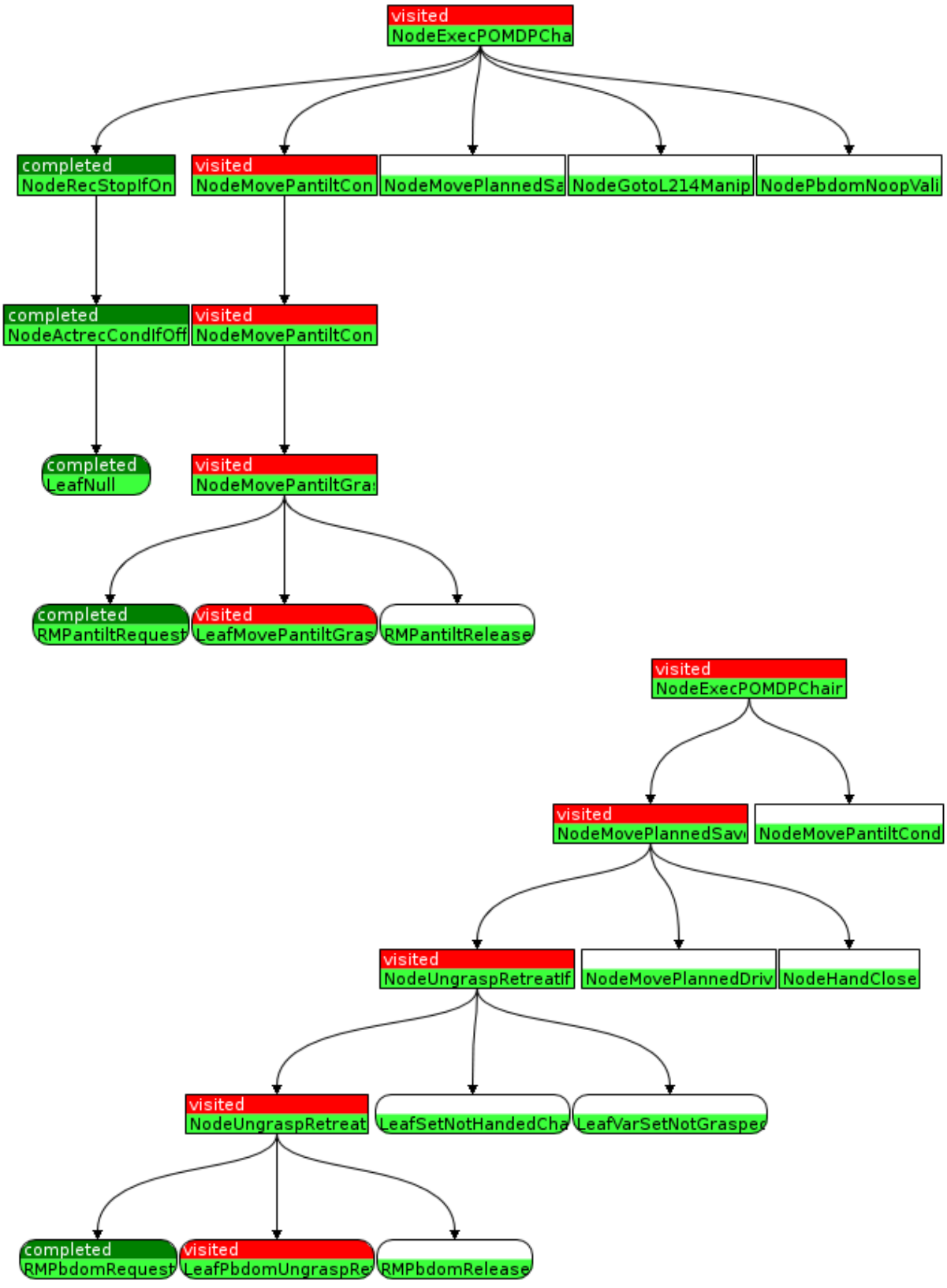


Figure C.2.: Further visualizations of FPs, executed in POMDP elementary actions in mission CESM-1 (see Section 5.1.2). "Goto" at thetop includes setting neck, hand and arm into a safe position for driving. "Ungrasp" at the bottom right includes moving into a safe position afterwards. - [125]

Table C.1.: *Example*: Simple single feature dependent transition.

$F_{1:robot-pose} = \{c_1^1 = WaitArea, c_2^1 = AtTable\}$,
$F_{2:small-obj-state} = \{c_1^2 = Noobject, c_2^2 = CupOnTable\}$,
$S = \{s_1 : c_1^1 \wedge c_1^2, s_2 : c_1^1 \wedge c_2^2, s_3 : c_2^1 \wedge c_1^2, s_4 : c_2^1 \wedge c_2^2\}$
$a_1 = gotoTable$, $p_T(c_1'^1 \mid c_1^1, a_1) = 0.1$, $p_T(c_2'^1 \mid c_1^1, a_1) = 0.9$
$\Rightarrow T(s_1, a_1) = (0.1, 0.0, 0.9, 0), T(s_2, a_1) = (0.0, 0.1, 0.0, 0.9)$

Explanation: The object state is conditionally independent from the mobility action. However, a transition has to be defined for origin states with both object states s_1, s_2. Therefore, for each mobility feature action effect probability, two flat transition model entries have to be assigned. With feature state spaces and feature sets much larger than in this example, such characteristics become more pronounced.

Table C.2.: *Example*: OPPL rule generating the catch-all feature state "OTHER" in $f_{robot-pose}$.

```
?act:INDIVIDUAL, ?s0:INDIVIDUAL, ?reg:INDIVIDUAL,
?trans:INDIVIDUAL, ?mod:CLASS,
?l:CONSTANT=MATCH("(.*?)(\\_*)?"),
?other:INDIVIDUAL=create(?l.GROUPS(1)+"OTHER")
SELECT ?act instanceOf Goto, ASSERTED ?act destRegion ?reg,
ASSERTED ?trans instanceOf GAGotoOther,
?trans instanceOf state0 some ?mod, ?trans destRegion ?reg,
ASSERTED ?mod subClassOf MdlState, ?mod subClassOf (RobLoc or RobOrient),
ASSERTED ?mod.IRI label ?l
BEGIN ADD ?other instanceOf ?mod, ADD ?other instanceOf StateOther,
ADD ?other.IRI label "OTHER" END;
```

Table C.3.: *Example*: OPPL rule generating the generic error feature state "ERROR" in $f_{robot-pose}$.

```
?act:INDIVIDUAL, ?reg:INDIVIDUAL, ?trans:INDIVIDUAL, ?mod:CLASS,
?l:CONSTANT=MATCH("(.*?)(\\s*)?"),
?other:INDIVIDUAL=create(?l.GROUPS(1)+"ERROR")
SELECT ?act instanceOf Goto, ASSERTED ?act destRegion ?reg,
ASSERTED ?trans instanceOf GAGotoCollision,
?trans instanceOf state0 some ?mod, ?trans destRegion ?reg,
ASSERTED ?mod subClassOf MdlState, ?mod subClassOf (RobLoc or RobOrient),
ASSERTED ?mod.IRI label ?l
BEGIN ADD ?other instanceOf ?mod, ADD ?other instanceOf StateError,
ADD ?other.IRI label "ERROR" END
```

Table C.4.: *Example*: DL axioms, representing manipulation action effect probabilities.

```
Transition(TrGraspChairFrontMissed)
Transition(TrGraspChairFrontAway)
manipulationStrategy(TrGraspChairFrontMissed, Front)
manipulationStrategy(TrGraspChairFrontAway, Front)
relatedObject(TrGraspChairFrontMissed, WoodenChair)
relatedObject(TrGraspChairFrontAway, WoodenChair)
effectStateSet(TrGraspChairFrontMissed, ObjStatePresentSet)
effectStateSet(TrGraspChairFrontAway, ObjStateInaccessibleSet)
Set(ObjStatePresentSet)
Set(ObjStateInaccessibleSet)
probability(TrGraspChairFrontMissed, 0.15)
probability(TrGraspChairFrontAway, 0.05)
```

Table C.5.: *Example*: SWRL rule for matching transitions onto new instances.

Grasp(?x0) ∧ Transition(?x1)
∧ relatedObject(?x0, ?x2) ∧ relatedObject(?x1, ?x2)
∧ manipulationStrategy(?x0, ?x3) ∧ manipulationStrategy(?x1, ?x3)
→ hasTransition(?x0, ?x1)

Table C.6.: *Example*: OPPL rule generating a manipulation generic error recovery action A_E.

?act:INDIVIDUAL, ?obj:INDIVIDUAL, ?trans:INDIVIDUAL, ?mod:CLASS,
?retract:INDIVIDUAL=create("SaveArmRetract")
SELECT ?act instanceOf Grasp, ?act relatedObject ?obj,
?trans instanceOf GraspTransDS, ?trans instanceOf (not DSManipJammed
and not DSManipSuccess), ASSERTED ?mod subClassOf MdlState,
?trans instanceOf state0 some ?mod, ?trans relatedObject ?obj,
?mod subClassOf (ObjLoc or ObjOrient)
BEGIN ADD ?retract instanceOf Release, ADD ?retract strategy SaveRetract END;

Table C.7.: *Example*: OPPL rule generating an information gain action A_I.

?goto:INDIVIDUAL, ?obj:INDIVIDUAL, ?manip:INDIVIDUAL,
?l:CONSTANT=MATCH("(.*?)(\\..*)?"),
?obs:INDIVIDUAL=create("ObserveNear"+?l.GROUPS(1))
SELECT ASSERTED ?goto instanceOf Goto, ASSERTED ?obj.IRI label ?l,
ASSERTED ?goto optimizedFor ?manip, ?goto relatedObject ?obj
BEGIN ADD ?obs instanceOf Observe, ADD ?obs relatedObject ?obj,
ADD ?obs optimizedFor ?manip END;

Table C.8.: *Example*: Sample number for grasping a chair from the front [163].

Gridsize for x, y = 10cm, θ in 30° steps.
$|x_i|$ = 15, $|y_i|$ = 21, $|\theta_k|$ = 12, samples per x-y fields: 41, Theta per x-y sample: 1:
(x, y, θ)-Voxel number: 15 * 21 * 12 = 3780
Samples: 15 * 21 * 12 * 41 = 154980
With an average of 5 seconds on a single CPU, this accounts to ∼9d.
By parallelization on two Quad-Core computers,
the MST-ORSALM can be computed over the course of two nights.
Higher densities of such uniform sampling are, however infeasible.

Table C.9.: *Example*: Merging primary and secondary action effect probabilities gathered by demonstration analysis (PbD) and geometric analysis (GA) process stages.

1) The robot can be near a RegionA where there is potentially a chair. In this case, there are potentially two interacting humans: human1, human2. The robot has the ability to grasp the chair:

$a = GraspChairFront$

$F_{1:robot-pose} = \{c_1^1 = AtRegionA, c_2^1 = Other\}$

$F_{2:furni-state} = \{c_1^2 = C_{(Chair)} \in RegionA, c_2^2 = C_{(Chair)} \notin RegionA,$
$c_3^2 = Grasped, c_4^2 = Jammed\}$

$F_{3:human1-pose} = \{c_1^3 = NearRegionA, c_2^3 = NotNearRegionA\}$

$F_{4:human2-pose} = \{c_1^4 = Present, c_2^4 = NotPresent\}$

$S = F_1 \times F_2 \times F_3 \times F_4$

Below, only states with the chair in RegionA and robot in AtRegionA are considered:

$s_{AtCw2Hums} := c_{AtRegionA}^{robot-pose} \wedge c_{C_{(Chair)} \in RegionA}^{furni-state} \wedge c_{NearRegionA}^{human1-pose} \wedge c_{Present}^{human2-pose}$

$s_{AtCwHum1} := c_{AtRegionA}^{robot-pose} \wedge c_{C_{(Chair)} \in RegionA}^{furni-state} \wedge c_{NearRegionA}^{human1-pose} \wedge c_{NotPresent}^{human2-pose}$

$s_{AtCwHum2} := c_{AtRegionA}^{robot-pose} \wedge c_{C_{(Chair)} \in RegionA}^{furni-state} \wedge c_{NotNearRegionA}^{human1-pose} \wedge c_{Present}^{human2-pose}$

$s_{AtCAlone} := c_{AtRegionA}^{robot-pose} \wedge c_{C_{(Chair)} \in RegionA}^{furni-state} \wedge c_{NotNearRegionA}^{human1-pose} \wedge c_{NotPresent}^{human2-pose}$

$\kappa(s_{humsDontMatter}) = s_{AtCw2Hums}$ or $s_{AtCwHum1}$ or $s_{AtCwHum2}$ or $s_{AtCAlone}$

$\kappa(s_{hum2DoesntMatter}) = s_{AtCw2Hums}$ or $s_{AtCwHum1}$

2) During demonstrations, two primary aspects are taught:

a) The presence of human2 is independent of anything else and has probability $p = 0.6$:

$p_{T_D}(c_{Present}^{human2-pose} | \kappa(s_{humsDontMatter}), a) = 0.6$

$p_{T_D}(c_{NotPresent}^{human2-pose} | \kappa(s_{humsDontMatter}), a) = 0.4$

b) If human1 is near the chair it may steal the chair while the robot tries to grasp it:

$p_{T_D}(c_{Grasped}^{furni-state} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.7$

$p_{T_D}(c_{C_{(Chair)} \notin RegionA}^{furni-state} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.3$

This leads to the following primary action effect joint probabilities:

$p_{T_D}(c_{Present}^{human2-pose} \wedge c_{Grasped}^{furni-state} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.6 * 0.7$

$p_{T_D}(c_{Present}^{human2-pose} \wedge c_{C_{(Chair)} \notin RegionA}^{furni-state} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.6 * 0.3$

$p_{T_D}(c_{NotPresent}^{human2-pose} \wedge c_{Grasped}^{furni-state} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.4 * 0.7$

$p_{T_D}(c_{NotPresent}^{human2-pose} \wedge c_{C_{(Chair)} \notin RegionA}^{furni-state} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.4 * 0.3$

(Continued)

(Continued)

3) Subsequently, during refinement, geometric analysis (GA) determines, motion planning will fail with $p = 0.2$, leading to the following secondary action effect probabilities:

$p_{T_{GA}}(c^{furni-state}_{Grasped} \mid \kappa(s_{humsDontMatter}), a) = 0.8$

$p_{T_{GA}}(c^{furni-state}_{C_{(Chair)} \in RegionA} \mid \kappa(s_{humsDontMatter}), a) = 0.2$

T_{GA} is anchored on $c^{furni-state}_{Grasped}$ in T_D, thus resulting in the following joint probabilities:

$p_{T_{D+GA}}(c^{human2-pose}_{Present} \wedge c^{furni-state}_{Grasped} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.6 * 0.7 * 0.8$

$p_{T_{D+GA}}(c^{human2-pose}_{Present} \wedge c^{furni-state}_{C_{(Chair)} \in RegionA} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.6 * 0.7 * 0.2$

$p_{T_{D+GA}}(c^{human2-pose}_{Present} \wedge c^{furni-state}_{C_{(Chair)} \notin RegionA} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.6 * 0.3$

$p_{T_{D+GA}}(c^{human2-pose}_{NotPresent} \wedge c^{furni-state}_{Grasped} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.4 * 0.7 * 0.8$

$p_{T_{D+GA}}(c^{human2-pose}_{NotPresent} \wedge c^{furni-state}_{C_{(Chair)} \in RegionA} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.4 * 0.7 * 0.2$

$p_{T_{D+GA}}(c^{human2-pose}_{NotPresent} \wedge c^{furni-state}_{C_{(Chair)} \notin RegionA} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.4 * 0.3$

Table C.10.: *Example*: Merging action effect probabilities gathered by demonstration analysis (PbD) and geometric analysis (GA) process stages with those computed by analysis of simulated trials (Sim).

Based on probabilities, computed by PbD and GA shown in Table C.9,
in the second refinement stage, dynamic simulation (Sim) determines
the following effect probabilities after successful motion planning:

$p_{T_{Sim}}(c^{furni-state}_{Grasped} \mid \kappa(s_{humsDontMatter}), a) = 0.5$ (Success)

$p_{T_{Sim}}(c^{furni-state}_{C_{(Chair)} \in RegionA} \mid \kappa(s_{humsDontMatter}), a) = 0.25$ (Missed)

$p_{T_{Sim}}(c^{furni-state}_{C_{(Chair)} \notin RegionA} \mid \kappa(s_{humsDontMatter}), a) = 0.15$ (Pushed away)

$p_{T_{Sim}}(c^{furni-state}_{Jammed} \mid \kappa(s_{humsDontMatter}), a) = 0.1$ (Unexpected forces)

T_{Sim} is anchored on $c^{furni-state}_{Grasped}$ in T_{D+GA}, leading
to the following joint probabilities:

$$p_{T_{D+GA+Sim}}(c^{human2-pose}_{Present} \wedge c^{furni-state}_{Grasped} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.6 * 0.7 * 0.8 * 0.5$$

$$p_{T_{D+GA+Sim}}(c^{human2-pose}_{Present} \wedge c^{furni-state}_{C_{(Chair)} \in RegionA} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.6 * 0.7 * 0.2 + 0.6 * 0.7 * 0.8 * 0.25$$

$$p_{T_{D+GA+Sim}}(c^{human2-pose}_{Present} \wedge c^{furni-state}_{C_{(Chair)} \notin RegionA} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.6 * 0.3 + 0.6 * 0.7 * 0.8 * 0.15$$

$$p_{T_{D+GA+Sim}}(c^{human2-pose}_{Present} \wedge c^{furni-state}_{Jammed} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.6 * 0.7 * 0.8 * 0.1$$

$$p_{T_{D+GA+Sim}}(c^{human2-pose}_{NotPresent} \wedge c^{furni-state}_{Grasped} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.4 * 0.7 * 0.8 * 0.5$$

$$p_{T_{D+GA+Sim}}(c^{human2-pose}_{NotPresent} \wedge c^{furni-state}_{C_{(Chair)} \in RegionA} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.4 * 0.7 * 0.2 + 0.4 * 0.7 * 0.8 * 0.25$$

$$p_{T_{D+GA+Sim}}(c^{human2-pose}_{NotPresent} \wedge c^{furni-state}_{C_{(Chair)} \notin RegionA} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.4 * 0.3 + 0.4 * 0.7 * 0.8 * 0.15$$

$$p_{T_{D+GA+Sim}}(c^{human2-pose}_{NotPresent} \wedge c^{furni-state}_{Jammed} \mid \kappa(s_{hum2DoesntMatter}), a) = 0.4 * 0.7 * 0.8 * 0.1$$

It can be noticed, e.g. $c^{furni-state}_{C_{(Chair)} \notin RegionA}$ resulting from two different causes:
some probability accounts from human1 stealing the chair, some probability accounts
from the robot accidentally pushing the chair away during failed grasp efforts.

D. Fundamentals

In the following, some fundamental computer science methods, discussed in Chapter 2 and utilized in Chapters 3 and 4 are explained more closely.

D.1. MDP Value Iteration

Basically, the MDP value iteration calculates the expected probability of a sequence of events times its sum of rewards. Combining the values of all possible sequences following a certain action being performed in a certain state, a utility U for that action can be computed. Because the agent will always choose the action with the highest utility in a certain state, that utility $max(U(s_i, A))$ can be assigned directly to the state s_i. That utility thus represents the sum of rewards the agent expects to collect when acting optimally after being in state s_i.

Yet, this approach leads to an interlocked problem: the utility of a state $U(s)$ can only be computed when the optimal policy to choose actions in a state is known while $U(s)$ is needed to compute the optimal policy. Interlocked problems can be tackled by iterative computation techniques and for MDP policy computation two techniques exist: policy iteration and value iteration.

Value iteration starts with an arbitrary policy and utility assignment $U(S)$. Then, relations between neighbouring states in the transition model are calculated which represent sequences of events which lead from one state to the other. The probability of the transition is multiplied with its immediate reward and the utility of the resulting state [11]:

$$U(s) = \gamma \, max_u \left(R(s,u) + \sum_{s'} T(s', u, s) U(s') \right) \qquad \text{[D.1]}$$

By these means, with each iteration, longer potential sequences of subsequent events are incorporated in the utility value of each state. Accordingly, the choice of the the best action to be performed in that state iteratively considers ever longer series of time steps. To be able to consider potentially infinite series of actions, a discount factor $\gamma < 1.0$ is introduced. For each further time step, each utility is multiplied with γ and thus discounted. This leads to events further in the future to be of lesser importance to $U(s)$ - and therefore the action choice

- than more immediate potential events. It also allows the utility computation to converge asymptotically as is forms a geometric series.

D.2. Clustering Methods

There are two ways using clustering: determining which points belong to which cluster with a fixed number of clusters k given on the one hand and automatic identification of both cluster number k as well as point membership on the other. The latter can be achieved either by computing clustering for relevant ranges of k, followed by computation of quality metrics for each k or by methods calculating both aspects concurrently. Both approaches are sketched in the following and applied in PMPM-PbD as discussed in Section 4.2.2.

k-means is an iterative technique for k clusters L_i with center c_i and points x, minimizing euclidean distance $d(x, c_i)$ [92]:

$$E = \sum_{i=1}^{k} \sum_{x \in L_i} d(x, c_i) \qquad \text{[D.2]}$$

Weaknesses are point initialization dependence, fixed k, spherical clusters because of euclidean distance and strong impact of outliers as indicated in Figure 2.18.

Expectation Maximization (EM) estimates data points by a mixture of k Gaussians. EM shares most disadvantages with k-means but using Gaussians has advantages compared to an unweighted euclidean distance as can be seen in Figure 2.18.

Fixed k quality measures compute a clustering quality value for a cluster number k on a given set of data points. Two properties of clustering are regarded when computing such measures: separability between and compactness within clusters.

Davies Bouldin Index (DB index) computes a cluster similarity measure M_{ij}, based on compactness p_i and separability d_{ij} [31]:

$$d_{ij} = d(c_i, c_j),\ p_i = \frac{1}{\| L_i \|} \sum_{x \in L_i} d(x, c_i) \quad [D.3]$$

$$R_{ij} = \frac{p_i + p_j}{d_{ij}} \quad [D.4]$$

$$R_i = \max_{j=1\ldots k, i \neq j} (R_{ij}), i = 1 \ldots k \quad [D.5]$$

$$DB = \frac{1}{k} \sum_{i=1}^{k} R_i \quad [D.6]$$

The smaller the DB index, the better, with the cluster number k having the smallest value indicating the optimal number of clusters.

SD-Index computes a relationship between total data variance $\sigma(X)$ and cluster variance $\sigma(L_i$ [53]:

$$Scat(k) = \frac{1}{k} \sum_{i=1}^{k} \frac{\| \sigma(L_i) \|}{\| \sigma(X) \|} \quad [D.7]$$

$$Dis(k) = \frac{\max_{i,j=1\ldots k} d(c_i, c_j)}{\min_{i,j=1\ldots k} d(c_i, c_j)} \sum_{i=1}^{k} (\sum_{j=1, i \neq j}^{k} d(c_i, c_j))^{-1} \quad [D.8]$$

$$SD = \alpha * Scat + Dis \quad [D.9]$$

The cluster number k with the smallest SD value represents optimal clustering.

Xie-Beni index (XB index) weighs cluster variance σ as compactness measure with separability d_{min} and fuzzy factor u [165]:

$$\sigma = \sum_{i=1}^{k} \sum_{j=1}^{N} u_{ij}^2 d(x_j, c_i)^2 \quad [D.10]$$

$$d_{min} = \min_{i,j=1\ldots,k, i \neq j} d(c_i, c_j)^2 \quad [D.11]$$

$$XB(k) = \frac{\sigma}{k * d_{min}} \quad [D.12]$$

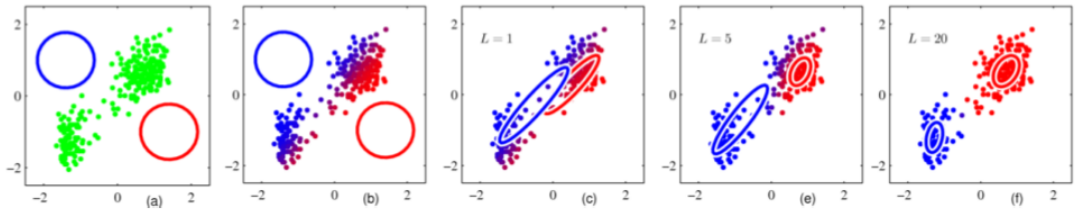


Figure D.1.: EM iteration after 0, 1, 5 and 20 steps. - [14]

Again, the smallest XB value denotes the optimal k.

DBScan is a density based clustering method with a continuous quality measure, no fixed k and able to find clusters of arbitrary size [44]. However, a parameter has to be found for a data set: $ce := min|x_{central}|$, defining the minimum number of points around a cluster center within a ε-distance.

The optimal cluster number k can be found as a result of the largest ε-distance range resulting in the same k. Typical results are depicted in Figure 2.18.

D.3. Gaussian Mixture Models

A GMM is defined on a D-dimensional manipulation space, approximating a set of sample points representing trajectories by K normal distributions with weights ω_k:

$$p_{mixture} = \sum_{k=1}^{K} \omega_k N(x|\mu_k, \Sigma_k^2) \qquad [D.13]$$

As in EM-clustering, iterative expectation maximization [14] is used to compute each mean μ and co-variance Σ as well as the weight vector $\vec{\omega}$ optimally approximating the data set. Being an iterative technique, parameters $\mu_k, \Sigma_k, \omega_k$ are either assigned randomly or using an initial heuristic, e.g. k-means clustering. Subsequently, parameters are computed iteratively based on data point samples $X := x_1, \ldots, x_n$ taken from a set of input trajectories in a D-dimensional manipulation space. An iteration step consist of an estimation stage and a maximization stage. In the estimation step, the likelihood $\gamma_{n,k}$ of x_n being covered by gaussian k is computed. Based on $\gamma_{n,k}$, new parameters $\mu_k, \Sigma_k, \omega_k$ are computed in the maximization stage. A schematic view can be seen in Figure D.1.

By these means, the log-likelihood $lnp(X)$ of the GMM on the set of data points is increased in each step:

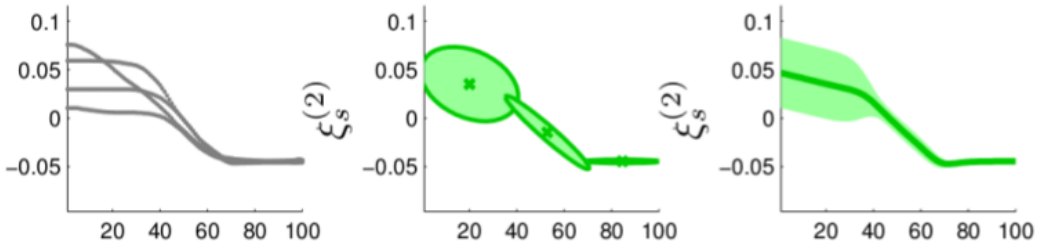


Figure D.2.: GMR tube around a trajectory boundle represented by a GMM. - [42]

$$ln\ p(X) = \sum_{n=1}^{N} ln\left(\sum_{k=1}^{K} \omega_k N_k(x_n)\right) \qquad [D.14]$$

Iteration terminates when change in log-likelihood of current parameters $\mu_k, \Sigma_k, \omega_k$ falls under an externally defined threshold $\varepsilon > 0$.

Based on a GMM representation, a Gaussian Mixture Regression (GMR) can be performed which provides an easily accessible interface for getting mean μ and co-variance Σ corresponding to a trajectory data point [28]. Consequently, a GMR forms a probabilistic, unbounded tube around a trajectory bundle it represents as depicted in Figure D.2.

E. Research Collaboration and Prior Publications

Most concepts of parts of Chapters 3, 4 and 5 have been published previously – as cited in the respective Sections and Figures – as conference papers or students' theses. Those publications are the result of collaborative research and discussion that crucially included the author of this thesis. The following, exhaustive list of publications and student theses includes all these essential prior publications. In those publications, the author of this thesis was either the corresponding author, a co-author, or the main thesis-advising researcher, crucially steering the project as well as participating in the formulation of problems and solutions, in discussions, visualization and experiments. This thesis is novel above and beyond those publications in presenting the whole concept and process in a coherent, refined manner.

- The execution-time architecture for autonomy and *filter*POMDP system, outlined in Sections 3.1 and 5.2, was developed and evaluated with support of Steffen Knoop, Martin Lösch, and Zhixing Xue [128], [133].

- Modeling human-robot interaction in POMDP mission models, discussed in Section 3.3, was developed with assistance of Martin Lösch and Steffen Knoop [129], [130].

- Modeling uncertainty in furniture localization, outlined in Section 3.3.4, was developed with Pascal Meissner in the scope of his furniture localization system [97], [98].

- Modeling tasks as elementary abstract actions as well as general mission modeling, see Sections 3.4 and 3.5, builds on some initial joint analysis with Manuel Leuschner [87].

- The value iteration policy visualization concept *StarViz*, described in Secion 3.5.2, was jointly devised with Johannes Pelzer [105].

- The functional expression toolbox, enabling focussed generation of complex POMDP models, presented in Section 3.6, was conceived with Gerhard Dirschl and Rainer Jäkel [127], [37].

- The description logic based background knowledge inference approach, presented in Sections 3.7 and 4.10, was devised jointly with Gerhard Dirschl [124].

- Development of PbD state mapping, see Section 4.2, was a collaboration with Jonas Stahl [147].
- PbD action mapping, outlined in Section 4.3, was developed jointly with Thorsten Engelhardt [42].
- PbD segmentation and mapping, mentioned in Sections 4.4, 4.6 and 5.5, were supported in different development stages by Tobias Utz, Fabian Romahn, Martin Lösch and Rainer Jäkel [157], [132], [131], [117].
- Smoothing of demonstration sequences, discussed in Section 4.5, was implemented by Jessica Kaufmann [74].
- A joint collaboration with Fabian Romahn and Gerhard Kurz led to generalization and request generation, discussed in detail in Sections 4.7, 4.8 and 5.6,[134], [117], [81].
- Model refinement by means of geometric analysis, described in Section 4.11, was developed with Christian Wischnewski in the scope of his path-planning framework and supported by Rainer Jäkel regarding MST-ORSALM [163].
- Dynamics simulation based model refinement, discussed in Sections 4.12 and 5.8, was a joint effort with Laurenz Berger and Martin Seidel, using OpenRAVE und ODE extensions developed by Laurenz Berger [12], [135].
- The utilized OViSE visualization, mentioned in Section 5.1, was a development by Alexander Kasper, Thorsten Mai and an initial effort by Johannes Pelzer [106], [73].
- CESM mission design, comprehensive service experiments and visualization thereof, outlined in Section 5.1, was an extensive joint effort with Gerhard Dirschl, Fabian Romahn, Thorsten Mai, Laurenz Berger, Tim Friedrich, Christian Wischnewski, Martin Seidel, Jonas Stahl, supported by Rainer Jäkel, Pascal Meissner, Alexander Kasper, Martin Lösch, Steffen Rühl, and Andreas Hermann [125].
- Infrastructure regarding the robot Albert and corresponding software management was crucially supported by Gerhard Dirschl.

It has to be noted that software implementation of jointly developed formalisms, models, and algorithms was sometimes, but not always, a joint effort. Further publications and theses with a less essential share in crucial aspects of the presented concept, as well as publications regarding human full body activity classification and manipulation strategy Programming by

Demonstration, that arose in collaboration with the author of this thesis, were cited in Chapter 2, when applicable.

- The author of this thesis supported development of small-object tracking developed by Pascal Meissner [96].
- Development of human full body activity classification, mentioned in Section 2.7.1, was supported [90], [91].
- Manipulation strategy development was initiated by supervising a thesis of Rainer Jäkel and supported subsequently [66], [65], [64], [67].

F. List of Figures

G. List of Tables

H. Bibliography

[1] D. Aberdeen. Policy-Gradient Methods for Planning. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[2] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*, 1998.

[3] J. Albiez and R. Dillmann. Behaviour Networks for Walking Machines - a Design Method. In *International Conference on Climbing and Walking Robots (CLAWAR)*, 2004.

[4] G. Antoniou and F. van Harmelen. *Handbook on Ontologies*, chapter Web Ontology Language: OWL. Springer, 2002.

[5] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A Survey of Robot Learning from Demonstration. *Robotics and Autonomous Systems*, 2009.

[6] T. Asfour, K. Regenstein, P. Azad, J. Schröder, N. Vahrenkamp, and R. Dillmann. ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. In *International Conference on Humanoid Robots (Humanoids)*, 2006.

[7] K. J. Aström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 1965.

[8] P. Azad, T. Asfour, and R. Dillmann. Combining Apperance-based and Model-based Methods for Real-Time Object Recognition and 6D Localization. In *International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[9] F. Baader, I. Horrocks, and U. Sattler. Description Logics. *Handbook on Ontologies*, 2009.

[10] C. Barck-Holst, M. Ralph, F. Holmar, and D. Kragic. Learning Grasping Affordance Using Probabilistic and Ontological Approaches. In *International Conference on Advanced Robotics (ICAR)*, 2009.

[11] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[12] L. Berger. Erstellung probabilistischer Entscheidungsmodelle für Objektmanipulation eines Serviceroboters durch Testläufe in Dynamiksimulation. Master's thesis, Studienarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2011.

[13] A. Billard, Y. Epars, S. Calinon, G. Cheng, and S. Schaal. Discovering Optimal Imitation Strategies. *Journal of Robotics and Autonomous Systems*, 2004.

[14] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[15] A. W. Black et al. The Festival Speech Synthesis System, 2012.

[16] S. A. Blum. From a CORBA-Based Software Framework to a Component-Based System Architecture for Controlling a Mobile Robot. In *International Conference of Computer Vision Systems (ICVS)*, 2003.

[17] C. Borst, M. Fischer, and G. Hirzinger. Grasp Planning: How to Choose a Suitable Task Wrench Space. In *International Conference on Robotics and Automation (ICRA)*, 2004.

[18] C. Borst, T. Wimbock, F. Schmidt, M. Fuchs, B. Brunner, F. Zacharias, P. R. Giordano, R. Konietschke, W. Sepp, S. Fuchs, C. Rink, A. Albu-Schaffer, and G. Hirzinger. Rollin' Justin - Mobile Platform with Variable Base. In *International Conference on Robotics and Automation (ICRA)*, 2009.

[19] M. Brenner and B. Nebel. Continual Planning and Acting in Dynamic Multiagent Environments. In *International Symposium on Practical Cognitive Agents and Robots*, 2006.

[20] R. Brooks. A Robust Layered Control System for a Mobile Robot. *Journal of Robotics and Automation*, 1986.

[21] R. Brooks. Intelligence Without Representation. *Journal of Artificial Intelligence*, 1991.

[22] R. E. Caflisch. Monte Carlo and quasi-Monte Carlo methods. *Acta Numerica, Cambridge University Press*, 1998.

[23] S. Calinon and A. Billard. A Probabilistic Programming by Demonstration Framework Handling Skill Constraints in Joint Space and Task Space. In *International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[24] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting Optimally in Partially Observable Stochastic Domains. In *National Conference on Artificial Intelligence*, 1994.

[25] A. Chia, M. Leung, H. Eng, and S. Rahardja. Ellipse Detection with Hough Transform in One Dimensional Parametric Space. In *International Conference on Image Processing (ICIP)*, 2007.

[26] D. Choi, Y. Kang, H. Lim, and B.-J. You. Knowledge-based Control of a Humanoid Robot. In *International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[27] J. Choi and E. Amir. Combining Planning and Motion Planning. In *Cognitive Robotics*, 2010.

[28] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active Learning with Statistical Models. *Journal of Artificial Intelligence Research*, 1996.

[29] S. G. I. Corp. Open Inventor, 2012.

[30] E. Coste-Maniere and R. Simmons. Architecture, the Backbone of Robotic Systems. In *International Conference on Robotics and Automation (ICRA)*, 2000.

[31] D. Davies and D. Bouldin. A Cluster Separation Measure. *Transactions on Pattern Analysis and Machine Intelligence*, 1979.

[32] P. Deckers, A. M. Dollar, and R. D. Howe. Guiding Grasping with Proprioception and Markov Models. In *Workshop on Robot Manipulation: Sensing and Adapting to the Real World, RSS Conference*, 2007.

[33] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, 2010.

[34] J. S. Dibangoye, G. Shani, B. Chaib-Draa, and A.-I. Mouaddib. Topological Order Planner for POMDPs. In *International Joint Conference on Artifical Intelligence (IJCAI)*, 2009.

[35] R. Dillmann, T. Asfour, M. Do, R. Jäkel, A. Kasper, P. Azad, A. Ude, S. Schmidt-Rohr, and M. Lösch. Advances in Robot Programming by Demonstration. *KI - Künstliche Intelligenz*, 2010.

[36] R. Dillmann, M. Kaiser, and A. Ude. Acquisition of Elementary Robot Skills from Human Demonstration. In *International Symposium on Intelligent Robotics Systems*, 1995.

[37] G. Dirschl, R. Jäkel, and S. R. Schmidt-Rohr. POMDP ModelBuilder. Technical report, Institut für Anthropomatik, Karlsruher Institut für Technologie (KIT), 2012.

[38] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel. Integrating Symbolic and Geometric Planning for Mobile Manipulation. In *International Workshop on Safety, Security and Rescue Robotics (SSRR)*, 2009.

[39] A. Edsinger and C. Kemp. Manipulation in Human Environments. In *International Conference on Humanoid Robotics (Humanoids)*, 2006.

[40] S. Ekvall, D. Aarno, and D. Kragic. Task Learning Using Graphical Programming and Human Demonstrations. In *International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2006.

[41] Y. Endo. Anticipatory Robot Control for a Partially Observable Environment Using Episodic Memories. In *International Conference on Robotics and Automation (ICRA)*, 2008.

[42] T. Engelhardt. Automatische Zuordnung von detaillierten Manipulationsstrategien auf entsprechende Teilaktionen von Missionsvorführungen bei Programmieren durch Vormachen von autonomen Servicerobotern. Master's thesis, Studienarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2011.

[43] C. Eppner, J. Sturm, M. Bennewitz, C. Stachniss, and W. Burgard. Imitation Learning with Generalized Task Descriptions. In *International Conference on Robotics and Automation (ICRA)*, 2009.

[44] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining*, 1996.

[45] R. Fikes and N. Nilsson. STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 1971.

[46] M. A. Fischler and R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, 1987.

[47] A. Foka and P. Trahanias. Real-time Hierarchical POMDPs for Autonomous Robot Navigation. *Journal of Robotics and Autonomous Systems*, 2007.

[48] G. Forney. The Viterbi Algorithm. In *IEEE*, 1973.

[49] E. Gat. On Three-layer Architectures. *Artificial Intelligence and Mobile Robots*, 1997.

[50] A. Genz. Numerical Computation of Rectangular Bivariate and Trivariate Normal and t Probabilities. *Statistics and Computing*, 2004.

[51] T. L. Griffiths, E. R. Baraff, and J. B. Tenenbaum. Using Physical Theories to Infer Hidden Causal Structure. *Conference of the Cognitive Science Society*, 2004.

[52] T. L. Griffiths, C. Kemp, and J. B. Tenenbaum. Bayesian Models of Cognition. *Cambridge Handbook of Computational Cognitive Modeling*, 2008.

[53] M. Halkidi, M. Vazirgiannis, and Y. Batistakis. Quality Scheme Assessment in the Clustering Process. *Principles of Data Mining and Knowledge Discovery*, 2000.

[54] I. Hirschman Jr. A Note on Entropy. *American Journal of Mathematics*, 1957.

[55] P. Hitzler and B. Parsia. Ontologies and Rules. In *Handbook on Ontologies*. Springer, 2009.

[56] J. Hoey, A. von Bertoldi, P. Poupart, and A. Mihailidis. Assisting Persons with Dementia during Handwashing Using a Partially Observable Markov Decision Process. In *International Conference on Vision Systems (ICVS)*, 2007.

[57] I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *International Conference on World Wide Web (WWW '04)*, 2004.

[58] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 2004.

[59] R. A. Howard. *Dynamic Programming and Markov Processes*. PhD thesis, M.I.T., 1960.

[60] K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez. Grasping POMDPs. In *International Conference on Robotics and Automation (ICRA)*, 2007.

[61] K. Hsiao, T. Lozano-pérez, and L. P. Kaelbling. Robust Belief-based Execution of Manipulation Programs. In *Workshop on the Algorithmic Foundations of Robotics*, 2008.

[62] L. Iannone, M. E. na, A. Rector, and R. Stevens. Augmenting the Expressivity of the Ontology Pre-Processor Language. In *Fifth OWLED Workshop on OWL: Experiences and Directions*, 2009.

[63] M. Imaging. SR4000 User Manual and API Documentation, 2009.

[64] R. Jaekel, S. Schmidt-Rohr, M. Loesch, A. Kasper, and R. Dillmann. Learning of Generalized Manipulation Strategies in the Context of Programming by Demonstration. In *International Conference on Humanoid Robots (Humanoids)*, 2010.

[65] R. Jaekel, S. R. Schmidt-Rohr, M. Loesch, and R. Dillmann. Representation and Constrained Planning of Manipulation Strategies in the Context of Programming by Demonstration. In *International Conference on Robotics and Automation (ICRA)*, 2010.

[66] R. Jäkel. Aus Beobachtung des Menschen gelernte probabilistische Objektmanipulation durch Serviceroboter. Master's thesis, Diplomarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2008.

[67] R. Jäkel, P. Meißner, S. Schmidt-Rohr, and R. Dillmann. Distributed Generalization of Learned Planning Models in Robot Programming by Demonstration. In *International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[68] R. Jaulmes, J. Pineau, and D. Precup. A Formal Framework for Robot Learning and Control Under Model Uncertainty. In *International Conference on Robotics and Automation (ICRA)*, 2007.

[69] S. Jayaraman and C. North. A Radial Focus+Context Visualization for Multi-Dimensional Functions. In *IEEE Computer Society Conference on Visualization*, 2002.

[70] B. Jensen, G. Froidevaux, X. Greppin, A. Lorotte, L. Mayor, M. Meisser, G. Ramel, and R. Siegwart. The Interactive Autonomous Mobile System RoboX. In *International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[71] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Journal of Artificial Intelligence*, 1998.

[72] G. A. Kaminka and I. Frenkel. Integration of Coordination Mechanisms in the BITE Multirobot Architecture. In *International Conference on Robotics and Automation (ICRA)*, 2007.

[73] A. Kasper and T. Mai. OViSE, 2012.

[74] J. Kaufmann. Glättung von Beobachtungssequenzen mittels HMMs für Programmieren durch Vormachen von probabilistischer Entscheidungsfindung. Master's thesis, Studienarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2011.

[75] C. Kemp, A. Perfors, and J. B. Tenenbaum. Learning Domain Structures. In *International Conference on Cognitive Science*, 2004.

[76] S. Knoop, M. Pardowitz, and R. Dillmann. From Abstract Task Knowledge to Executable Robot Programs. *Journal of Intelligent and Robotic Systems: Theory and Applications. Robotics, Virtual Reality, and Agents and their Body: A Special Issue in Memory of Marco Somalvico*, 2007.

[77] S. Knoop, S. R. Schmidt-Rohr, and R. Dillmann. A Flexible Task Knowledge Representation for Service Robots. In *International Conference on Intelligent Autonomous Systems (IAS)*, 2006.

[78] S. Knoop, S. Vacek, and R. Dillmann. *Human Body Model for Articulated 3D Pose Tracking*, chapter 28. International Journal of Advanced Robotic Systems, Special Issue on Humanoid Robots, 2007.

[79] K. Konolige, K. L. Myers, E. H. Ruspini, and A. Saffiotti. The Saphira Architecture: A Design for Autonomy. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, 1997.

[80] H. Kurniawati, D. Hsu, and W. Lee. SARSOP: Efficient Point-based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Robotics: Science and Systems (RSS)*, 2008.

[81] G. Kurz. Erzeugung von Lehranforderungen bei Programmieren durch Vormachen probabilistischer Entscheidungsfindung autonomer Serviceroboter. Master's thesis, Studienarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2011.

[82] T. Lang and M. Toussaint. Relevance Grounding for Planning in Relational Domains. *Machine Learning and Knowledge Discovery in Databases*, 2009.

[83] T. J. Lang. *Planning and Exploration in Stochastic Relational Worlds*. PhD thesis, Technische Universität Berlin, 2011.

[84] J. Lebowitz. Boltzmann's Entropy and Time's Arrow. *Physics Today*, 1993.

[85] M. D. Lee. A Hierarchical Bayesian Model of Human Decision-Making on an Optimal Stopping Problem. *Journal of Cognitive Science*, 2006.

[86] S. Lemaignan, R. R., M. L., A. R., and M. Beetz. ORO, a Knowledge Management Module for Cognitive Architectures in Robotics. In *International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[87] M. Leuschner. Systematischer Entwurf von probabilistischen Entscheidungsmodellen für Missionsszenarien von Haushaltsrobotern. Master's thesis, Studienarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2010.

[88] H. Li, X. Liao, and L. Carin. Region-based Value Iteration for Partially Observable Markov Decision Processes. In *International Conference on Machine Learning (ICML)*, 2006.

[89] M. Lösch. *Erkennung menschlicher Aktivitäten zur Belehrung von Robotern*. PhD thesis, Karlsruhe Institute of Technology, 2012.

[90] M. Lösch, S. Schmidt-Rohr, S. Knoop, S. Vacek, and R. Dillmann. Feature Set Selection and Optimal Classifier for Human Activity Recognition. In *Robot and Human interactive Communication (RO-MAN)*, 2007.

[91] M. Lösch, S. R. Schmidt-Rohr, and R. Dillmann. Making Feature Selection for Human Motion Recognition More Interactive Through the Use of Taxonomies. In *International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2008.

[92] J. MacQueen et al. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, 1967.

[93] P. C. Mahalanobis. On the Generalised Distance in Statistics. In *National Institute of Science, India*, 1936.

[94] L. Maisonobe. Quick Computation of the Distance Between a Point and an Ellipse. Technical report, Spaceroots, 2006.

[95] M. J. Matarić. Behavior-based Robotics as a Tool for Synthesis of Artificial Behavior and Analysis of Natural Behavior. *Trends in Cognitive Sciences*, 1998.

[96] P. Meissner. Integration multisensoriell gewonnener Bilddaten zur modellbasierten Objektverfolgung in Punktwolken. Master's thesis, Studienarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2008.

[97] P. Meissner. Teile- und konturorientierte Klassifizierung von Tischen und Stühlen samt Lageschätzung mittels multisensoriell gewonnener Tiefendaten zur autonomen Robotermanipulation. Master's thesis, Diplomarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2010.

[98] P. Meißner, S. R. Schmidt-Rohr, M. Lösch, R. Jäkel, and R. Dillmann. Robust Localization of Furniture Parts by Integrating Depth and Intensity Data Suitable for Range Sensors with Varying Image Quality. In *International Conference on Advanced Robotics (ICAR)*, 2011.

[99] M. Mühlig, M. Gienger, and J. J. Steil. Human-Robot Interaction for Learning and Adaptation of Object Movements. In *International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[100] A. Nilsson, R. Muradore, K. Nilsson, and P. Fiorini. Ontology for Robotics: a Roadmap. In *International Conference on Advanced Robotics (ICAR)*, 2009.

[101] S. Ong, S. Png, D. Hsu, and W. Lee. POMDPs for Robotic Tasks with Mixed Observability. In *Robotics: Science and Systems (RSS)*, 2009.

[102] M. Pardowitz, R. Zollner, and R. Dillmann. Learning Sequential Constraints of Tasks from User Demonstrations. In *International Conference on Humanoid Robots (Humanoids)*, 2005.

[103] C. Parlitz, M. Haegele, P. Klein, J. Seifert, , and K. Dautenhahn. Care-obot 3 - Rationale for Human-robot Interaction Design. In *International Symposium on Robotics*, 2008.

[104] P. Patrón, E. Miguelañez, Y. R. Petillot, and D. M. Lane. Fault Tolerant Adaptive Mission Planning with Semantic Knowledge Representation for Autonomous Underwater Vehicles. In *International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[105] J. Pelzer. Visualisierung von Multidimensionalen Entscheidungsfunktionen. Master's thesis, Studienarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2008.

[106] J. Pelzer. Visualisierung probabilistischer Entscheidungsfindung im Szenenkontext. Master's thesis, Diplomarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2009.

[107] R. A. Peters, II, K. Kawamura, D. M. Wilkes, K. A. Hambuchen, T. E. Rogers, and et al. ISAC Humanoid: An Architecture for Learning and Emotion. In *International Conference on Humanoid Robots (Humanoids)*, 2001.

[108] J. Pineau, G. Gordon, and S. Thrun. Point-based Value Iteration: An Anytime Algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[109] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun. Towards Robotic Assistants in Nursing Homes: Challenges and Results. *Journal of Robotics and Autonomous Systems*, 2003.

[110] J. Pineau and S. Thrun. High-level Robot Behavior Control Using POMDPs. In *In AAAI Workshop notes*, 2002.

[111] J. M. Porta, N. Vlassis, M. T. Spaan, and P. Poupart. Point-Based Value Iteration for Continuous POMDPs. *Journal of Machine Learning Research*, 2006.

[112] Primesense. Nite - the Natural Interaction Middleware from PrimeSense, 2011.

[113] M. Pucher, A. Türk, J. Ajmera, and N. Fecher. Phonetic Distance Measures for Speech Recognition Vocabulary and Grammar Optimization. In *3rd congress of the Alps Adria Acoustics Association*, 2007.

[114] M. Quigley, E. Berger, and A. Y. Ng. STAIR: Hardware and Software Architecture. In *AAAI Robotics Workshop*, 2007.

[115] L. R. Rabiner and B. H. Juang. An Introduction to Hidden Markov Models. *IEEE Signal Processing Magazine*, 1986.

[116] G. Rizzolatti and L. Craighero. The Mirror-Neuron System. *Annual Review of Neuroscience*, 2004.

[117] F. Romahn. Generierung probabilistischer Entscheidungsmodelle mit Generalisierung von Handlungswissen bei Programmieren durch Vormachen von autonomen Servicerobotern. Master's thesis, Studienarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2011.

[118] S. Ross, B. Chaib-draa, and J. Pineau. Bayes-Adaptive POMDPs. In *Neural Information Processing Systems*, 2007.

[119] S. Rudolph and M. K. und Pascal Hitzler. All Elephants are Bigger than All Mice. In *International Workshop on Description Logics*, 2008.

[120] S. Ruehl, A. Hermann, Z. Xue, T. Kerscher, and R. Dillmann. Graspability: A Description of Work Surfaces for Planning of Robot Manipulation Sequences. In *International Conference on Robotics and Automation (ICRA)*, 2011.

[121] M. Ruhnke. Unüberwachtes Lernen von 3D Modellen für nicht stationäre Objekte auf volumetrischen Daten. Master's thesis, Albert-Ludwigs-Universität Freiburg, 2008.

[122] S. Russell and P. Norvig. *Artifcial Intelligence - A Modern Approach*. Pearson Education, 2 edition, 2003.

[123] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Control, Planning, Learning, and Imitation with Dynamic Movement Primitives. In *Workshop on Bilateral Paradigms on Humans and Humanoids, International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[124] S. R. Schmidt-Rohr, G. Dirschl, P. Meissner, and R. Dillmann. A Knowledge Base for Learning Probabilistic Decision Making from Human Demonstrations by a Multimodal Service Robot. In *International Conference on Advanced Robotics (ICAR)*, 2011.

[125] S. R. Schmidt-Rohr, G. Dirschl, F. Romahn, T. Mai, L. Berger, T. Friedrich, C. Wischnewski, M. Seidel, J. Stahl, and R. Dillmann. Project Albert Service Experiments (PASE). Technical report, Karlsruhe Institute of Technology (KIT), IFA, 2012.

[126] S. R. Schmidt-Rohr, R. Jäkel, and R. Dillmann. Vorlesung Robotik II - Programmieren von Robotern. Lecture Slides and Exercise Sheets, Karlsruhe Institute of Technology, 2010.

[127] S. R. Schmidt-Rohr, R. Jäkel, M. Lösch, and R. Dillmann. Compiling POMDP Models for a Multimodal Service Robot from Background Knowledge. In *EUROS Conference*, 2008.

[128] S. R. Schmidt-Rohr, S. Knoop, M. Lösch, and R. Dillmann. Bridging the Gap of Abstraction for Probabilistic Decision Making on a Multi-modal Service Robot. In *Robotics: Science and Systems (RSS)*, 2008.

[129] S. R. Schmidt-Rohr, S. Knoop, M. Lösch, and R. Dillmann. Reasoning for a Multi-Modal Service Robot Considering Uncertainty in Human-robot Interaction. In *Conference on Human-Robot Interaction (HRI)*, 2008.

[130] S. R. Schmidt-Rohr, M. Lösch, and R. Dillmann. Human and Robot Behavior Modeling for Probabilistic Cognition of an Autonomous Service Robot. In *International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2008.

[131] S. R. Schmidt-Rohr, M. Lösch, and R. Dillmann. Learning Flexible, Multi-modal Human-robot Interaction by Observing Human-human-interaction. In *International Symposium in Robot and Human Interactive Communication (RO-MAN)*, 2010.

[132] S. R. Schmidt-Rohr, M. Lösch, R. Jäkel, and R. Dillmann. Programming by Demonstration of Probabilistic Decision Making on a Multi-modal Service Robot. In *International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[133] S. R. Schmidt-Rohr, M. Lösch, Z. Xue, and R. Dillmann. Hardware and Software Architecture for Robust Autonomous Behavior of a Domestic Robot Assistant. In *International Conference on Intelligent Autonomous Systems (IAS)*, 2008.

[134] S. R. Schmidt-Rohr, F. Romahn, P. Meissner, R. Jäkel, and R. Dillmann. Learning Probabilistic Decision Making by a Service Robot with Generalization of User Demonstrations and Interactive Refinement. In *International Conference on Intelligent Autonomous Systems (IAS)*, 2012.

[135] M. Seidel. Lernen aus Erfahrung in Dynamiksimulation für einen Serviceroboter auf Basis von Programmieren durch Vormachen. Master's thesis, Bachelorarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2012.

[136] A. P. Shon, J. J. Storz, and R. P. N. Rao. Towards a Real-Time Bayesian Imitation System for a Humanoid Robot. In *International Conference on Robotics and Automation (ICRA)*, 2007.

[137] A. P. Shon, D. Verma, and R. P. N. Rao. Active Imitation Learning. In *National Conference on Artificial intelligence*, 2007.

[138] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. *Computer Vision and Pattern Recognition (CVPR)*, 2011.

[139] R. Simmons, R. Goodwin, K. Z. Haigh, S. Koenig, and J. O'Sullivan. A Layered Architecture for Office Delivery Robots. In *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, 1997.

[140] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2007.

[141] T. Smith and R. Simmons. Heuristic Search Value Iteration for POMDPs. In *Uncertainty in Artificial Intelligence (UAI)*, 2004.

[142] T. Smith and R. Simmons. Point-based POMDP Algorithms: Improved Analysis and Implementation. In *Uncertainty in Artificial Intelligence (UAI)*, 2005.

[143] R. Smith et al. Open Dynamics Engine, 2012.

[144] E. J. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Stanford university, 1971.

[145] M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 2005.

[146] M. Sridharan, J. Wyatt, and R. Dearden. HiPPo: Hierarchical POMDPs for Planning Information Processing and Sensing Actions on a Robot. In *The International Conference on Automated Planning and Scheduling (ICAPS)*, 2008.

[147] J. Stahl. Modellierung und automatische Zustandsraumdiskretisierung für interaktives Lernen probabilistischer Entscheidungsmodelle von Servicerobotern. Master's thesis, Masterarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2011.

[148] S. Streeting et al. OGRE3D, 2012.

[149] A. L. Strehl, C. Diuk, and M. L. Littman. Efficient Structure Learning in Factored-State MDPs. In *AAAI*, 2007.

[150] F. Stulp, A. Fedrizzi, F. Zacharias, M. Tenorth, J. Bandouch, and M. Beetz. Combining Analysis, Imitation, and Experience-based Learning to Acquire a Concept of Reachability in Robot Mobile Manipulation. In *International Conference on Humanoid Robots (Humanoids)*, 2009.

[151] I. H. Suh, G. H. Lim, W. Hwang, H. Suh, J. H. Choi, and Y. T. Park. Ontology-based Multi-layered Robot Knowledge Framework (OMRKF) for Robot Intelligence. In *International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[152] M. Tenorth and M. Beetz. KnowRob — Knowledge Processing for Autonomous Personal Robots. In *International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[153] M. Tenorth and M. Beetz. Priming Transformational Planning with Observations of Human Activities. In *International Conference on Robotics and Automation (ICRA)*, 2010.

[154] S. Thrun. Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

[155] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. The MIT Press, 2005.

[156] S. G. Toya. *Detection of Structured Objects with a Range Camera*. PhD thesis, Eidgenössische Technische Hochschule Zürich, 2008.

[157] T. Utz. Aufzeichnung, Segmentierung und Analyse menschlicher Tätigkeiten zur Programmierung eines Serviceroboters. Master's thesis, Studienarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2009.

[158] H. Veeraraghavan and M. M. Veloso. Learning Task Specific Plans Through Sound and Visually Interpretable Demonstrations. In *International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[159] M. Veloso, J. Carbonell, A. Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 1995.

[160] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel. Sphinx-4: A Flexible Open Source Framework for Speech Recognition. Technical report, SUN Microsystems, 2004.

[161] E. J. Wamsley, M. Tucker, J. D. Payne, J. A. Benavides, and R. Stickgold. Dreaming of a Learning Task Is Associated with Enhanced Sleep-Dependent Memory Consolidation. *Current Biology*, 2010.

[162] J. D. Williams, P. Poupart, and S. Young. Using Factored Partially Observable Markov Decision Processes with Continuous Observations for Dialogue Management. Technical report, Cambridge University Engineering Department Technical Report: CUED/F-INFENG/TR.520, 2005.

[163] C. Wischnewski. Arbeitsraumbezogene Analyse der Wahrscheinlichkeiten von Aktionseffekten für probabilistisches Entscheiden eines autonomen Roboters. Master's thesis, Diplomarbeit, KIT, Betreuer: Sven Schmidt-Rohr, Gutachter: Rüdiger Dillmann, 2012.

[164] D. Wolpert. Probabilistic Models in Human Sensorimotor Control. *Human Movement Science*, 2007.

[165] X. Xie and G. Beni. A Validity Measure for Fuzzy Clustering. *Transactions on Pattern Analysis and Machine Intelligence*, 13, 1991.

[166] L. Yu and H. Liu. Efficient Feature Selection via Analysis of Relevance and Redundancy. *Journal of Machine Learning Research*, 5, 2004.

[167] F. Zacharias and C. B. und Gerd Hirzinger. Capturing Robot Workspace Structure: Representing Robot Capabilities. In *International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[168] R. Zöllner, M. Pardowitz, S. Knoop, and R. Dillmann. Towards Cognitive Robots: Building Hierarchical Task Representations of Manipulations from Human Demonstration. In *International Conference on Robotics and Automation (ICRA)*, 2005.