Hochschule
Bonn-Rhein-Sieg
Bonn-Rhein-Sieg University

Fachbereich Informatik
Department of Computer Science

Master's Thesis

Master of Science in Computer Science

Affordance-Based Action Abstraction in Robot Planning

Daniel Höller

Date of Submission:
October 24, 2013

First advisor : Prof. Dr.-Ing. Gerhard K. Kraetzschmar
Bonn-Rhein-Sieg University
Department of Computer Science

Second advisor : Prof. Dr. rer. nat. Paul G. Plöger
Bonn-Rhein-Sieg University
Department of Computer Science

External advisor : Dr.-Ing. Erich Rome
Fraunhofer Institute for Intelligent
Analysis and Information Systems IAIS

Für Nadja.

Abstract

This work extends the affordance-inspired robot control architecture introduced in the MACS project [35] and especially its approach to integrate symbolic planning systems given in [24] by providing methods to automated abstraction of affordances to high-level operators. It discusses how symbolic planning instances can be generated automatically based on these operators and introduces an instantiation method to execute the resulting plans.

Preconditions and effects of agent behaviour are learned and represented in Gärdenfors' *conceptual spaces* framework. Its notion of similarity is used to group behaviours to abstract operators based on the affordance-inspired, function-centred view on the environment. Ways on how the capabilities of conceptual spaces to map subsymbolic to symbolic representations to generate PDDL planning domains including affordance-based operators are discussed.

During plan execution, affordance-based operators are instantiated by agent behaviour based on the situation directly before its execution. The current situation is compared to past ones and the behaviour that has been most successful in the past is applied. Execution failures can be repaired by action substitution. The concept of using contexts to dynamically change dimension salience as introduced by Gärdenfors is realized by using techniques from the field of *feature selection*.

The approach is evaluated using a 3D simulation environment and implementations of several object manipulation behaviours.

Contents

List of Tables

List of Figures

List of Algorithms

Acknowledgements

I would like to thank Prof. Dr. Gerhard K. Kraetzschmar, Prof. Dr. Paul G. Plöger and Dr. Erich Rome for supervising this thesis and giving me the opportunity to write it in such an inspiring area of research.

Beside my advisors, I would particularly like to thank Iman Awaad for giving me the chance to do my thesis in the context of her Ph. D. project and for contributing central ideas to the topic of this work.

1. Introduction

An *affordance* is an interaction possibility of an agent with its environment. It is directly perceivable by the agent and thus does not require object recognition. The term *affordance* is originally from the field of ecological psychology and was introduced by James J. Gibson.

> "The *affordances* of the environment are what it *offers* the animal, what it *provides* or *furnishes*, either for good or ill. The verb to *afford* is found in the dictionary, but the noun *affordance* is not. I have made it up. I mean by it something that refers to both the environment and the animal in a way that no existing term does. It implies the complementarity of the animal and the environment." [16, p. 127]

The MACS project introduced a new way to transfer this concept to the field of robotics, enabling a more flexible interaction of robots with their environment [34, 35]. A main difference to other work has been the decision to reason about the offered affordances, based on a representation called *affordance triple*. Lörken and Hertzberg contributed a way to combine affordance-based robot control with PDDL planning to implement goal-directed behaviour [24]. It benefits from the affordance-based approach by avoiding symbol grounding: the affordance-based approach inherently uses no environment model, thus grounding for model building is avoided. During plan execution, they ground actions in an affordance triple to avoid the second direction of symbol grounding. Any suitable entity that provides the affordance that is looked for is used.

Beforehand, affordance triples are grouped (by the system designer) into so-called *abstract affordances*. When e. g. the agent can lift two kinds of entities, let's say red and green ones (maybe also using different behaviours), there is more than one triple, but only one abstract affordance *liftable*. Based on this abstract affordance, planning operators are modelled (by hand) and used for planning. If there is an action *lift* in the plan, all triples belonging to this abstract affordance are retrieved and the agent searches the environment for a cue belonging to one of them. The first one that is found is exploited by applying the corresponding behaviour.

The idea of grouping triples to form abstract affordances and to use these for (symbolic) planning gives the agent the chance to take advantage of the increasing performance of state-of-the-art planning systems. It further introduces a way to abstract from concrete agent behaviour during planning and thus to have fewer operators and reduce the planning effort.

The late commitment to a specific entity providing an affordance helps to increase the flexibility and robustness of agent behaviour; both benefits are introduced by the affordance-based approach.

> "The idea behind this is that there are situations where one wants to achieve something but one does not actually care about how or with the help of what object to reach the desired goal." [35, p. 185]

In this thesis, these advantages are transferred from environment entities to agent behaviour. One question is which affordance, and thus which behaviour, is used if the agent has the choice. The second question is whether it is possible to generate the abstract affordances without (additional) human modelling, i. e. based on the behaviour representation. This would automate the abstraction step before planning.

If the goal is to *move* (planning operator) a *movable* (abstract affordance) entity, this can be done by using *push*, *pull*, *pick & place*... (agent behaviour). In this *action instantiation* step, a 'good choice' is made by choosing a behaviour that has been successful for similar situations in the past, also providing a second or third choice that can be tried if the first one fails (action substitution in case of behaviour execution failure).

In line with the function-centred view on the environment the idea for the abstraction step is to cluster affordances based on their effect, i. e. behaviours causing similar change to the environment belong to the same abstract affordance (the question is *what* to achieve and not *how* to do it). If the agent can do this clustering autonomously, this is one step towards a closed cycle of learning new behaviours, e. g. by demonstration or even playing, and applying them to achieve goal-directed behaviour. However, for closing this cycle, an autogeneration of the planning domain is also necessary, but is not within the scope of this thesis.

The work is done in context of the Ph. D. project of Iman Awaad at Bonn-Rhein-Sieg University that is introduced in [2]. Its goal is to use the affordance concept to increase the flexibility of robot behaviour, demonstrated in the following scenarios.

- **Object substitution** is the classical example for the use of the affordance concept. Some (maybe not present) kind of object is substituted by another one that offers a similar affordance, e .g. a *mug* by a *cup* to drink coffee.

- **Object substitution** as tool usage. Objects might be used for tasks they have not been intended for in the first place. On a windy day, one might put a cup on the paper to prevent it from flying away.

- **Object substitution or use as performance enhancement.** An agent that can reason not only about the category of an object but about its functional properties might be able to come up with clever substitutions that enhance solution quality.

- **Action substitution.** The grouping of behaviours that result in similar change to the environment and their flexible instantiation might enable an agent to recover from action-execution failures by choosing another instantiation to substitute the failed action. The thesis in hand contributes to this scenario.

In this thesis, two methods are developed. One to generate abstract affordances and one for action instantiation that decides based on a planned action (e. g. *move*) and the current sensory input which agent behaviour is used (e. g. *push*, *pull*,...). It will not only provide a first choice, but also others that can be tried if the first try fails. For the abstraction and the instantiation, similarity has to be estimated,

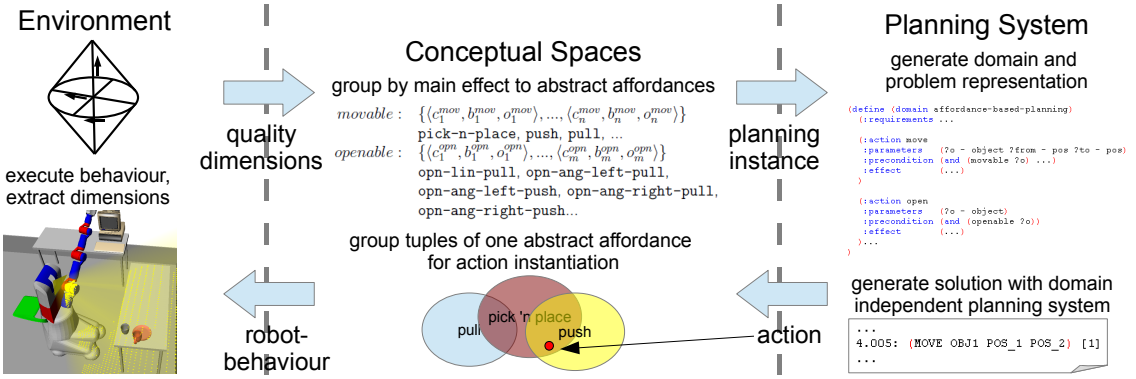


Figure 1: Affordance-based action abstraction in robot planning. The system is composed of three parts. The (subsymbolic) environment shown on the left side, a (symbolic) deliberation layer on the right; and a conceptual layer mediating. The arrows show data flow between different layers.

similarity of environment entities for the former, similarity of agent behaviour for the later. An approach that can be used for similarity calculation is Gärdenfors' *conceptual spaces* framework [12]. It measures similarity by distance in a special vector space and thus also provides a computationally handable approach. Though it has primarily been used to represent static entities, it has been extended to represent actions and their effects in the last years [13, 3, 5]. A so-called *event* combines representations of agent (the *actor*), *patient* (the entity an action is applied to) and the caused change (the so-called *displacement vector*). Both methods developed in this work will be based on a conceptual spaces representation of agent behaviour.

This work contributes a flexible, situation-adequate behaviour usage to the field of affordance-based robot control. It realizes a further step towards a closed cycle of autonomous learning and application of new agent behaviour. The abstraction step before action planning reduces the number of operators for planning.

The thesis is divided into the following parts. Section 2 introduces automated planning (Sec. 2.1), affordance-based robot control (Sec. 2.2), conceptual spaces (Sec. 2.3) and feature selection (Sec. 2.4). Afterwards Section 3 introduces the intended system that is given in Figure 1. Based on an affordance representation in conceptual spaces (defined in Sec. 3.2), abstract affordances are generated (Sec. 3.3.1). A planning instance is created (not part of this work, but discussed in Sec. 3.3.2) and solved by some planning system. The resulting plan contains affordance-based actions that are instantiated by the method given in Sec. 3.3.3. The behaviour is executed, in this work using an OpenRAVE [7] prototype that is described in Section 4. Quality dimensions are extracted from simulation and stored in conceptual spaces. The described approach is evaluated in Section 5 and concluded in Section 6.

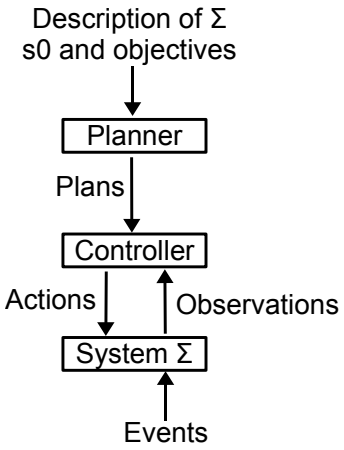


Figure 2: Conceptual model of planning (source: [15, p. 8])

2. Related Work

As described in the introduction, this thesis combines topics from several areas of research that are introduced in this section. It starts with a short introduction of *automated planning* in Sec. 2.1; introduces the affordance-inspired robot control architecture developed in the MACS projects in Sec. 2.2 and especially its use of automated planning. Afterwards Gärdenfors' conceptual spaces are introduced (Sec. 2.3) with focus on action and event representation. As techniques from the field of *feature selection* are used to find most relevant dimensions for a specific decision (context), feature selection is sketched in Sec. 2.4.

2.1. Automated Planning

If the behaviour of an agent shall go beyond a pure reaction to stimuli of the environment, it has to reason about change of its environment and about how it is able to change the environment by itself. If it is able to predict the applicability and effects of its doings, it can try to realize a desired goal by changing the environment with adequate actions. This "...explicit deliberation process that chooses and organizes actions by anticipating their expected outcomes." [15, p. 1] is called planning. A commonly used model of planning (e. g. described in [15, p. 8] or [28, p. 47]) is given in Figure 2. It consists of three main parts: a *planner* (or *planning system*) that gets a formal description of the environment Σ, the initial state of the environment (commonly called s_0) and the agents objectives (or goals) G. It chooses a set of actions with some ordering constraints that, when applied to the environment, presumably realizes the agent's objectives (called the *plan* or *solution*). The application is done by the *controller* that also gets a feedback from the environment in form of observations. The third part of the model is the actual system that is described by Σ. Following

[15, Sec. 1.4], Σ is defined to be a tuple

$$\Sigma = (S, A, E, \gamma)$$

that consists of a set of possible states of the environment S, a set of actions A that can be applied by the agent, a set of events E that also change the state of the system but are not controlled by the agent and a function γ that maps a state of the system, an action and an event to a set of possible outcome states.

$$\gamma : S \times A \times E \to 2^S$$

The overall model is commonly split up into a *domain definition* and a *problem definition.*

The *(planning) domain* definition describes the environment and contains definitions of the possible states and operators to change it. In general, it is stable for different problems. It might be mentioned that the term domain is also used in context of Gärdenfors' conceptual spaces framework. If the context is unclear, a prefix (planning or pl. as well as conceptual spaces or cs.) will be used.

A *(planning) problem* defines a concrete problem to solve. It gives the definition of s_0 and a (partial) definition of a goal state to reach. Due to the partial definition of the final state, it is likely that there is more than one goal state. To compare the quality of different goal states, the planning problem may also contain some metric definition.

A *(planning) instance* is the combination of a domain and a (suitable) problem and gives a planning system all information it needs to generate a solution.

A *(planning) solution (or plan)* is a set of actions together with an ordering relation defined on it that belongs to a planning instance. An *action* is a fully instantiated operator. In the simplest case, the action can be ordered as a sequence. When the actions are applied to the initial state defined in the problem, with respect to the ordering relation and using the transition model given in the domain, a state that fulfils the criteria of a goal state is reached.

Although the concrete implementations of planning systems is not relevant for this work, a schema is given that classifies the systems based (1) on the problems they can be applied to as well as (2) the knowledge that is provided in the model and used to finding a solution. The schema is following [28, p. 47-53]. According to it, there are the following three main groups of planning systems:

1. *Domain-specific Planning Systems* are implemented to solve a special class of problems. This limitation in applicability enables the introduction of a large amount of domain knowledge by the human designer and (may) result in efficient plan generation and high quality plans.

2. The other extreme are *Domain-independent Planning Systems* that get not only a problem, but also a domain definition and are (in general) able to solve every problem in every domain that is provided in an appropriate definition language.

A widely used definition language that will be introduced later in this section is the *Planning Domain Definition Language* (PDDL). Domain-independent planners normally do not have any information on how to solve the planning problems of a specific domain. For example, the PDDL specification states that the language has to "...express the physics of a domain..." and "...to provide no advice at all..." [25, both p. 1]. The planning systems have to implement universal heuristics to find a goal state instead. In state-of-the-art planning systems, these work quite well. However, there are surprisingly simple domains that stress the currently used heuristics[1].

3. The third group of planning systems is somewhere in between. It includes systems that work on a richer model that contains not only the physics of the domain, but also advice on how to solve planning problems. Those can be provided in several forms, there are systems working on rules that describe states that can be pruned from search space (see e. g. *badSituation*-rules in situation calculus described e. g. in [32]). A widely used technique is *Hierarchical Task Network* (HTN) Planning (e. g. described in [15, Chap. 11]) that provides rules on how to decompose abstract tasks into concreter ones, until all can be applied directly. For examples of HTN systems, see e. g. the SHOP [27] or the PANDA system [38]. Common to domain-configurable systems is the advice on how to solve problems that is provided in the domain model.

One attempt of this thesis is to do a further step towards a learned planning domain. Therefore there will be no domain knowledge beyond environment change observed by the agent. I. e. the agent has just physical knowledge, inferred by induction on training examples; and no knowledge on how to solve planning problems. The most adequate planning approach in such a scenario is the one that needs least domain knowledge – domain-independent planning. There will be a short introduction on PDDL in the remainder of this section.

But even domain-independent planning systems need some model and thus somehow do not fit into affordance-based approach. At this point, this work follows the approach introduced in the MACS project and described by Rome *et. al.* to

> "...reason about affordances instead of acting directly upon an affordance percept. [The agent] ...is not controlled by the environment. It can rather use the information provided by the affordance of a situation and reason about them in a goal-directed manner selecting those afforded actions that will lead to its goal." [35, p. 184]

The MACS project and its approach to affordance-based robot control and especially its approach to integrate planning systems is introduced later in this section.

If the parts of the presented system are classified into the conceptual model given in Fig. 2, the abstraction step prepares the formulation of the operators for the

[1] see e. g. the *Visit-All* domain that comes with many competing goals:
www.plg.inf.uc3m.es/ipc2011-deterministic/DomainsSequential.html#Visit-all
www.plg.inf.uc3m.es/ipc2011-deterministic/attachments/DomainsSequential/visit-all-doc.pdf

description of Σ based on the observations of the controller. The encoding is not part of this thesis but shortly discussed. An interesting variation from standard work flow in planning is the creating of the domain for a specific problem (like introduced in [18]). The conceptual layer helps also to create the input for the planning system. The planning step is intended to be executed by an external planning system. The instantiation is already part of the execution and belongs to the controller.

The Planning Domain Definition Language. Due to the regular planning contests at the *International Planning Competition* (IPC), performance of domain-independent planning systems increased during the past years. The used description language PDDL has been extended steadily. The initially STRIPS-like language (described in [25]) enabled the definition of a planning domain, including a type hierarchy, predicates used to describe a world's state and operators that can be used to cause change. Operators are composed of parameters, precondition and effects. The parameters specify all objects that are used in the precondition and effects. A separate definition gives the planning problem that consists of the domain it belongs to, a set of objects and the definitions of the initial state as well as a (partial) definition that goal states have to fulfill. A plan is a simple sequence of actions that transfer the initial state to some goal state when they are applied.

Fox and Long extended the language in [9] by numeric functions that can be used in addition to the predicates to describe the states of the world. This is a quite meaningful extension, since it makes the former finite search-space infinite. Using a numeric metric, it is further possible to rank goal states not just by plan length but by the quality of the final state. A second major extension given in [9] is the option to define temporal planning operators in the following ways. A first option is to split the action in a discrete start- and end-point. For each of them as well as the interval between, conditions can be defined that have to hold. Effects can be defined for the start and end of the operator. There is another way of introducing time into the planning process given in [9, Sec. 5.3]. It enables the definition of continuous effects that could be used e. g. to define the loading of the battery of a robot. It may be emphasized that the introduction of durative actions changes the appearance of a plan significantly. Planned actions are no longer executed in sequence, but may be in parallel to other actions. However, numeric and temporal planning form one step towards planning real world problems.

Another interesting feature for real world domains is the ability to define timed initial literals, introduced in [8]. These are properties that are set at some point in time. A popular example to demonstrate its usefulness is the definition of opening hours of a shop.

[14] extends the language by hard and soft constraints that can be defined on the plan. In the blocks world, one could define that towers always look like traffic lights (as a hard constraint). Or that it is preferable to use both hands of a robot to solve the task.

An overview of the language as well as further literature is given on the web page of the IPC [21].

Although there is an increasing number of well-working state-of-the-art planning systems, most of them support only a subset of the PDDL features, so the system that fits the needs of the problem at hand has to be selected carefully.

The integration of an out-of-the-box planning system enables a quite simple realization of goal directed agent behaviour. But when using a symbolic planner, there are also several drawbacks. The maybe worst disadvantage when comparing to purely reactive approaches is the need to define a symbolic model for planning based on the subsymbolic sensory input. Gärdenfors proposes the *conceptual layer* beneath the symbolic layer that can be used for similarity estimation and symbol grounding. It will be introduced in Section 2.3.

2.2. Affordance-based Robot Control

Gibson's term of an affordance as *directly perceivable interaction possibility of the environment to a (specific) agent* offers valuable concepts to the field of robotics. In many situations the way an 'object' can be used might be more important than its identity or the class it belongs to. In the original interpretation, the affordances' obviousness makes any model needless. However, this view may result in a mere direct perception-action coupling.

> "The use of affordances within Autonomous Robotics is mostly confined to behaviour-based control of the robots, and that its use in deliberation remains a rather unexplored area." [35, p. 178][2]

The work in this thesis is based on an approach developed in the *Multi-sensory Autonomous Cognitive Systems* (MACS) project [11] that introduces a (spare) model of the perceived affordances to include them in the deliberation process. The approach is introduced in the remainder of this section.

The MACS project investigated how a robot control-architecture can benefit from the affordances' function-centred view, not only by a direct perception-action mapping, but by plan-based robot control founded on perceived action possibilities [35, p. 180]. The project's main goal has been a more flexible and robust acting of robot [35, p. 174]. To implement goal-directed behaviour, state-of-the-art planning systems, like introduced in the last section, have been integrated into the system.

This combination is further discussed in the next section. Before the reasoning about affordances is introduced, the term itself has to be specified. Following Rome *et. al.*, an affordance is defined as

Definition 1 *"An (agent) affordance is a relation between an agent and its environment which affords a capability. The agent/environment relation affords a capability if the agent*

[2] cf. [12, p. 122]: "The fundamental cognitive role of concepts is to serve as a bridge between perceptions and actions. In the simplest case, where there is a direct coupling between perception and action, a concept can be seen as a decision procedure where the perception is categorized and the chosen category then determines a choice of action. ... In humans and many of the higher animals, cognition is dominated by other more indirect models that are not triggered by perceptions alone."

1. *has the capacity to recognize that it is in such a relation between itself and its environment, and it*

2. *has the ability to act to bring about that capability." [35, Def. 1]*

The affordances are represented by a triple with the following elements: a *cue* defines features of the environment that indicate the existence of the affordance. The *behaviour* describes the agent's doings. Its outcome is defined in the *outcome descriptor*.

Definition 2 *"(Affordance Representation). An affordance representation or affordance triple is a data structure:*

(cue descriptor, behavior descriptor, outcome descriptor)

Here, a cue descriptor or an outcome descriptor is specified as a list of attribute value pairs. A behavior descriptor consists of one or more behavior identifiers. Optionally, parameters for these behaviours can be specified." [35, Def. 2]

The cue and outcome descriptor thereby is not necessarily given by a single value, but can also be a range of values [35, p. 184f]. Cue and outcome are not limited to be offered from a single object (in general linguistic usage). To pronounce this issue, the more general term of an 'entity' will be used to describe the source of the cue and the target (the 'patient' in Gärdenfors' terminology) of an affordance.

Planning in Affordance-based Robot Control. In [24], Lörken and Hertzberg introduce a way to combine automated planning and affordance-based robot control with the goal of avoiding symbol grounding. The affordance-based approach inherently uses no environment model in sense of a representation of recognized and classified objects, thus the first direction of grounding for building a model is avoided. When a solution to a planning instance is generated, they propose to ground the planned actions in an affordance triple. Therefore any suitable object is used that provides the affordance that is looked for.

In a first step, the affordance triples are grouped to so-called *abstract affordances*. When e. g. the robot can lift two kinds of objects, let's say red and green ones (maybe also using different behaviours), there is more than one triple, but only one abstract affordance *liftable*. Based on these abstract affordances, PDDL planning operators are modelled and used for planning with standard PDDL planning systems. Beside these affordance depended operators there might be others that have no affordance dependence. When an affordance is perceived, the position where it has been perceived is tagged in the robot's map. After generating a plan using a PDDL planning system, affordance depended operators are grounded in affordance-triples. Let e. g. be *lift(something)* in the plan, then all triples belonging to this abstract affordance are retrieved from the repository and the robot searches the environment for a cue that is contained in a triple. The first one that is found is exploited by applying the corresponding behaviour. Thereby it can also be triggered by an entity

that has not been perceived before and is therefore flexible in choice of the used entity as long as it has the desired properties.

The approach presented in the next sections builds up on [24] by using the idea of grouping triples to abstract affordances and to use these for (symbolic) planning. It extends the approach in three main points (2 and 3 will be further discussed in section 2.3).

1. Lörken and Hertzberg focus on being flexible in the choice of the *object* used for execution. In this work the focus is on being flexible in the choice of the used *behaviour*. It is tried to make a 'good choice' by using conceptual spaces for similarity estimation and to choose a behaviour that has been successful for analogue situations in the past.

2. The representation enables also the clustering of affordance triples and thus the grouping to abstract affordances by the robot without human help.

3. The conceptual spaces representation offers ways to translate subsymbolic to symbolic representations. Here it is discussed how these can be used to auto-generate planning operators. This would enable the agent to learn new operators autonomously.

In [39], Ugur *et. al.* introduce a system that uses the concept of affordances to learn the mapping between low level sensory input and the outcome of a set of behaviours. A trained *support vector machine* (SVM) is then used for planning. Training and evaluation is done in simulation, the learned actions are transferred to and tested on a real robot.

Therefore, feature vectors are extracted from sensory data and recorded during behaviour execution. A triple is stored containing the initial feature vector, the effect (i. e. the difference of initial and final vector) and the executed behaviour.

For each behaviour b, the effect vectors are clustered, the result is (presumably) more that one cluster. The clusters represent the different outcomes of the same behaviour. They can be interpreted as successes and failures and thus be used to detect affordances provided by an object.

Using the initial feature vector, a SVM is trained to predict in which cluster an unseen object will be after applying the behaviour. Planning is done using the SVM to predict the outcome cluster and applying by the mean effect of that cluster to the feature vector. It is planned using forward chaining. Therefore, there is no symbolic domain representation.

The system is tested on predicting affordances provided by an object (on simulation and real-world sensory input) and on finding plans with two steps as well on scenes with more than one object.

They give an interesting approach on learning affordances and outcome of actions. The leak of a symbolic representation of the planning operators prevents the use of domain independent planning system as proposed in this work.

An interesting extension to this work would be the clustering approach to distinguish success and failure of robot behaviour. However due to the parameterized

behaviours that will be used here, this is not straight forward, but has to be done based on the difference between intended and archived displacement vector.

2.3. Conceptual Spaces

In his book [12], Gärdenfors introduces a new level for cognitive modelling that is placed between *associationism*, like e. g. representations using neural networks, and *symbolic* representation. A main issue is to estimate similarity of entities using a simple distance measurement. The new layer can also be used to translate between the subsymbolic and symbolic representations.

> "Here, I advocate a third form of representing information that is based on using *geometrical* structures rather than symbols or connections among neurons. On the basis of these structures, similarity relations can be modelled in a natural way." [12, p. 2]

This section first introduces his basic framework called conceptual spaces as proposed in [12]. Afterwards a formal definition based on vector spaces is introduced. It concludes with ways to represent actions and events in conceptual spaces.

The basic element of the conceptual spaces framework is a *quality dimension*. A quality dimension gives a property that can be used to differentiate entities. It can have different structures, e. g. a linear structure starting at some fix point (when e. g. somebody's age is given in years), a number system with floating-point numbers like in $\mathbb{R}$ (here e. g. is also no starting point), or a simple categorization in classes. Even cyclic structures are possible when e. g. an angle is given in degree or radian measure.

There are dimensions that can be judged without considering any other, think about the weight of an entity standing on a table. If an agent wants to judge if it is possible to lift it, this dimension is meaningful by itself. Considering its relative position to the agent, the information of two dimensions is poor until the third dimension is added. The position dimensions are called to be *integral*, the weight dimension is *separable* [12, chap. 1.8].

The dimensions are partitioned into *domains*. A domain is defined as

> "*... a set of integral dimensions that are separable from all other dimensions.*" [12, p. 26]

A point in a domain is meaningful by without other information. One can define regions in a domain that describe a certain *property* of the described entity.

> "CRITERION P A *natural property* is a convex region of a domain in a conceptual space." [12, p. 71]

A property could e. g. define a range in relative distance from the agent that can be reached by its arm or leg. Thus properties can be used to assign symbols in subsymbolic space and translate a relative position of an object o into a $reachable(o) = \{true/false\}$ in the planning problem.

A *concept* consists of a set of domains that represent some entity. Gärdenfors introduces the following example [12, p. 102 f]: an apple can be described by its colour, its shape and its taste. All these domains can be examined separately form the others, but they are not independent from each other. A green apple will most probable be sourer than a red one. Such constructs can be represented as *concept* in a *conceptual space*.

> "CRITERION C A *natural concept* is represented as a set of regions in a number of domains together with an assignment of salience weights to the domains and information about how the regions in different domains are correlated." [12, p. 105]

A *(concept) instance* is a point in that space. As stated earlier, similarity is estimated in conceptual spaces via distance. Gärdenfors introduces weighted Euclidean or city-block metric for distance calculation [12, p. 20]:

$$d_E(x, y) = \sqrt{\sum_i w_i \times (x_i - y_i)^2} \tag{2.1}$$

$$d_C(x, y) = \sum_i w_i \times |x_i - y_i|$$

Both use dimension weights w_i, belonging to a specific context. This enables the system to be more flexible and to come up with situation-adequate similarity estimations. Gärdenfors also calls it "attention-weight" [12, p. 20] or "salience" [12, p. 103].

> "The main effect of applying a concept in a particular *context* is that certain domains ... are put into focus. ... the context determines the salience of the domains. This results in a *dynamic* conceptual space, which in turn makes concepts and similarity judgments dynamic ..." [12, p.132]

In this work, conceptual spaces are used to represent agent behaviour. Several benefits of the approach will be exploited. Similarity between entities (i. e. robot behaviours) has to be estimated in two ways: the *abstraction space* is used to cluster behaviours with similar effects on the environment and the *instantiation context* to choose the behaviour used to fulfill some intermediate goal.

A formal definition. Raubal introduces a way to formalize conceptual spaces as vector spaces [31]. This formalization is introduced in the following paragraph and will be used in this thesis.

A *conceptual vector space* is defined as a tuple, where each element is either a quality dimension or a (integral) domain, i. e. consists itself of a set of dimensions. To get comparable units on each quality dimension, Raubal proposes a standardization on quality dimension level using *z-transformation*. Similarity between two instances is calculated in a two-step procedure: after dimension standardization, the similarity is measured, as proposed by Gärdenfors, as weighted Euclidean distance.

A newly introduced instance with all mean values of the dimensions is used as concept prototype. Mappings between conceptual spaces are introduced, e. g. to map from a computer's conceptual space to the user's. This approach to transfer some space in another will be used in the abstraction step.

Raubal applies the approach in a case study that investigates how to determine which facade suits best for being a way point in human path finding. Here, e. g. with the two contexts *day* and *night* and a mapping between computer (RGB) and human (HSV) colour spaces.

Action and Event representation. In [13], Gärdenfors and Warglien discuss a way to represent dynamics in the conceptual spaces framework. They start with the representation of actions and extend it with the effects on the entity it is applied to. The resulting construct of actions and effects is called an *event*. For actions they suggest the following representation.

> "... we submit that *the fundamental cognitive representation of an action consists of the pattern of forces that generates it.*" [13, p. 12]

Thereby the *forces* are not intended to be physical measurements, but "psychological constructs" [13, p. 12] that are inferred by an agent e. g. based on perceived movement.

> "... we hypothesize that the brain extracts the forces that lie behind different kinds of movements and other actions." [13, p. 12]

They also explicitly include completely non-physical forces like emotional or social ones. The forces of an action are stored as quality dimensions in conceptual spaces. The resulting action space can be used to calculate similarity between different actions and to define action categories as convex region.

Events combine several other spaces with the goal to represent the effects of an action on a patient. The first element is a space describing the agent performing the action, including the force domains representing the action itself. The second space describes the parts of the patient that are relevant for the event and can also contain a domain of counter-forces. To represent the change of the system, a *state* is defined as a point in a conceptual space, the *change of state* is a vector that describes it and a *path* is a series of changes [13, p. 18f]. Now the third element of an event can be defined, the *result* (or *displacement*) *vector*, describing the change of the patients state. So an event is a mapping between the agent's action and the effects it causes on the patient.

Time is thereby represented as quality dimension. The authors restrict their work on events with clear start- and endpoints ('bounded events' [13, p. 23]) in contrast to ongoing 'processes'. Events can be reflexive when agent equals patient; and there must not be any agent at all (e. g. in case of a falling robot).

The framework of Gärdenfors and Warglien is used in this thesis to represent the robot's behaviour execution and the perceived effects on the entity it is applied to. This representation of past behaviour is the basis of the two main contributions of

this work, (1) the clustering of affordance triples to abstract affordances and (2) the action instantiation.

The idea is to do the abstraction via clustering of the displacement vectors, i. e. behaviours that cause similar change to the patient belong to the same abstract affordance. The instantiation is done by a clustering of the cues of affordances that belong to one abstract affordance. It is based on the patient's properties before behaviour execution, i. e. a behaviour is chosen that has been successful in the past and thus seems to be appropriate for the current situation.

There are other interesting questions in this context that are not tackled here. One interesting topic is the classification of cs actions (so here: robot behaviour), i. e. the force domain of the robot like tried in [3] (see discussion below). Another question is whether it is possible to 'plan' behaviour entirely as morph between actions performed in similar situations (this would enable learning of actions that are demonstrated by other robots or humans).

In [5], Cubek and Ertel try to "learn the goal behind a human-demonstrated task" [5, Sec. 1.3] using an event representation in conceptual spaces. Therefore they use an OpenRAVE simulation environment with several tables placed in a room. Each table holds a few blocks like in blocksworld. Vision is simulated by extracting the desired properties from simulation and putting a certain noise on it. Then actions are performed (using 'scripted' robot behaviour) and cs instances are extracted. By finding the "similarities among demonstrations" [5, Sec. 4.6], a second robot shall learn the goal behind the performed action.

A cs event is a connection between two blocks on a table (source and target). If there are more than two blocks, more than one event will result from one robot behaviour. The used conceptual space contains certain properties of the involved (two) objects as well as their spatial relation. By investigating the possible combinations of source and target dimensions, the systems tries to find clusters like *red boxes are always put on a yellow box*. These clusters are interpreted as goal behind the demonstration.

Beside for similarity estimation, Cubek and Ertel use conceptual spaces also to transfer the subsymbolic sensor data into a (symbolic) PDDL problem definition, using a hand coded domain. The found clusters are thereby used as goals; i. e. the robot that is planned for imitates the learned action.

Though Cubek and Ertel's work has a different goal, they try to learn relations between concept instances to re-achieve the goal in a different situation, it is interesting for this work due to the use of cs to translate between subsymbolic and symbolic representation for PDDL planning.

The work of Beyer *et. al.* presented in [3] also uses the approach of Gärdenfors and Warglien for their action representation. They use data extracted from a video data set that shows parents demonstrating actions to their children. The region of high motion is extracted and its position relative to two landmarks is stored. Landmarks are the initial and the final position of the object. The force pattern is a Boolean vector containing four values: *isAbove* and *isLeftOf* values for each landmark.

Their concept contains solely the force domain and no agent nor patient domains. As a consequence, action categories are not defined as concept but as property. Action categories are defined by the Voronoi cell belonging to a prototype that is the mean instance of a cluster.

The question they investigate is whether this representation is rich enough to discriminate action categories. Therefore two evaluations are done: one on cluster purity and one on the classification of new actions. The results show that their setup does not enable a satisfying discrimination of action categories. If the effects of behaviours are indistinguishable, this may provide an argument to use behaviour clusters (as abstract operators) instead.

2.4. Feature Selection

Concepts are defined on a domain subset of the conceptual space. In this work it is tried to automate the learning of behaviour concepts, so the subset should not be human-defined. A related task is the weighting of domains in the context of a certain task/situation. In fact the subset selection will be realized by a weight of 0, so the subset selection and the weighting will be the same task in the presented system.

In this work the weighting will be realized by using techniques from feature selection and implemented by using out-of-the-box feature selection packages in the environment for statistical computing *R* [30]. The weighting will be based on dimension-level and with the objective to weight the dimensions based on their influence on the behaviour outcome. This section gives a short introduction to the field of *feature selection*.

Feature or variable selection selects a subset of the original set of variables according to some criteria. The objective can be an improved resource usage or result quality of some regression or classification system (in context of this work: a classifier), or simply the better understanding of a system by focusing on relevant dimensions [17].

In [23], Liu and Yu develop a categorization system for feature selection techniques in the context of classification and clustering. They divide the main approaches into *filter* and *wrapper* techniques. Wrappers are designed to optimize the combination of a certain (preselected) predictor with the used data. The criteria used for filters work solely on the data. In the context of this work, especially filter methods that rank the input variables according to some score that reflect their relevance are interesting. They are not bound to a classifier and reflect relations in the data. Their scores will be used to weight the quality dimensions. Liu and Yu identified the following groups of evaluation criteria.

1. Distance measures work based on distances between clusters, i. e. they accent features that help separating the data.
2. Information measures use criteria from information theory like *information gain*.
3. Dependency measures are based on correlation between the features and the dependent variable.

4. Consistency measures try to "find a minimum number of features that separate classes as consistently as the full set of features can" [23, p. 7].

Surveys on subfields of feature selection relevant for this work can be found in [17], giving a general introduction, [23] especially with focus on classification and clustering and [37] with focus on filter methods.

In the evaluation several methods will be used. The goal is not an exhaustive evaluation and to find the method that fits best to be combined with conceptual spaces, but to investigate if there are techniques that are able to autonomously generate weights that improve the results of the introduced system.

3. Affordance-Based Action Abstraction

3.1. Of Affordances, Concepts and Planning Instances

This section discusses how the components introduced in the last section are combined.

In a first step a new affordance representation is given. It is based on the event representation is Gärdenfors' conceptual spaces framework. The design goal is to enable the agent to deliberate about the affordances offered and on how to exploit them to reach the agent's goals. Rome *et. al.* describe it as:

> "It [the agent] is not controlled by the environment. It can rather use the information provided by the affordances of a situation and reason about them in a goal-directed manner selecting those afforded actions that will lead to its goal." [35, p. 184]

The represented facts are very similar to the affordance representation in the MACS project. The environment offers some features that indicate a possibility for action (cue representation). The agent executes some behaviour (behaviour information) and observes its effect on the environment (effect). Here a concept space is defined based on the domains that the agent is able to perceive. This space is the same for all behaviours.

Regions in that space define the applicability of a behaviour. Beside this applicability that can be represented in any conceptual space, the agent has to memorize the effects of its doings. Thus the used conceptual space is an event space; the defined (cue-)concept gives information of the patient and the displacement vector the effects. An additional domain holds information about the applied behaviour. Here, this is the behaviour identifier as well as success-information.

So far all domains give information about the patient. Additional domains can be added to hold agent information. This can be static information, e. g. about the appearance of the agent or the force domains. Domains that describe the agent could be interesting if the learned behaviours shall be transferred to other agents. The restriction that affordances are agent specific could be relaxed in that way. Instead, the need of a certain similarity between two agents could replace it.

The represented behaviours are grouped to abstract affordances based on the function-centred view of affordance-based robot control. This can be done based on the displacement vectors of behaviours that the agent assumes to be 'successful'. A challenge when doing such abstraction is especially the use of parameterized behaviours because several instances of one behaviour may result in unsimilar displacement vectors. Gärdenfors described quality dimensions as "... ways stimuli are judged to be similar or different." [12, p. 6]. The term of effect similarity used in this work will be defined based on quality domains instead of quality dimensions. This decision is due to the fact that a dimension subset of a (integral) domain has no meaningful interpretation [12, ch. 1.8]. To overcome the problem of parameterized behaviours, the relative change to each domain is regarded. This means that the

total change of a behaviour execution will be normalized to some value (e. g. to be 1.0).

The following definition for similarity of behaviour effects will be used in this work.

Definition 3 *The effects of behaviours are similar if their relative change to the patient's quality domains is similar.*

This relative change will be called *force pattern*. It is also useful to define planning operators' effects, like 'change to the domain colour can be caused by the paint behaviour'. In this context the capabilities of cs to translate subsymbolic into symbolic representations have to be exploited, e. g. by introducing cs property definitions that are included into the planning process. Though the domain generation is not in the scope of this thesis it will be discussed in Sec. 3.3.2.

There are several concepts defined in the same conceptual space. Each of them represents one agent behaviour. The mapping of behaviours to concepts is in line with Gärdenfors framework.

> "The fundamental cognitive role of concepts is to serve as a bridge between perception and actions." [12, p. 122]

One has to consider that the regions of applicability of behaviours could overlap. If the agent shall benefit from the approach given here, they should overlap. If the regions of applicability of two behaviours overlap and their effects are similar, the agent has the chance to achieve a desired change of the environment in two different ways. If the first attempt of the agent fails, there might be a chance to reach the goal by trying the second one.

This leads to the second contribution of this work. If a planning system as generated a plan using affordance-based operators, the agent has to decide which behaviour to use when the plan is executed. This *action instantiation* has to choose the behaviour with the highest success probability. The hypothesis for this decision is that the execution of a behaviour in similar situations will have the same outcome. Section 5.3 shows that this holds then choosing proper context weights. This similarity is defined on the initial patient domains as well as the intended change. It is likely that the decision does not depend on all patient domains in the same way. When painting a patient, its initial colour may influence the result. When moving the patient it most probably will not. This is exactly the scenario Gärdenfors proposes to include the context of the decision into the similarity estimation. This is done by using dimension weighting. Thus for each generated abstract affordance, a context is defined that is used in the instantiation decision. To automate this step it is proposed to use techniques from feature selection to find dimensions that influence the behaviour outcome.

The dimension weighting is used for the following tasks:

1. As stated above, to increase instantiation quality.
2. It represents the dimensions that are important for a behaviour to succeed – so the weighting indicates preconditions to use in the planning domain.

3. Thinking of a robot: when searching an entity that offers some affordance, the extraction of unused dimensions could be stopped to reduce the systems load. So it can be used for attention control.

So far it has been discussed how the used concepts will be used to realize the proposed aims of this work. How there are two questions left: where do the concepts come from? And where do the underlying quality domains/dimensions come from. In this work it is proposed to learn the concepts based on a set of behaviour instances (this means by lazy learning). Similar behaviour instances form a new concept. This is in line with Gärdenfors' framework:

> "Because criterion C defines concepts as regions of conceptual spaces, similarity, in turn, can be used to define concepts." [12, p. 111]

As the introduced system learns the general concept from a sample-set, the system uses inductive inference. An interesting question in such learning systems is what the learning is based on (the *inductive bias*). Here, the requirement of region convexity can be seen as inductive bias of the approach: if two entities are found that fulfill some property (if e. g. knowing that both are *red*), every entity in-between is also regarded as *red*, otherwise it would be necessary to learn the colour of every single point of entity space by itself.

> "... a learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances." [26, p. 42]

However, it remains to be investigated how well the behaviours of a robot fulfils the convexity property.

The second question was where the quality domains come from. In this specific context, let's assume the agent to be some robot. In this case the quality dimensions will be based on some sensor system. To mesh with the affordance-based approach, the processing of the sensory data has to be lightweight. On top of that, they have to represent properties of the patient, not e. g. a category or class. In this work domains like e. g. the size, colour and a simple orientation are used. Extensions to that could e. g. be shape or silhouette information. Here, all quality dimensions/domains will be defined by the system designer. However, it is an interesting question if meaningful dimensions/domains can be extracted from sensory input by the robot itself.

A practical question is how biased the implemented dimensions are with respect to the intended robot use. This leads to the following questions: The first is if (all) introduced dimensions are useful for the system. This can be evaluated based on the dimension weights and the (as discussed before) the extraction of uninformative dimensions can be stopped. The second question regards the extensibility of the system. Will the representation be able to work on new agent behaviour? For this question, Gärdenfors gives the following answer:

> "The addition of new domains is often connected with new forms of *actions* that require attention to previously unnoticed aspects of concepts." [12, p. 103]

So, if new behaviours deal with yet unnoticed aspects of the environment new domains will be added to the system.

3.2. Representing Affordances in Conceptual Spaces

This section gives a formal definition of the proposed representation; it follows the conceptual spaces representation as vector space given in [31]. However, to improve legibility, dimension-nesting of (non-event) concepts will be restricted to two layers, as defined by Gärdenfors.

3.2.1. Conceptual Vector Space

Let D be the set of domains and CS the set of conceptual spaces. A conceptual space C is the set of vectors of the following form.

$$C = \{(d_1, d_2, ..., d_n) \mid d_i \in D\},\ C \in CS \tag{3.2}$$

A domain $d \in D$ is defined as a vector over an algebraic field $(F, \times_F, +_F)$.[3]

$$d = \{(f_1, f_2, ..., f_{|d|}) \mid f_k \in F\} \tag{3.3}$$

The number of dimensions of a domain is denoted with $|d|$. $c_{j,k}$ is the k-th dimension of the j-th domain of the concept instance c. Let $|c|$ be the number of domains of c and $|c_j|$ the number of dimensions of its j-th domain.

To define the needed concepts in a conceptual space C, a set of applicability regions has to be defined in a domain-subset of the vector space. As proposed by Raubal for the vector representation, a weight of 0 will be assigned to unused dimensions instead of giving a domain subset.[4]

The regions correspond to the applicability of a concept and can be used for cue detection. However, cue detection is not part of this work. Formally, it might be realized using a decision function that maps dimension input to $\mathbb{B}$ to decide which point is inside the convex region. Section 5.3 gives some further discussion on cue detection.

An interesting question to investigate is whether the region convexity could be used as criterion to decide which domains belong to the domain subset.

[3] In the realized system this will usually be the set of real numbers along with its common definitions of addition and multiplication $(\mathbb{R}, \times, +)$. If not stated differently, this structure will be assumed.

[4] Strictly speaking, it is the additive identity of the algebraic field. Let $\overline{0}$ be the additive identify and $+$ and $\times$ the addition and multiplication operator. By definition $x + \overline{0} = x$ for all x out of the used set. As $\overline{0}$ is in the set, this means also that $\overline{0} + \overline{0} = \overline{0}$. In an algebraic field, the set is a closure under the operators, distributive law holds and there are inverse elements (let x^{-1} be the additive inverse of x). With this properties, $x \times \overline{0} = x \times (\overline{0} + \overline{0}) = x \times \overline{0} + x \times \overline{0}$. Adding the inverse element, $x \times \overline{0} + (x \times \overline{0})^{-1} = x \times \overline{0} + x \times \overline{0} + (x \times \overline{0})^{-1} \Leftrightarrow \overline{0} = x \times \overline{0}$, for every x. This means that dimensions with $\overline{0}$ weight are not considered in distance calculation.

3.2.2. Behaviour and Event Space

Based on the definition of conceptual spaces, action and event space can be defined. The first difference is that these hold the values for each dimension for more than one point in time. This can be realized by introducing a new dimension in each domain representing time during action execution and storing a series of instances for each behaviour execution. However, to improve readability, it is defined as vector space C^{n+1} $n \in \mathbb{N}$ over some conceptual space $C \in CS$. n is the number of time slices and there is an additional $c \in C$ holding the displacement vector δ (its definition will be given below).

Beside the 'observations' the agent sensed while doing some behaviour, it is represented which behaviour was executed and if (the agent thinks that) the execution has been successful. Let $B = \{b_1, b_2, ..., b_p\}$ be a set of (agent) behaviour-identifiers and $s_i \in \mathbb{B}$ holding the success value. These dimensions are constant for one event, i. e. there is no need to store them for every point in time. They are added as new *behaviour*-concept to the event that only holds one domain that is composed of the behaviour identifier and the success value. An event over some conceptual space C is defined as:

$$E^C : (B \times \mathbb{B}) \times C^{n+1}$$

with $n \in \mathbb{N}$ and $C \in CS$. Due to readability one might skip parenthesis around the behaviour concept. An instance e_i of an event has the form

$$\begin{aligned} e_i = (((b^i, s^i)), c^{i,1}, c^{i,2}, ..., c^{i,n_i}, \delta^i) \text{ with } & b^i \in B, s^i \in \mathbb{B}, in \geq 2, \\ \text{and } & c^{i,j} \in C, \delta^i \in C, C \in CS \\ \text{as well as } & \delta^i = f_\delta(c^{i,1}, c^{i,2}, ..., c^{i,in}) \end{aligned} \tag{3.4}$$

The displacement-vector is calculated by a function

$$\begin{gathered} f_\delta : C^n \to C \\ \text{with } f_\delta(c^1, c^2, ..., c^n) = c^n - c^1 \end{gathered}$$

Subtraction is thereby defined dimension-wise; i. e. the result is a concept instance that has the same number of domains and dimensions and each dimension holds the difference of the corresponding dimensions of the subtracted concepts.

Let us summarize the event definition. An event is a vector of $n + 1$ concept instances of the same conceptual space. They hold the values for n time slices as well as the displacement-vector δ that is the difference of the initial and final concept vector. It is extended by an instance of the behaviour concept, holding the behaviour name and a Boolean success-value.

The definitions in the next sections will contain several instances of events with several time slices. The following indexing will be used. As defined above, dimension l of domain k of the concept c will be denoted as $c_{k,l}$. To denote time slice j of event i, it is written $c^{i,j}_{k,l}$.

3.2.3. Affordance Space

Affordance space can now be defined as special event space as follows (cf. Eq. 3.4 and Def. 2 on p. 10).

$$E^C : (B \times \mathbb{B}) \times C^3$$

An instance e_i of an affordance has the form

$$\begin{aligned} e_i = (((b^i, s^i)), c^{i,1}, c^{i,2}, \delta^i) \text{ with } & b^i \in B, s^i \in \mathbb{B}, \\ \text{and } & c^{i,1}, c^{i,2}, \delta^i \in C, C \in CS \\ \text{as well as } & \delta^i = f_\delta(c^{i,1}, c^{i,2}) \end{aligned} \tag{3.5}$$

In the original definition the cue and effect were represented by "a list of attribute value pairs" [35, p. 181], here they are points in the used vector space given by $c^{i,1}$ and $c^{i,2}$. Beside $c^{i,2}$ that gives absolute values, the effect is given by the (relative) displacement δ^i. The behaviour descriptor is replaced by the behaviour b^i and its success value s^i. As the new affordance representation is a special case of an event, definitions in the following sections may also use the (more general) event space.

3.3. Deliberation Methods

After event representation has been defined, this section describes the algorithmic part of the system. Algorithm 1 gives the top-level view on the approach described

Algorithm 1 Top-level algorithm

```
function PLAN(M, G)
    A ← genAbstractAffordances(E)          ▷ abstraction, Sec. 3.3.1
    (D, P) ← genPlanningInstance(G, A)     ▷ domain generation, Sec. 3.3.2
    (π, ≺) ← callPlanningSystem(D, P)
    return execute(E, π, ≺)                ▷ includes instantiation as
                                             main step, given in Sec. 3.3.3
end function
```

in this thesis. Based on a training set of events M (the agents Memory), abstract affordances are generated, using a mapping to the *abstraction* space that enables the comparison of different behaviours. The mapping and the algorithm is described in Section 3.3.1.

Based the abstract affordances, a planning instance (i. e. planning domain and problem) is encoded in some domain-independent language, e. g. PDDL or the Situation Calculus. This step has been explicitly excluded from this thesis due to time reasons, but is shortly discussed in Section 3.3.2. Using the planning instance, a solution is generated that is a (partial) ordered plan consisting of a tuple of actions π and a predecessor relation $\prec$ that defines the (partial) order on π.

Algorithm 2 Plan execution

```
1: function EXECUTE(E, π, ≺)
2:     F ← ∅
3:     while F ⊊ π do
4:         select a ∈ {c ∈ [π − F] | [∃ b : (b, c) ∈ ≺] ⇒ [b ∈ F]}
5:         (b_1, b_2, ..., b_n) ← instantiate(a)          ▷ instantiation, Sec. 3.3.3
6:         i ← 1
7:         while i ≤ n do
8:             (qdims, success) ← execute(b_i)
9:             E ← E ∪ {(b_i, success, qdims)}
10:            if success then
11:                F ← F ∪ {a}
12:                break
13:            else
14:                i ← i + 1
15:            end if
16:        end while
17:        if a ∉ F then
18:            return failure
19:        end if
20:    end while
21:    return success
22: end function
```

The solution is executed by the function given in Algorithm 2. First, applicable actions are identified and one of them is selected (line 4). Then it is instantiated by a robot behaviour using the *instantiation* context and the algorithm given in Section 3.3.3 (line 5). When an execution failure occurs, the next choice is tried. After behaviour execution, the set of events is updated. In the algorithm, this is done by just adding the new event. However, a more sophisticated method could be considered to (1) keep the number of examples low and (2) get good learning results.

A further description of the proposed execution system, also introducing how to execute solutions with parallel and temporal actions, can be found in [19, Sec. 2.3]. How to generate a partial ordered plan based on a PDDL definition to improve execution flexibility is described in [19, Sec. 2.2].

3.3.1. From Agent Behaviour to Abstract Affordances

In [24], the concept of abstract affordances has been introduced that enables an abstraction from concrete affordance triples during planning process. In the MACS project, the grouping was introduced by the (human) domain designer. This section introduces an approach to automate the abstraction step.

The introduced method clusters affordance triples that change the environment in a similar way, i. e. by the effect that they have on it. It thereby ignores the situations it is applicable to (so to say, the cues and precondition). Speaking in terms of conceptual spaces, clustering is done based on the displacement vector of the affordance instances.

It has to be considered that abstraction is done on instances of different behaviour types: *is* b_{pull} *more equal to* b_{turn} *than to* b_{push}*?* – this is done using a mapping to a special space that is introduced now, before coming to algorithmic aspects of the abstraction afterwards.

Mapping to the abstraction space. Different behaviour types are compared by their change on the patient. The *force pattern* of a behaviour is therefore defined based on the similarity definition given in Def. 3. It is based on the displacement vector of a behaviour and gives for each domain the portion of the entire change caused to the patient. If a behaviour changes e. g. the colour and the weight of a patient, it has to be considered how much these domains are changed and an overall change of 1.0 is divided according to the amount of change to the domains, e. g. *colour-change* = 0.9 and *weight-change* = 0.1 for painting the patient (using a quite heavy paint). So the force pattern would be (0.9, 0.1) for this behaviour.

Force patterns allow to make general statements about a certain behaviour type: there are infinite ways to $b_{oil\text{-}paint}$ some *oil-paintable* object that may result in very small and very large displacement vectors (i. e. different colours), but it is assumed that they share a large relative change to the colour domain (together with some minor other changes).

To generate the force pattern two steps of normalization are done. When comparing change over different domains, different ranges of values have to be considered. In

a first step these are normalized via z-transformation as proposed by [31] for the vector space representation. The z-transformation z for a value x is defined as:

$$z = \frac{x - \overline{x}}{s_x} \tag{3.6}$$

where $\overline{x}$ is the arithmetic mean value of the dimension and s_x the experimental standard deviation.

After z-transformation, the new 'unit' of a dimension is its standard derivation. This enables the comparison of change in different domains. In event space, z-transformation is done for each dimension over all time slices of all events.

Now different dimensions of a vector can be compared. In a second step the relative part of change for every domain is calculated. This is no normalization over one dimension of all instances like before, but over the different domains of one instance. This can not be realized using a context, because every instance would need different context weights. Instead, a mapping f_{abs} to another conceptual space is defined. It will be called abstraction space, because it will be used primarily for the abstraction step. However, as states above, it also provides general information about the change of a behaviour and thus can help in planning domain generation.

The new space has two domains: the behaviour identifier B in its own domain and a force domain that has $|C|$ dimensions with values between 0 and 1, each holding the part of change of one domain of the original space.

$$\begin{aligned} & f_{abs} : E^C \to B \times \left([0..1]^{|C|}\right) \\ & f_{abs}(e_i) = \left(b^i, \left(d_1, d_2, ..., d_{|C|} \mid d_j = \frac{\sqrt{\sum_{m=1}^{|\delta_j^i|} \left(\delta_{j,m}^i\right)^2}}{\sum_{k=1}^{|\delta^i|} \sqrt{\sum_{l=1}^{|\delta_k^i|} \left(\delta_{k,l}^i\right)^2}} \right) \right) \end{aligned} \tag{3.7}$$

every d_j gives the (unweighted) Euclidean distance in one domain, divided by the sum of the Euclidean distance of all domains.

In the resulting space, a prototype for each behaviour is generated as an instance holding the average value over all successful instances in every dimension. These prototypes enable statements like *on average, a turn behaviour results in a large change of entity orientation as well as a small change in entity position.* Based on this function-centred view on the behaviours, similar behaviours are clustered to abstract affordance operators. It is also the basis for the effect statement of the operator in the planning domain. Section 5.1 gives results of a clustering based on displacement patterns.

3.3.2. Generate PDDL Domains

Due to time reasons, the task of generating a working planning instance is not part of this thesis. However, this section gives some discussions on how this could be

done.

The main problem when combining affordance-based robot control with symbolic planning is to find a proper degree of modelling. To exploit the capabilities of state-of-the-art planning systems, domain has to provide enough information to reason about. Models in affordance-based robot control are inherently sparse. In the MACS approach, sensed affordances have been assigned to regions of an environment map [24] and entities or objects are not modelled at all. However, to realize a (robot) agent that shall e. g. be used as domestic service robot, it is necessary deal with object identities, but increase or decrease constraints based on application context (cf. the work of Iman Awaad, given in [2, p. 5f]). Instead of giving 'the solution' to this dilemma, this section discusses a range of possible alternatives for automated generation of planning operators based on the abstract affordances as realized in Sec. 3.3.1.

Gärdenfors' quality dimensions describe properties based on that two entities can be distinguished. Therefore quality dimensions and domains seem suitable to describe a state of the world. In PDDL this can be realized by using them either as *property* or *function*. The use of PDDL properties may imply the definition of cs properties that are mapped to each other.

The operator's effects on the patient have already been used to cluster the affordances. This information can now be reused in the operator definition to find the operators name as well as effects. For each abstract affordance, an operator `change<Doms>` is created, where `Doms` is replaced by the domains with the main effect. As parameters, one could use the entity that is the patient. When using properties and not functions to describe the domain, the property before and after the execution are also parameters. E. g.

```
(:action changePosition
  :parameters (?e - entity ?init ?final - aPosition)
```

for property-based encoding or

```
(:action changePosition
  :parameters (?e - entity)
```

for function-based encoding. A function-based encoding might provide more information to the planning system. But it has also to be considered that domains (normally) consist of more than one dimension and thus the encoding effort might be increased rapidly. Which dimensions are important can be decided based on the context weights, i. e. only dimensions with a certain weight are considered for planning.

When realizing the encoding with properties, a map-based encoding inspired by the MACS encoding could be realized. I. e. sensed affordances are written into an environment map. So, a precondition could encode that an affordance `positionChangeable` (in this case, a PDDL property) has been sensed at some agent position for some entity `?e`.

```
                    ; encoding affordance-perception
  :precondition (and (positionChangeable ?e ?agent-pos)
                     (position ?init ?e)
```

By encoding preconditions by affordance perception (in some map region), the planning domain becomes sparse and the actual decision on whether the operator is applicable or not is done in the conceptual layer. However, this prevents the planning system from realizing a precondition whose affordance has not been sensed before. Here, some 'finding' operator (e. g. `find-liftable`) could be implemented that seeks to find the needed affordance.

Based on the main effect of an operator, the effects can be encoded e. g. as

```
  : effect      (and (not (position ?init ?e))
                     (position ?final ?e)
                )
```

A full property-based encoding could look like

```
(define (domain rob)
  (:requires :typing)
  (:types entity property ; basic types
      agent-pos           ; regions in map
      anOrientation aSize aPosition aColour... - property values
  )

  (:predicates
    ; map of sensed affordances
    (positionChangeable ?e - entity ?p agent-pos)

    ; entity properties
    (orientation ?o anOrientation ?e - entity)
    (size ?s - aSize ?e - entity)
    (position ?p - aPosition ?e - entity)
    (colour ?c - aColour ?e - entity)
    ...
  )

  (:action changePosition
    :parameters (?e - entity               ; patient of the event
                ?init ?final - aPosition  ; old and new position
                ?p - agent-pos)
    :precondition (and (rotpos ?p)
                       (positionChangeable ?e ?p)
                       (position ?init ?e)
                  )
    : effect      (and (not (position ?init ?e))
```

```
                    (position ?final ?e)
                )
)
```

The dominant change of an abstract affordance is not limited to one cs domain. There might also be operators changing two or more.

```
(:action changeOrientationAndPosition
  :parameters (?e - entity                  ; patient of the event
              ?iOri ?fOri - anOrientation ; old and new orien-
              ?iPos ?fPos - aPosition     ; tation and position
              ?p - agent-pos)
  :precondition (and (rotpos ?p)
                    (oriAndPosChangeable ?e ?p)
                    (position ?init ?e)
                )
  : effect      (and (not (position ?iPos ?e))
                    (position ?fPos ?e)
                    (not (orientation ?iOri ?e))
                    (orientation ?fOri ?e)
                )
)
```

To sum up, the information gained for abstraction and instantiation already provide many of the facts that are needed to encode a planning domain:

- The dimension weights that will be introduced in the next section are used to decide which dimensions are relevant for precondition encoding.
- The force patterns defined for abstraction step give information on the effects of affordance-based operators.
- The quality dimensions and domains that are either in an effect of an operators or have some weight in the instantiation provide information about which of the (human introduced) features of the environment are relevant for the planning problem. Thus there are two uses for them:
  1. They describe a state of the planning system, so they should be used as (1) property or (2) function.
  2. As mentioned in Sec. 3.1, it can be used to control robot attention when searching for a certain affordance. The controller can stop the extraction of unused dimensions and thus speed-up agent vision.

However, it has to be further investigated how the capabilities of Gärdenfors' conceptual spaces to translate subsymbolic to symbolic representations are used most gainful.

3.3.3. Instantiate Actions by Agent Behaviour

When the agent has planned using affordance-based planning operators, it has to decide which behaviour to use to instantiate the action. This is done by choosing a behaviour that has been successful for 'similar situations' in the past. Here again, similarity has to be measured. As in the abstraction step, it is estimated using distance in conceptual space.

Different dimension ranges are normalized as before using z-transformation. For each abstract affordance, a context is introduced that sets dimension weights according to their relevance on the results. Weight calculation is done using methods from the field of feature selection. Based on normalized instances, instantiation is done via k-Nearest Neighbours (k-NN) classification and using weighted Euclidean distance. The instantiation results not only in one behaviour that should be applied but in an ordered tuple containing the behaviours in the order they should be executed, i. e. first choice, second choice and so on.

Let an event instance e be *relevant* for the instantiation of a if it

- describes the execution of a behaviour type that has been grouped to the abstract affordance type of a and
- has been finished successfully.

Let B_a be the set of behaviour types that have been grouped to the abstract affordance type of a and $|B_a|$ the number of behaviours in it. Alg. 3 defines the

Algorithm 3 k-NN instantiation algorithm

Require: $k \in \mathbb{N}$ arbitrary constant number of neighbours

1: **function** INSTANTIATE(M, a)
2: $S \leftarrow \{e \in M \mid relevant(e, a)\}$
3: $w \leftarrow weightDim(S)$
4: **for** $j = 1$ to $|B_a|$ **do**
5: $F \subseteq S$ with $|F| = k$;
6: $\forall f \in F(\forall s \in S((d_E(s, c, w) < d_E(f, c, w)) \Rightarrow s \in F))$
7: $i_j \leftarrow arg\,max_{b \in B_a}(|F_b|)$ ▷ random tie-break
8: $S \leftarrow S - S_b$
9: **end for**
10: **return** $(i_1, i_2, ..., i_{|B_a|})$
11: **end function**

action instantiation. $d_E(e_i, e_j, w)$ is assumed to be a distance-function as introduced in Eq. 2.1 that uses the dimension-weight-vector w.

The function gets a set of training examples M (the agent's memory) and the action a that has to be instantiated as input. It requires the definition on how many neighbours to consider, k. In a first step the set of relevant instances is determined (line 2). S contains the set of positive training examples whose behaviours belong to the abstract affordance type of a. In line 3 the dimension weights are calculated. In

line 4-9 the choices are determined one after the other. The k nearest neighbours are extracted from S (line 5 and 6). The behaviour that appears most often in the set is chosen using random tie-break (line 7). For the next choice, instances of behaviours that have already been chosen are not considered anymore.

When using k-NN algorithm, it might be useful to select M as a subset of the training set that contains equal numbers of positive and negative examples of every behaviour. This is also done in the evaluation in Sec. 5. In this case, it has to be evaluated if the used feature selection method benefits from the reduced or from the full training set. A short evaluation of this issue is given in Appendix D. It shows that there are methods that seem to benefit from the full training set as well as methods that benefit from the reduced one. So this has to be considered after choosing some method.

4. An OpenRAVE Prototype

This section introduces the system used to evaluate new approach given in the last section. Therefore it sketches the used robot and simulation frameworks and the used robot before introducing the quality dimensions, domains and the implemented robot behaviour.

4.1. System Overview

The system is based on the *Robot Operating System* (ROS) [29, 36] and the *OpenRAVE* simulation environment [7, 6].

ROS enables a simple communication between different parts of a software system that are called *nodes*. It has one central component, the *roscore* that provides e. g. a name service. The nodes register at the roscore and can publish and subscribe *messages* to asynchronous communication channels called *topics*. Synchronized communication is enabled by using *services*. Interoperable nodes can be implemented in several programming languages: C++, Python, Octave and Lisp. There is also some limited Java support. A node is supposed to implement only a narrow bunch of functionality to increase reusability. From this point of view the ros message definitions provide the interface between nodes. Nodes that belong together are combined in *packages*. ROS provides a large repertoire of tools to support developers, ranging from software building, deployment, logging of ROS messages, program structure- and communication-analysis, launching and mechanisms to visualize sensory data.

Although ROS also provides a 3D simulation environment named *Gazebo*, in this project it is combined with another simulation environment called OpenRAVE. OpenRAVE provides a wide range of predefined algorithms for robot motion planning. Thus it enables a simple implementation of a large number of different behaviours as necessary for the evaluation. It can be extended by so called plugins and provides a Python as well as a C++ interface for programming. A ROS plugin enables it to publish sensor data into a ROS network.

The prototype is implemented in ROS with one simulation node that is connected to ROS and also starts an instance of the OpenRAVE simulation environment. It implements several robot behaviours and executes them whenever it receives a command via a ROS message. It extracts and publishes current quality dimensions into the system. This architecture makes it possible to combine the advantages of the well-known ROS system and the algorithmic power of OpenRAVE.

The implementation in ROS enables a simple transfer to a real robot by exchanging the simulation node by real-world components and thus executing the commands by a real robot and publishing dimensions extracted from real sensors.

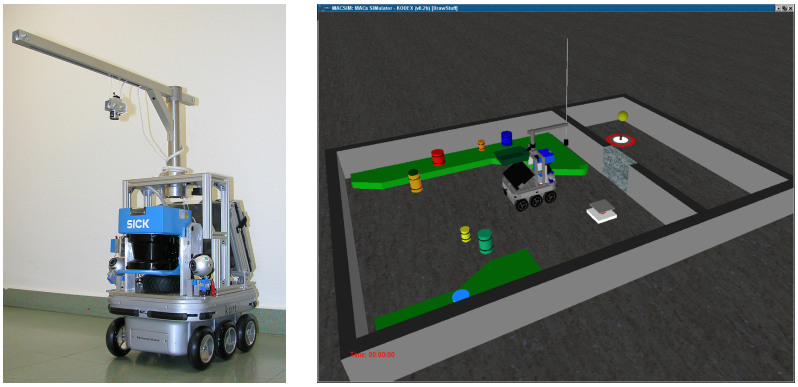


Figure 3: Robot and environment used in the MACS project (source: [34, Fig. 6, 9])

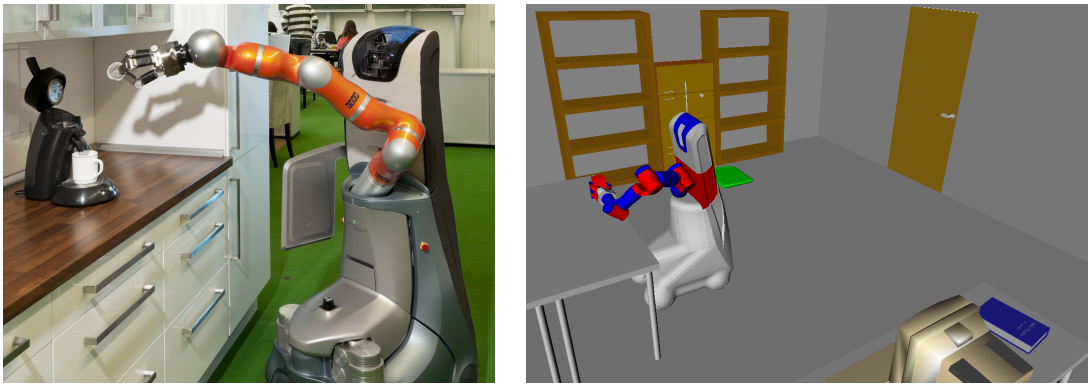


Figure 4: The Care-O-bot *Jenny* at Bonn-Rhein-Sieg University and in simulation

4.2. Robot and Environment

The MACS project used a robot with quite limited manipulation capabilities. It is based on the KURT3D [41] that has been extended with a crane manipulator with a magnetic gripper [35, p. 196 f]. Figure 3 shows the MACS robot and the simulation environment.

This robot has been sufficient for MACS approach. However, due to the focus on robot behaviour, it is not for this work. To group behaviour causing similar change, a number of different robot behaviours have to be implemented. Therefore another robot is used for this prototype: the Care-O-bot 3 [10] from Fraunhofer IPA. It provides the necessary manipulation capabilities and is also (physically) present at the robot team *b-it-bots* [4] at Bonn-Rhein-Sieg University. So the results gained in this project can be transferred from OpenRAVE simulation to the real world. Figure 4 shows the real robot called *Jenny* and its counterpart in OpenRAVE.

4.3. Quality Dimensions and Domains

This section introduces the used quality dimensions. It is divided into two parts: the agent dimensions, describing properties of the acting robot, the patient dimensions

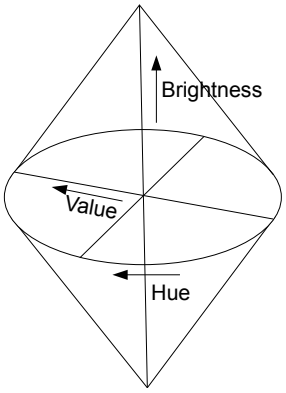


Figure 5: Colour space as described by Gärdenfors

that are most relevant for the introduced approach.

4.3.1. Agent Domains

As described in Section 2.3, affordances are agent-specific. This can be illustrated e. g. by considering any manipulation behaviour. It is clear that the individual properties of an agent like the existence of a manipulator, its type, range of reachability and strength affect the ability to transfer learned behaviour to another agent. The agent domains characterize the acting agent and enable a comparison to other agents. In this way, the estimation of transferability is one possible application of the given domains.

4.3.2. Patient Domains

These dimensions are most relevant for the approach introduced so far. They describe properties of the entity that is manipulated. Due to the affordance-base approach, all dimensions are in principle directly perceivable and not based on classification.

Colour. Gives the colour of the patient entity in *hue-saturation-value* (HSV) model. The domain consists of the three quality dimensions hue, saturation and value that are summarized in Table 1. Its shape is described in [13, p. 3] and given in Figure 5.

name	value range
hue	[0 .. 360)
saturation	[0 .. 1]
value	[0 .. 1]

Table 1: Colour domain

Size. The *size domain* gives the extents of the *axis-aligned bounding box* of the manipulated entity from the agents point of view. To have it independent from the

orientation of the entity it is not defined based on axis orientation but ordered by extents (a longer sider of the object and a smaller side if the object). Instances of the two dimensions can have every positive value, both dimensions are a linear shape. The domain is summarized in Table 2.

name	value range
small side	$(0 \,..\, \infty]$
large side	$(0 \,..\, \infty]$

Table 2: Size domain

Orientation. This domain gives a very simple and thus directly perceivable indicator for the orientation of an entity described by the relation of the vertical and horizontal extents of the *axis-aligned bounding box*. This is the only dimension of the domain. It has a linear shape and instances can have every positive value. The domain is given in Table 3.

name	value range	description
orientation	$(0 \,..\, \infty]$	$\frac{\text{horizontal extent}}{\text{vertical extent}}$

Table 3: Orientation domain

Relative position. Gives the relative position of the patient to the agent. The domain is composed of three linear dimensions. The dimensions are summarized in Table 4.

name	value range
x	$[-\infty \,..\, \infty]$
y	$[-\infty \,..\, \infty]$
z	$[-\infty \,..\, \infty]$

Table 4: Position domain

4.4. Robot Behaviour

OpenRAVE provides a wide range of predefined algorithms that can be used to implement custom behaviour. Therefore the following paragraphs give a short introduction of the used predefined behaviour; these will be used as primitive commands in the custom algorithms presented afterwards. Some of them use randomized algorithms and can therefore have a nondeterministic outcome.

4.4.1. OpenRAVE Behaviour

GraspPlanning. There are several utilities related to grasp planning, from simple commands to open and close the manipulator to algorithms to calculation grasps

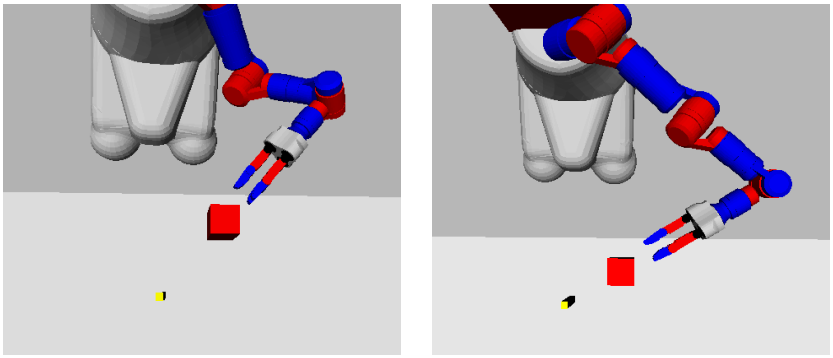


Figure 6: Push object with fingertips. The yellow object marks target position.

given a special object and robot hand.

MoveHandToPosition. Moves the robot's hand collision-free to the given pose.

MoveHandStraight. Moves the robot's hand in a straight line into a given direction. A stepsize and a range of steps are given to define how long the movement is done.

4.4.2. Custom Behaviour

This section introduces the behaviours used for evaluation.

Pick & Place. This behaviour is entirely based on a predefined OpenRAVE behaviour. It uses grasp planning to grasp the object, *MoveHandStraight* to lift it a bit, *MoveHandToPosition* to change its position and *MoveHandStraight* to put it down.

Pull and Pull. Is a behaviour that uses grasp planning to grasp the object and *MoveHandStraight* to change its position, i. e. it is left on the table surface and pulled (pushed) into its final position. However, due to technical problems with 'collisions' between object and table that stopped the *MoveHandStraight*, the behaviour used in the evaluation has to lift the object slightly be fore doing the straight movement, and put it down afterwards.

Fingertip-Push. Is another push behaviour that enables the robot to push the object into a position that can not be reached using a grasped push or a pick & place behaviour.

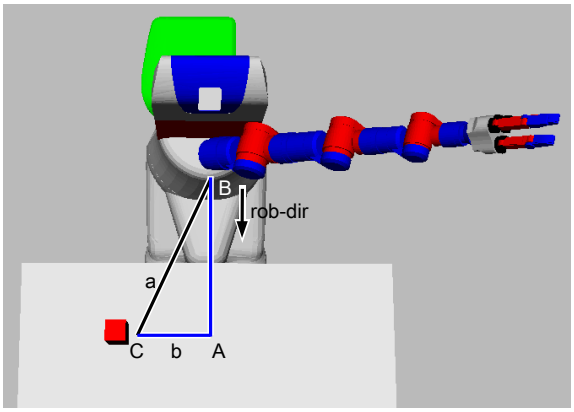


Figure 7: Sketch of the *Turn* behaviour

Algorithm 4 Fingertip-Push behaviour

1: **procedure** TFPUSH($robot, target, initDist$)
2: $s \leftarrow target.position$
3: $d \leftarrow destination$
4: $dir \leftarrow s - d$
5: $\alpha \leftarrow atan\left(\frac{dir_x}{dir_y}\right)$
6: $handRot \leftarrow getXRotMatrix(\frac{\pi}{2}) \times getZRotMatrix(-\alpha)$
7: $dir \leftarrow normalize(dir)$
8: $handPos \leftarrow scale(dir, initDist) + s$
9: $handTransform \leftarrow handRot \times getTransMatrix(handPos)$
10: $MoveHandToPosition(handTransform)$ ▷ Start move
11: $MoveHandStraight(direction = -dir, steps = [1..\infty])$
12: $grabbed(target)$ ▷ Prevent collision with target to stop movement
13: $MoveHandStraight(direction = -dir, steps = [1..\frac{dist(s,d)}{stepsize}])$
14: $\neg grabbed(target)$
15: **end procedure**

Algorithm 4 gives the fingertip–pushing behaviour. In a first step the direction and rotation of the hand are calculated (line 2-5). The hand-rotation has to be transformed because fingertips point in position z-axis direction (line 6). The pose of the hand is calculated in line 7. Then the movement is done. The hand is placed in line with the object's initial and target position. This is illustrated in Figure 6. Then the gripper is moved in a straight line towards target until it collides with the object. The object is pushed to the target position.

Turn. The *turn* behaviour is a special pick and place operation that is used to turn an object. A sketch visualizing the used parameters is given in Fig. 7. The names of lines and points given here and in the Alg. 5 can be found in there. The points B and C as well as the direction the robot is facing are given. Point A can be calculated

as the nearest point between the line defined by B and the direction vector *rob-dir* and the target-point C. The robot hand is opened and moved to a point on line b, with a hand and object depended distance. The fingers are placed the in front and behind the object. Afterwards it is grasped, lifted, turned and put down.

Algorithm 5 Turning behaviour

procedure TURN($robot, target, gripperLength, liftheight$)
  $C \leftarrow target.position$ ▷ Given things
  $B \leftarrow robot.position$
  $dir \leftarrow robot.rearDirection$
  $A \leftarrow -\left(\frac{dir \times (B-C)}{dir \times dir}\right)$ ▷ Calculate other elements
  $handDir \leftarrow normalize(C - A)$
  $\beta \leftarrow atan\left(\frac{handDir_x}{handDir_y}\right)$
  $handRot \leftarrow getXRotMatrix(\frac{\pi}{2}) \times getZRotMatrix(\beta)$
  $handPos \leftarrow C - scale(handDir, gripperLength)$
  $handTrans \leftarrow getTransMatrix(handPos)$
  $handTransform \leftarrow handRot \times handTrans$
  $ReleaseFingers()$ ▷ Start moving
  try
    $MoveHandToPosition(handTransform)$

  catch (planning failed)
    on error: grasp from other side ▷ skipped here

  $CloseFingers()$
  try
    $MoveHandStraight(direction = (0,0,1)^T,$
      $steps = [\frac{liftheight}{stepsize}]..\frac{liftheight}{stepsize}])$
    $turned \leftarrow getXRotMatrix(\frac{\pi}{2}) \times getHandTransform$
    $MoveHandToPosition(turned)$
    $MoveHandStraight(direction = (0,0,-1)^T,$
      $steps = [1..\frac{liftheight}{stepsize}])$

  catch (planning failed)
    $MoveHandToPosition(endPosition)$

  $ReleaseFingers()$
end procedure

The algorithm is given in 5, Figure 8 illustrates the steps. The robot places the gripper beside the object (a), grasps and lifts it (b and c). Then it is turned (d) and lowered until it collides with the ground (e). As last step the gripper is opened.

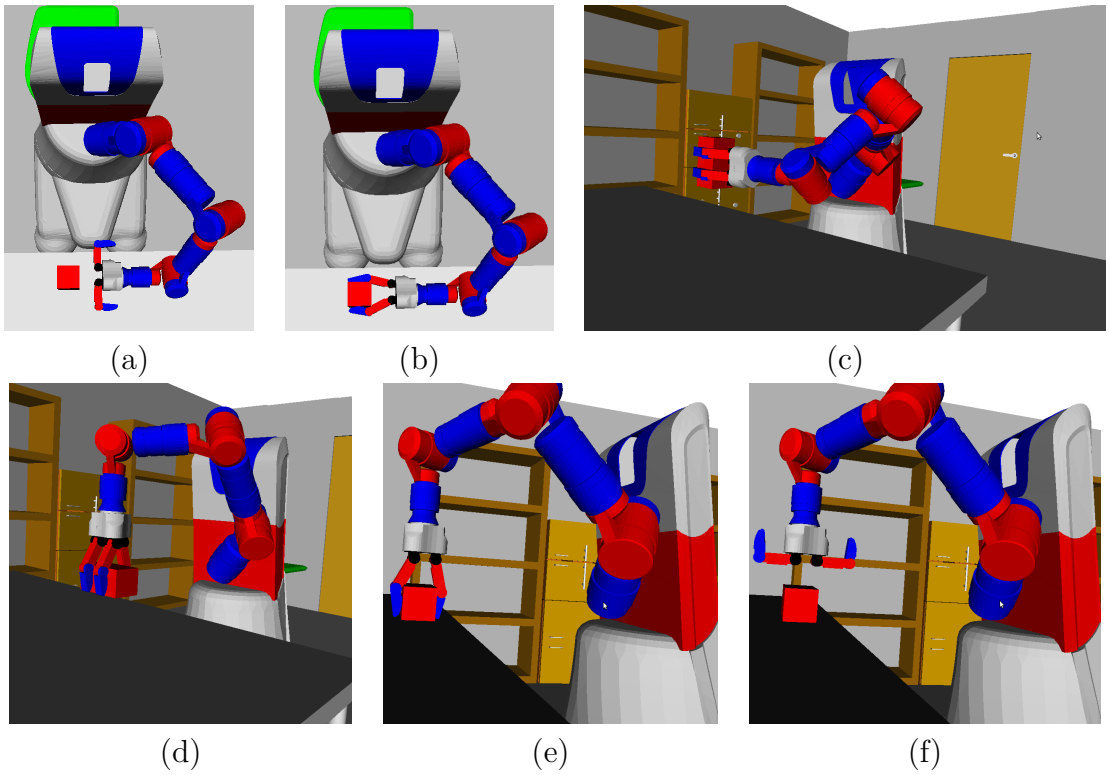


Figure 8: Care-O-bot 3 turning an object

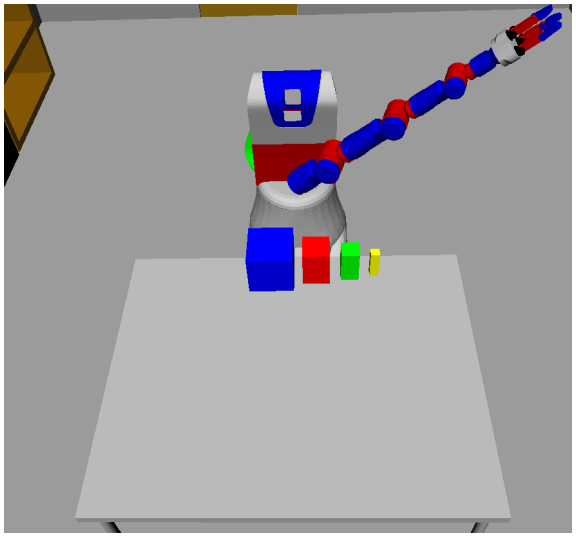


Figure 9: Experiment setup

5. Evaluation

In this section the following three points are evaluated in the OpenRAVE system.

- Can the abstraction space be used to replace the human abstraction? How do the resulting behaviour clusters look like?
- Does the proposed instantiation method increase the success-rate of chosen behaviours, using a 'guessing' instantiation as base-line?
- Are the dimension-weights generated with methods of feature selection able to improve the instantiation results?

Beside the evaluation given above, a short digression on predicting the success of a behaviour execution by k-NN classification is given in Sec. 5.3. Success prediction can be seen as cue detection. However, there are some conceptual problems that will be discussed. Though cue detection is not in the focus of this thesis, statements on how reliable the outcome of a behaviour can be predicted by comparison to similar situations in the past is an interesting question to get a feeling on whether the used similarity measure is plausible or not.

Experiment setup. The evaluation is based on randomly generated experiments. The robot is standing in front of a table with its arm directed towards it. The setup is illustrated in Figure 9. The depth of the table is larger than the robot's arm length, i. e. the robot is not able to reach objects standing to far from it.

For each experiment, an object is generated on the table that the robot has somehow to manipulate. All experiment parameters are generated uniformly distributed and statistically independent from each other. The behaviour is chosen from the set

width	depth	height
0.02	0.02	0.07
0.04	0.04	0.08
0.06	0.06	0.09
0.10	0.10	0.10

Table 5: Object sizes used in experiments

behaviour	experiments	successful	unsuccessful	success rate
b_{ftpush}	3850	102	3748	0.026
b_{pnp}	750	165	585	0.22
b_{pull}	1150	113	1037	0.098
b_{push}	1100	110	990	0.100
b_{turn}	700	87	613	0.124

Table 6: Overview experiments for training set

$\{b_{pnp}, b_{push}, b_{pull}, b_{ftpush}$ and $b_{turn}\}$ that have been described Sec. 4.4. Object x and y-position is generated, i. e. the position on horizontal plane on the table. All behaviours except b_{turn} need not only an initial position, but also a final position, i. e. a position the object is intended to be after the behaviour execution. The object's size and colour are also generated randomly. The used object sizes are given in Tab. 5.

Behaviour success is measured entirely based on the result on the object, (i. e. its position or orientation, respectively) using a delta of 0.05 meters for distance and 0.01 rad for orientation.

The experiment sets. Using this setup, two sets of experiments are used: a training set and an evaluation set. Different success rates of the behaviours result in different numbers of experiments to enable a training set that includes the same number of succeeded examples for k-NN classification. Table 6 summarizes the experiment runs of the training set.

The evaluation set is used to evaluate the instantiation method. It includes 1000 distinct problems. Each is tried to be solved by each behaviour of the operator `changePosition`. Table 7 shows the results of the evaluation set.

behaviour	experiments	successful	unsuccessful	success rate
b_{ftpush}	1000	28	972	0.028
b_{pnp}	1000	236	764	0.236
b_{pull}	1000	108	892	0.108
b_{push}	1000	97	903	0.097

Table 7: Overview experiment runs for evaluation set

Solutions	Experiments
0	738
1	64
2	189
3	9
4	0

Table 8: Number of experiments solved by 0-4 behaviours

Behaviour	Experiments
b_{ftpush}	19
b_{pnp}	44
b_{pull}	1
b_{push}	0

Table 9: Number of experiments solved only by one behaviour

The set of solved experiments in the evaluation set includes 262 problem instances. 469 solutions have been found. The success-rates of the two sets are comparable. Table 8 gives the number of experiments with $0-4$ solutions.

Table 9 gives the number of experiments that have been solved only by one behaviour. The number of experiments solved only by b_{ftpush} might be due to a wider range of reachability. It is the only behaviour that does not grasp the object. The number just solved by b_{pnp} can also be explained easily, because b_{push} and b_{pull} have to find a solution in the arm's joint space that results in a linear gripper movement. This seems to be more likely to fail. The large number of b_{pnp} in this set should be considered when interpreting the instantiation results.

Table 10 shows the experiments with two solutions. It gives the combinations of successful behaviours that appeared. The results show that most combinations are b_{pnp} and one of the b_{pull}/b_{push} behaviours, what seems to be plausible. These two experiment subsets, solved by one or two behaviours, will be used to evaluate the action instantiation.

As defined in Sec. 3.2.3, the representation includes for every dimension (except behaviour information that is represented only once)

1. the initial value

	b_{ftpush}	b_{pnp}	b_{pull}	b_{push}
b_{ftpush}	-	1	0	0
b_{pnp}	1	-	100	83
b_{pull}	0	100	-	5
b_{push}	0	83	5	-

Table 10: Behaviour combinations in the experiments with two solutions

quality domain	quality dimensions
info	behaviour
	experiment name
	success
colour	hue
	saturation
	value
position	x
	y
	z
size	short side
	long side
orientation	orientation

Table 11: Dimension overview

2. an absolute final value and

3. a relative final value (displacement value) denoted as δ.

Table 11 summarizes the quality domains with their dimensions. Some characteristic numbers for every dimension are given in Appendix A.

For a better understanding of the dimension weighting used in cue detection and instantiation, the evaluation of the abstraction step is given first. The dimensions used for weighting depend on that step.

5.1. Action Abstraction

All successful instances of the behaviours are transferred to the abstraction space as described in Sec. 3.3.1. For each behaviour a prototype is generated that holds the average values in every domain. These are clustered to generate abstract affordances. The prototypes are visualized in Fig. 10. It shows the supposed similarity between b_{ftpush}, b_{pnp}, b_{pull} and b_{push} that cause a dominant change to the position domain. b_{turn}, on the other hand, cause two-thirds of its change to the orientation domain and the rest to the position domain. This is due to the change in height of the boxes centre position.

Table 12 gives the exact values of the prototypes. The change in size domain is caused by the *axis-aligned bounding-box* measurement in combination with an (unintended) turning of the boxes.

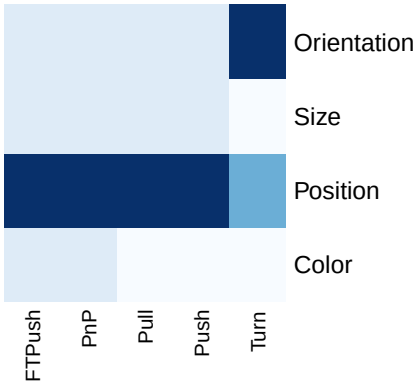


Figure 10: Force pattern of behaviours. Gives the average value of relative change in the domains for each behaviour type. The darker the blue, the more change. The values are scaling column-wise. This is the visualization of Tab. 12.

	b_{ftpush}	b_{pnp}	b_{pull}	b_{push}
b_{pnp}	0.007295633			
b_{pull}	0.109537785	0.102583253		
b_{push}	0.131000853	0.124395188	0.027697612	
b_{turn}	0.930382691	0.926318336	0.854673490	0.840075933

Table 13: Euclidean distances between behaviour prototypes in abstraction space.

	b_{ftpush}	b_{pnp}	b_{pull}	b_{push}	b_{turn}
Colour	0.000000	0.000000	0.000000	0.000000	0.000000
Position	0.994709	0.989341	0.897103	0.870585	0.338677
Size	0.003688	0.008607	0.051172	0.044108	0.000008
Orientation	0.001603	0.002052	0.016327	0.012580	0.661315

Table 12: Force pattern of behaviours (visualized in Fig. 10)

The distance matrix of the behaviour prototypes in abstraction space is given in Tab. 13. Fig. 11 visualizes them as Dendrogram. It illustrates the distances (dissimilarity) between different clusters and gives the points where a hierarchical clustering method would merge two clusters. If shows that b_{ftpush} and b_{pnp} cause the most similar change. b_{pull} and b_{push} are slightly less similar. The clusters that would result from merging b_{ftpush} and b_{pnp} and from merging b_{pull} and b_{push} would be merged next. The distance from the resulting cluster to the b_{turn} prototype is quite large. These distances can be used to estimate the number of clusters. In the rest of this evaluation a number of two clusters will be assumed, one containing the b_{turn} and one that contains the other behaviours.

Clustering behaviours in the abstraction space seems to result in abstract affordances that are intuitive for humans. It has to be further investigated if the number

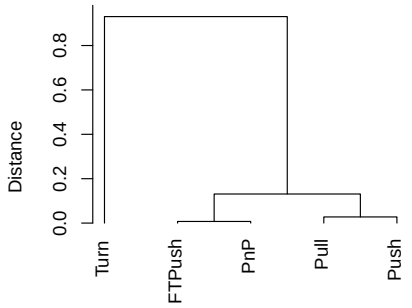


Figure 11: Dendrogram of behaviour prototypes in abstraction space

of clusters can always be identified as clearly as in the given prototype. In the next step the domains that contain the dominant change of each behaviour are calculated, discarding change beyond some threshold. This can be used as effect of the operator.

5.2. Dimension Weighting

Based on the abstraction generated in the last section, affordance-based planning operators are generated. As discussed in Sec. 3.3.2 the parameter-set of these operators include the initial and final values of the quality domains they change. Thus an operator is used to change some domain from the initial to the final value.

Looking at the clusters generated in the last section, the following parameter-sets are assumed for the operators.

```
changePosition(initPos, finalPos)
changeOrientation(iOri, fOri, iPos, fPos)
```

The instantiation method estimates the most promising robot behaviour to use for a given operator. Therefore, beside all initial dimensions, they work also on the final and δ values of the domains to change. This can be seen as the intention behind the operator (or here better: action).

How useful the dimensions are for the decision is estimated by methods from feature selection. The weight estimation given here were estimated using packages from the R system [30]. They are compared to an unweighted and a human weighted version to evaluate their influence on the decision.

The weights given here are computed using the *FSelector* package [33] that implements several feature selection methods, some by using the R interface to the *Weka* (*Waikato Environment for Knowledge Analysis*) software [20, 40].

The weights represent the 'influence' of

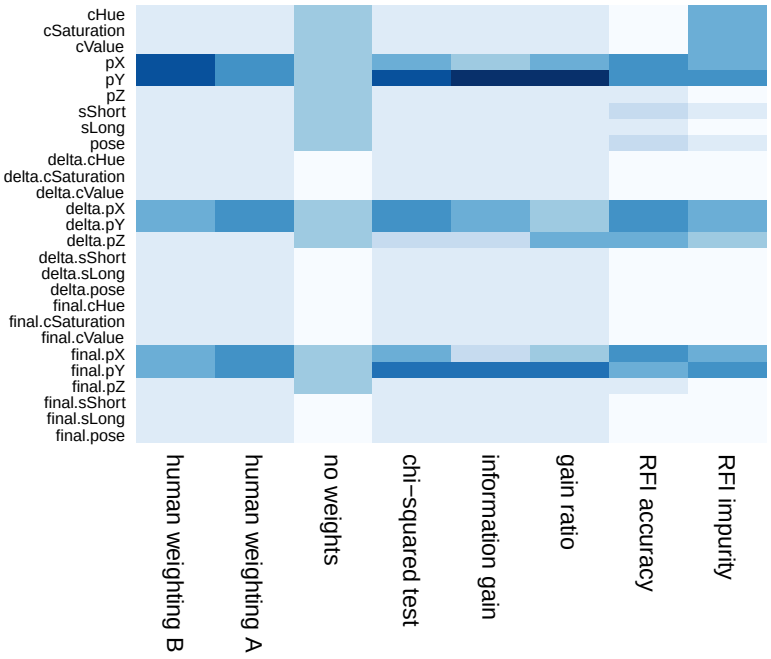


Figure 12: Dimension weights. The first five estimated by different techniques from feature selection. One weighting using all dimensions and two human-set weightings. The darker the blue, the more weight is on a dimension. The weights are scaling column-wise.

- the initial dimensions as well as
- the intended displacement (i. e. the final position given in the experiment) as δ and
- absolute value on the success-value.

The 'no-weights' weighting includes exactly the dimensions. It is computed using the full training set. As mentioned before, the question if a reduced training set that includes equal-sized subsets for every behaviour and behaviour-outcome leads to better results is not targeted here. However, a short evaluation is given in Appendix D. It seems that there are weightings that benefit from the full set and some that benefit from the reduced set.

Two human-set weightings are compared to the generated values.

In summary, all weightings accent the x and y-positions, just like the human weightings. The weights are visualized in Fig. 12 and given in Appendix C.

5.3. A Side Note on Cue Detection via k-NN Classification

Although in practice k-NN success prediction like given here could work quite well, there is one flaw: the agent behaviours are defined as cs concepts, i. e. the regions of

	b_{ftpush}		b_{pnp}		b_{pull}		b_{push}	
execution worked	true	false	true	false	true	false	true	false
cue detected	27	154	210	166	98	204	87	202
no cue detected	1	818	26	598	10	688	10	701

Table 14: Confusion matrix of k-NN cue detection for (human weighting A, k=1).

behaviour	AUC	sensitivity	specificity	pred. error
b_{ftpush}	0.90	0.964	0.842	0.155
b_{pnp}	0.84	0.890	0.783	0.192
b_{pull}	0.84	0.907	0.771	0.214
b_{push}	0.84	0.897	0.776	0.212

Table 15: Characteristic numbers of cue detection via k-NN classification (human weighting A, k=1).

applicability have to be convex. When applicability is defined by k-NN classification, it will most likely result in a non-convex applicability region.

For the instantiation step, on the over hand, the k-NN method is no problem, since it chooses between concepts whose applicability regions overlap.

Table 14 gives the confusion matrix of the classification using a human weighting that includes x and y positions (with a weight of 1.0) and no other dimension. The table has two columns for each behaviour, the left one gives cases when the execution worked in the experiment, the right one cases where it did not. Each of them has two rows giving the predictions. It can be summarized that number of false positives is quite height for all behaviours, i. e. that executions that have been predicted to work do not work in the experiment. Figure 13 gives the *receiver operating characteristic* (ROC) curves of the classifiers.

A ROC curve is a widely used way to visualize the performance of binary classification methods. It gives the classifier's sensitivity[5] and specificity[6]. The optimal ROC curve would be at the top left corner of the diagram. The performance is thereby measured by the *area under the curve* (AUC) that should be larger than 0.5. Otherwise the classifier is no better than random choice [22, p. 147].

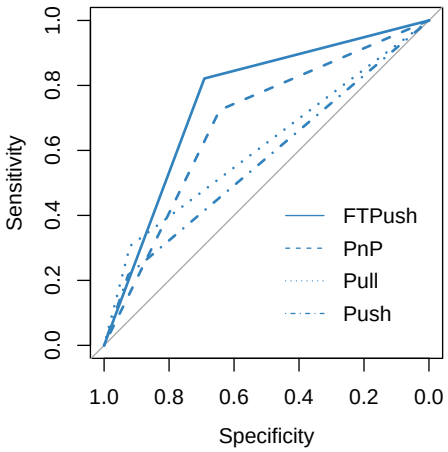
Figure 13 gives the ROC curves of classifiers using no weighting at all (top left), using a weighting based on information gain (top right) and a human defined weighting (bottom left).

To refer to the different classifiers, the following notation will be used: the first letters give the weighting, here e. g. NW for no weight, IG for information gain and HW for human weighting. Afterwards a number gives the neighbours that have been included into the decision.

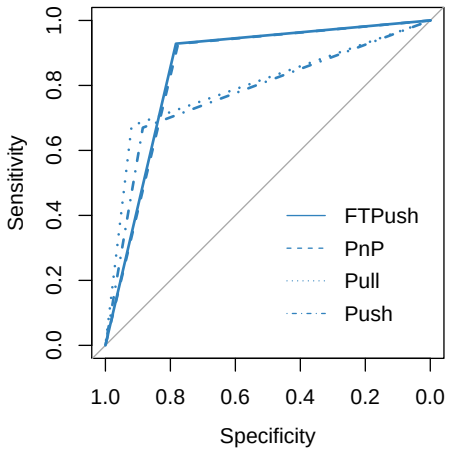
The HW1 classifier performs quite well for every behaviour. It reaches AUC

[5]sensitivity: probability of a cue detection given that the behaviour execution works

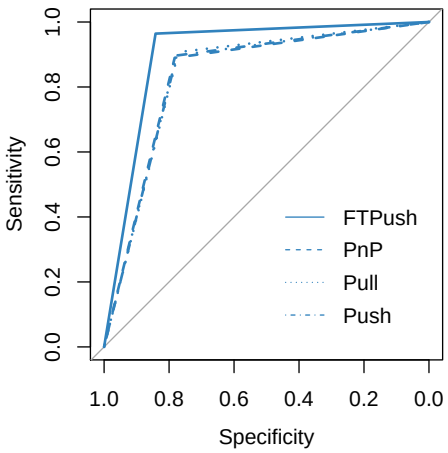
[6]specificity: probability of to get no cue detection given that the behaviour execution fails



no weighting, k=1



information gain, k=3



human weighting A, k=1

Figure 13: Receiver operating characteristic curves on cue detection

values between 0.84 and 0.90 and has a prediction error rate of 0.2 (see Tab. 15 for characteristic numbers). The IG3 classifier is worse in predicting b_{push} and b_{pull}, but with acceptable overall performance. A full table with results of all weightings as well as all ROC curves can be found in Appendix 5.3.

In summary, there exist weights that enable a quite good prediction of behaviour success. Thus when using this weights for measuring similarity, similar problem instances will result in the same behaviour outcome. A second thing to notice is that there are methods that are able to automatically generate dimension weights that outperform an unweighted prediction.

5.4. Action Instantiation and Substitution

The instantiation is most useful for abstract affordances that can be realized by a large number of robot behaviours. In the current prototype only the `changePosition` operator comes with more than one realization. So this evaluation is entirely based on instantiating `changePosition` actions.

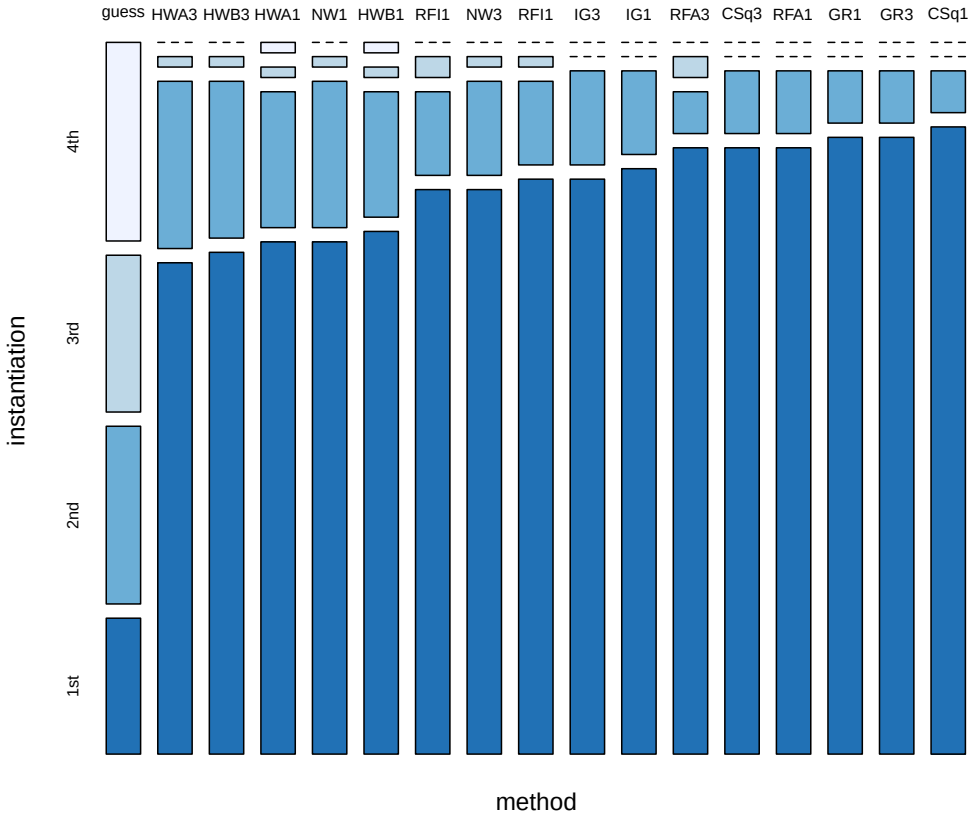
The first part discusses the instantiation of experiments that come with exactly one solution. There are four behaviours to chose. It is compared which choice (1^{st}, 2^{nd}, 3^{rd} or 4^{th}) is the right one. The results are visualized in Fig. 14. The height of the bars represents the number of cases where each choice was the successful behaviour. The darkest blue indicates the first choice and the blue becomes lighter with each choice. This means that the perfect instantiation would result only in one dark blue bar. To have a base-line, an instantiation has been implemented that randomly chooses some behaviour (label *guess* in the figure). In this evaluation the human weightings are outperformed by all other weightings.

It seems that the other weights 'prefer' b_{pnp} behaviour. In the far most instances of this evaluation set, the b_{pnp} behaviour is the only successful behaviour. So the preference of this behaviour results in a higher overall success.

Figure 15 gives the results for the evaluation set with two successful behaviours. To evaluate the performance for this case it is tested how often both successful behaviours are in the first two choices. The number of these cases is visualized by the darkest bar. For the cases where the two successful behaviours have not been chosen first, it is counted how often they are in the first three choices. This is visualized by the middle-blue bar. The lightest blue gives the rest of the cases, i. e. the cases where all choices have been needed to choose both successful behaviours.

This evaluation shows a greater spread between different weightings. The instantiation that uses no weights is nearly as bad as random choice. Here especially the chi-squared and information gain based weightings give good results. This is in line with the classification results (cf. Appendix B).

In summary, the proposed instantiation method outperforms random behaviour choice (guessing). There are methods to estimate dimension weights that improve the results compared to unweighted instantiation.



Legend

NW	no dimension weights
CSq	chi-squared test

entropy-based

GR	gain ratio
IG	information gain

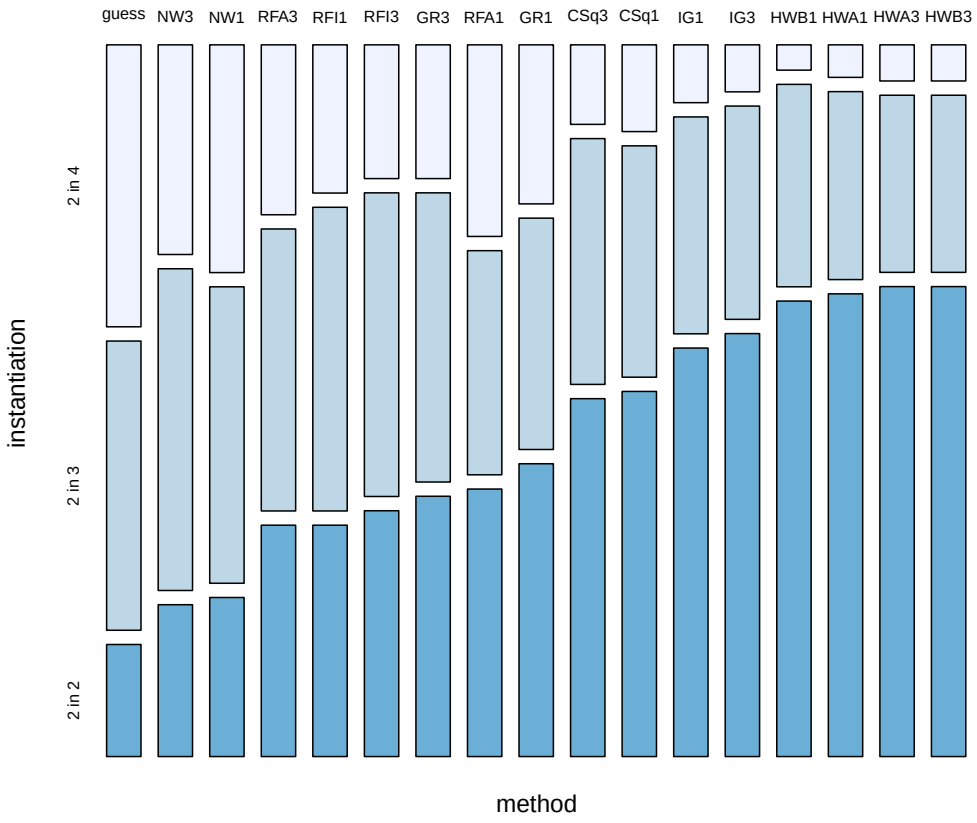
random forrest-based

RFA	random forrest importance, mean decrease in accuracy
RFI	random forrest importance, mean decrease in node impurity

human weighting

HWA	inital, final and delta values of x and y position are 1
HWB	inital position 1, final and delta values 0.5

Figure 14: Success of behaviour instantiation. Success rate of each choice for problems with one solution.



Legend

NW	no dimension weights
CSq	chi-squared test

entropy-based

GR	gain ratio
IG	information gain

random forrest-based

RFA	random forrest importance, mean decrease in accuracy
RFI	random forrest importance, mean decrease in node impurity

human weighting

HWA	inital, final and delta values of x and y position are 1
HWB	inital position 1, final and delta values 0.5

Figure 15: Success of behaviour substitution. The diagram gives problems that have (exactly) two successful behaviours (denoted with s_1 and s_2). The diagram shows $\mathrm{P}(\{s_1, s_2\} \subseteq \{b_1, b_2\})$, $\mathrm{P}(\{s_1, s_2\} \nsubseteq \{b_1, b_2\} \wedge \{s_1, s_2\} \subseteq \{b_1, b_2, b_3\})$ and $\mathrm{P}(\{s_1, s_2\} \nsubseteq \{b_1, b_2, b_3\} \wedge \{s_1, s_2\} \subseteq \{b_1, b_2, b_3, b_4\})$

6. Conclusion

This thesis extends the integration of affordance-based robot control and automated planning given by Lörken and Hertzberg by introducing a method to automate the abstraction step that groups affordances to abstract affordances. The focus of the original work, to be flexible in which (kind of) object to use to reach some goal was shifted to be flexible in which behaviour to use.

The presented approach is based on a newly introduced affordance representation, introduced in Sec. 3.2 that is based on Gärdenfors' conceptual spaces. The grouping, discussed in Sec. 3.3.1, was realized in line with the affordance-inspired function-centred view on the environment. Behaviours that result in similar change of the environment are grouped together. The generation of PDDL operators was excluded due to the limited time, but general directions were discussed in Sec. 3.3.2. The formulation of a planning instance based on the affordance operators is somehow delicate. One has to find a compromise between the sparse model that is typically used by the affordance-inspired approach and a sufficient amount of information that enables a planning system to find significant plans. This is the main task for future work.

After a domain-independent planning system has generated a plan based on the generated domain, all affordance-based operators have to be instantiated by some affordance. This means it has to be decided which behaviour is used to reach the desired goal. The original approach chooses the behaviours randomly. Section 3.3.3 contributes a k-NN-based method to prefer behaviours with higher probability to establish the intended outcome. This is also realized based on the conceptual spaces representation. It is used to find similar situations from the past. The assumption is that behaviours that worked in such similar situations will also work in the current situation. Gärdenfors proposes to use weighted dimensions to adapt the similarity measurement to a specific context. In this work it was proposed to use techniques from feature selection for weighting.

A prototype was implemented based on the 3D simulation system OpenRAVE that was used for evaluation. Based on a Care-O-bot 3, a *pick and place*, *pull*, two distinct *push* and a *turn* behaviour have been implemented. As quality domains the *colour*, *size*, *orientation* and *relative position* of the manipulated entity are extracted from simulation.

The evaluation showed that the proposed abstraction space results in human intuitive abstract operators. The instantiation method outperformed a random behaviour choice. Weights were calculated using the FSelector package [33] of the *R* environment for statistical computing [30]. The evaluation showed that weights generated with these methods further increase the success rate of the instantiation.

To further evaluate the introduced similarity measurement, a k-NN classification system was introduced that predicts behaviour applicability. Though this can be seen as cue detection, this contradicts Gärdenfors' representation because it does not ensure region convexity.

One line of future work is to realize the remaining steps to enable the agent to

learn new behaviours autonomously and apply them without human help. The first thing to do is to automate the operator generation. Here the features of conceptual spaces have to be further exploited to generate a symbolic representation. Another interesting question is whether the quality dimensions/domains can be extracted automatically from sensory data.

As seen in the presented work, the new conceptual layer between the symbolic and subsymbolic layer enables similarity estimation. This is not possible in entirely symbolic planning in this form. So another thing to investigate is how the overall process can further benefit from this new form of deliberation.

A step towards real-world application can be done by integrating the approach into *Jenny*, the robot at Bonn-Rhein-Sieg University, and the work of Iman Awaad [2]. To do this, a representation in a semantic markup language as the *conceptual space markup language* (introduced in [1]) might be helpful. Most gainful would be the transfer of a trained system from simulated to real robot to reduce costs of the experiment runs.

A. Experiment Data Sets

The following subsections give some characteristic numbers for the quality dimensions before any preprocessing. For most of them, the initial character indicates the quality domain it belongs to (`c` for `colour` or `p` for `position`). The numbers have been generated using [30].

A.1. Training Set

```
 behaviour       expName      success          cHue
ftpush:3850   exp0001:   1   False:6973   Min.   :  0.0167
pnp   : 750   exp0002:   1   True : 577   1st Qu.: 90.0852
pull  :1150   exp0003:   1               Median :178.7650
push  :1100   exp0004:   1               Mean   :179.5544
turn  : 700   exp0005:   1               3rd Qu.:270.2681
              exp0006:   1               Max.   :359.9979
              (Other):7544
 cSaturation          cValue              pX
Min.   :0.006143   Min.   :0.02417   Min.   :-0.751462
1st Qu.:0.494576   1st Qu.:0.63473   1st Qu.:-0.373831
Median :0.702381   Median :0.79368   Median :-0.003175
Mean   :0.663182   Mean   :0.75083   Mean   : 0.002494
3rd Qu.:0.865084   3rd Qu.:0.90978   3rd Qu.: 0.380388
Max.   :0.999956   Max.   :0.99993   Max.   : 0.751318

      pY                pZ            sShort
Min.   :-1.3099   Min.   :0.8390   Min.   :0.020
1st Qu.:-1.0812   1st Qu.:0.8415   1st Qu.:0.025
Median :-0.8521   Median :0.8490   Median :0.040
Mean   :-0.8536   Mean   :0.8540   Mean   :0.055
3rd Qu.:-0.6256   3rd Qu.:0.8690   3rd Qu.:0.100
Max.   :-0.3952   Max.   :0.8690   Max.   :0.100

    sLong             pose          final.cHue
Min.   :0.0700   Min.   :0.2857   Min.   :  0.0167
1st Qu.:0.0725   1st Qu.:0.3393   1st Qu.: 90.0852
Median :0.0800   Median :0.5000   Median :178.7650
Mean   :0.0850   Mean   :0.6131   Mean   :179.5544
3rd Qu.:0.1000   3rd Qu.:1.0000   3rd Qu.:270.2681
Max.   :0.1000   Max.   :1.0000   Max.   :359.9979

final.cSaturation   final.cValue        final.pX
Min.   :0.006143   Min.   :0.02417   Min.   :-0.7510581
1st Qu.:0.494576   1st Qu.:0.63473   1st Qu.:-0.3708308
```

```
Median :0.702381   Median :0.79368   Median : 0.0004907
Mean   :0.663182   Mean   :0.75083   Mean   :-0.0003116
3rd Qu.:0.865084   3rd Qu.:0.90978   3rd Qu.: 0.3735401
Max.   :0.999956   Max.   :0.99993   Max.   : 0.7511941

   final.pY          final.pZ       final.sShort
Min.   :-1.3099   Min.   :0.7927   Min.   :0.02000
1st Qu.:-1.0839   1st Qu.:0.8490   1st Qu.:0.04000
Median :-0.8581   Median :0.8590   Median :0.05933
Mean   :-0.8564   Mean   :0.8549   Mean   :0.05516
3rd Qu.:-0.6277   3rd Qu.:0.8690   3rd Qu.:0.10000
Max.   :-0.3930   Max.   :0.9590   Max.   :0.15115

 final.sLong        final.pose       delta.cHue delta.cSaturation
Min.   :0.07000   Min.   :0.2854   Min.   :0    Min.   :0
1st Qu.:0.07654   1st Qu.:0.5000   1st Qu.:0    1st Qu.:0
Median :0.09000   Median :0.6667   Median :0    Median :0
Mean   :0.08513   Mean   :0.6351   Mean   :0    Mean   :0
3rd Qu.:0.10000   3rd Qu.:1.0000   3rd Qu.:0    3rd Qu.:0
Max.   :0.16494   Max.   :3.5000   Max.   :0    Max.   :0

 delta.cValue    delta.pX            delta.pY
Min.   :0     Min.   :-1.474344   Min.   :-0.900508
1st Qu.:0     1st Qu.:-0.388587   1st Qu.:-0.232826
Median :0     Median : 0.000000   Median : 0.000000
Mean   :0     Mean   :-0.002805   Mean   :-0.002849
3rd Qu.:0     3rd Qu.: 0.375594   3rd Qu.: 0.230614
Max.   :0     Max.   : 1.480604   Max.   : 0.893610

   delta.pZ           delta.sShort        delta.sLong
Min.   :-0.0462554   Min.   :0.0000000   Min.   :0.0000000
1st Qu.: 0.0000000   1st Qu.:0.0000000   1st Qu.:0.0000000
Median : 0.0000000   Median :0.0000000   Median :0.0000000
Mean   : 0.0008927   Mean   :0.0001577   Mean   :0.0001324
3rd Qu.: 0.0000000   3rd Qu.:0.0000000   3rd Qu.:0.0000000
Max.   : 0.1000038   Max.   :0.0511488   Max.   :0.0649352

  delta.pose
Min.   :-0.09505
1st Qu.: 0.00000
Median : 0.00000
Mean   : 0.02199
3rd Qu.: 0.00000
Max.   : 3.21427
```

A.2. Evaluation Set

```
 behaviour             expName      success          cHue
ftpush:1000   exp0001ftpush:   1   False:3531   Min.   :  0.585
pnp   :1000   exp0001pnp   :   1   True : 469   1st Qu.: 84.526
pull  :1000   exp0001pull  :   1               Median :175.549
push  :1000   exp0001push  :   1               Mean   :177.088
              exp0002ftpush:   1               3rd Qu.:270.014
              exp0002pnp   :   1               Max.   :359.239
              (Other)      :3994
 cSaturation          cValue              pX
Min.   :0.01701   Min.   :0.08691   Min.   :-0.751398
1st Qu.:0.48887   1st Qu.:0.63463   1st Qu.:-0.391874
Median :0.70780   Median :0.79321   Median : 0.013618
Mean   :0.66261   Mean   :0.74905   Mean   : 0.004079
3rd Qu.:0.85918   3rd Qu.:0.90892   3rd Qu.: 0.380275
Max.   :0.99916   Max.   :0.99920   Max.   : 0.750404

      pY                pZ            sShort
Min.   :-1.3096   Min.   :0.8390   Min.   :0.02000
1st Qu.:-1.0828   1st Qu.:0.8390   1st Qu.:0.02000
Median :-0.8523   Median :0.8490   Median :0.04000
Mean   :-0.8490   Mean   :0.8539   Mean   :0.05484
3rd Qu.:-0.6127   3rd Qu.:0.8590   3rd Qu.:0.06000
Max.   :-0.3955   Max.   :0.8690   Max.   :0.10000

    sLong              pose          final.cHue
Min.   :0.07000   Min.   :0.2857   Min.   :  0.585
1st Qu.:0.07000   1st Qu.:0.2857   1st Qu.: 84.526
Median :0.08000   Median :0.5000   Median :175.549
Mean   :0.08493   Mean   :0.6116   Mean   :177.088
3rd Qu.:0.09000   3rd Qu.:0.6667   3rd Qu.:270.014
Max.   :0.10000   Max.   :1.0000   Max.   :359.239

final.cSaturation  final.cValue        final.pX
Min.   :0.01701   Min.   :0.08691   Min.   :-0.75110
1st Qu.:0.48887   1st Qu.:0.63463   1st Qu.:-0.40345
Median :0.70780   Median :0.79321   Median :-0.01719
Mean   :0.66261   Mean   :0.74905   Mean   :-0.01012
3rd Qu.:0.85918   3rd Qu.:0.90892   3rd Qu.: 0.35947
Max.   :0.99916   Max.   :0.99920   Max.   : 0.74839

   final.pY          final.pZ       final.sShort
Min.   :-1.3099   Min.   :0.8383   Min.   :0.02000
```

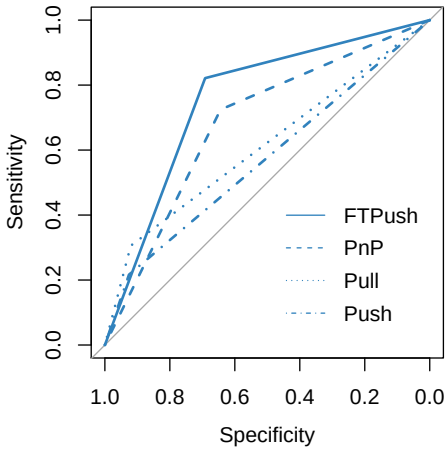
```
1st Qu.:-1.1062   1st Qu.:0.8490   1st Qu.:0.02162
Median :-0.8553   Median :0.8590   Median :0.04403
Mean   :-0.8572   Mean   :0.8552   Mean   :0.05503
3rd Qu.:-0.6229   3rd Qu.:0.8690   3rd Qu.:0.06991
Max.   :-0.3947   Max.   :0.9044   Max.   :0.14781

 final.sLong        final.pose       delta.cHue delta.cSaturation
Min.   :0.07000   Min.   :0.2836   Min.   :0    Min.   :0
1st Qu.:0.07082   1st Qu.:0.3070   1st Qu.:0    1st Qu.:0
Median :0.08280   Median :0.5353   Median :0    Median :0
Mean   :0.08511   Mean   :0.6127   Mean   :0    Mean   :0
3rd Qu.:0.09688   3rd Qu.:0.7320   3rd Qu.:0    3rd Qu.:0
Max.   :0.16196   Max.   :1.1601   Max.   :0    Max.   :0

 delta.cValue    delta.pX            delta.pY
Min.   :0     Min.   :-1.469542   Min.   :-0.866875
1st Qu.:0     1st Qu.:-0.448111   1st Qu.:-0.274161
Median :0     Median : 0.001955   Median :-0.016321
Mean   :0     Mean   :-0.014200   Mean   :-0.008203
3rd Qu.:0     3rd Qu.: 0.419043   3rd Qu.: 0.272658
Max.   :0     Max.   : 1.422535   Max.   : 0.892865

   delta.pZ           delta.sShort        delta.sLong
Min.   :-0.0007373   Min.   :0.0000000   Min.   :0.0000000
1st Qu.: 0.0000000   1st Qu.:0.0000000   1st Qu.:0.0000000
Median : 0.0000000   Median :0.0000000   Median :0.0000000
Mean   : 0.0013062   Mean   :0.0001866   Mean   :0.0001779
3rd Qu.: 0.0000000   3rd Qu.:0.0000000   3rd Qu.:0.0000000
Max.   : 0.0358697   Max.   :0.0478051   Max.   :0.0619598

  delta.pose
Min.   :-0.087397
1st Qu.: 0.000000
Median : 0.000000
Mean   : 0.001113
3rd Qu.: 0.000000
Max.   : 0.243908
```

B. Cue Detection

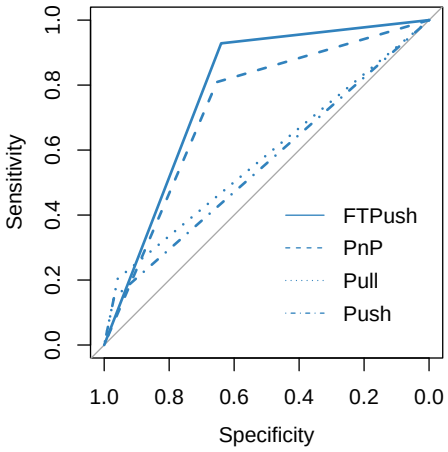
		noW1	cueChiSq1	cueInfGain1	cueGainRatio1	cueRf11	cueRf21	cuemyW1	cuemyW21
Area under ROC	FTPush	0,7564	0,8892	0,906	0,8882	0,8935	0,828	0,9029	0,9029
	PnP	0,6843	0,8456	0,8417	0,8419	0,8064	0,7714	0,8363	0,8348
	Pull	0,613	0,6623	0,727	0,5691	0,6448	0,6443	0,8394	0,8208
	Push	0,5711	0,6387	0,7016	0,5925	0,5769	0,635	0,8366	0,8412
Error rate	FTPush	0,305	0,148	0,149	0,15	0,207	0,267	0,155	0,155
	PnP	0,337	0,18	0,186	0,179	0,231	0,262	0,192	0,192
	Pull	0,146	0,116	0,124	0,116	0,111	0,141	0,214	0,218
	Push	0,143	0,129	0,14	0,121	0,141	0,144	0,212	0,212
Average AuC		0,6562	0,75895	0,794075	0,722925	0,7304	0,719675	0,8538	0,849925
Average Error		0,23275	0,14325	0,14975	0,1415	0,1725	0,2035	0,19325	0,19425

		noW3	cueChiSq3	cueInfGain3	cueGainRatio3	cueRf13	cueRf23	cuemyW3	cuemyW23
Area under ROC	FTPush	0,7842	0,8568	0,8557	0,8501	0,8525	0,8503	0,8782	0,8746
	PnP	0,7312	0,8464	0,8521	0,8544	0,7967	0,784	0,8488	0,8511
	Pull	0,5811	0,6557	0,7976	0,5633	0,5808	0,6479	0,853	0,8437
	Push	0,5572	0,6781	0,7775	0,5314	0,5206	0,6196	0,846	0,8355
Error rate	FTPush	0,352	0,211	0,213	0,224	0,253	0,291	0,203	0,21
	PnP	0,31	0,19	0,188	0,18	0,239	0,263	0,193	0,185
	Pull	0,123	0,106	0,107	0,104	0,109	0,12	0,226	0,228
	Push	0,11	0,116	0,136	0,115	0,118	0,122	0,22	0,239
Average AuC		0,663425	0,75925	0,820725	0,6998	0,68765	0,72545	0,8565	0,851225
Average Error		0,22375	0,15575	0,161	0,15575	0,17975	0,199	0,2105	0,2155

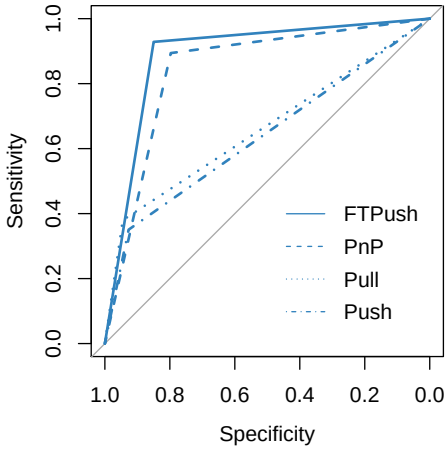
no weighting k=1



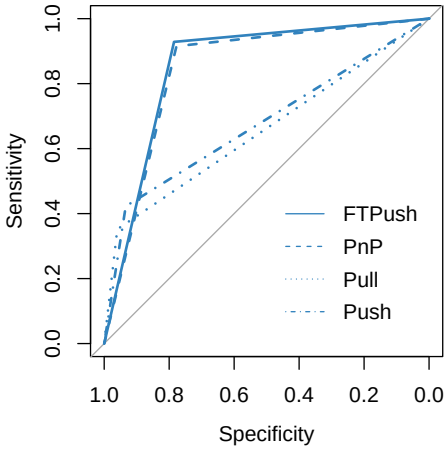
no weighting B k=3



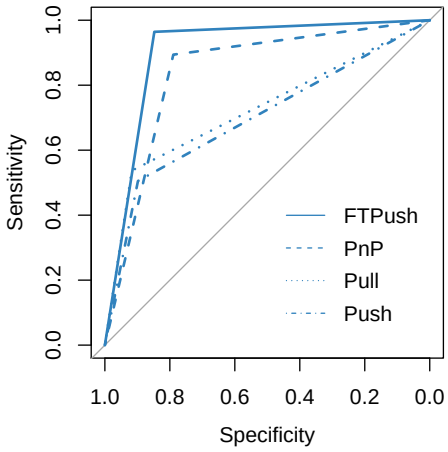
chi square k=1



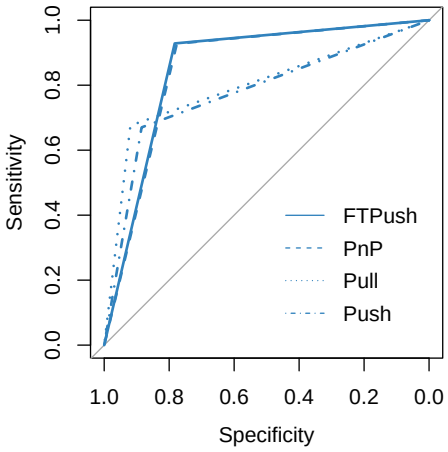
chi square k=3



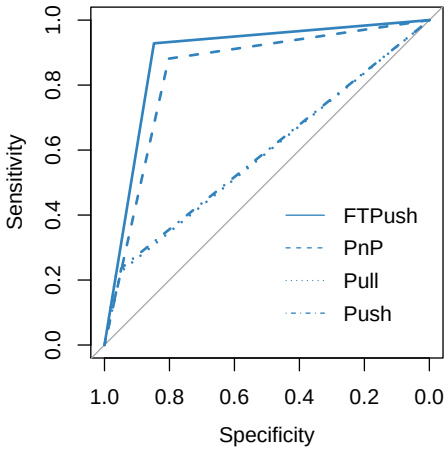
information gain k=1



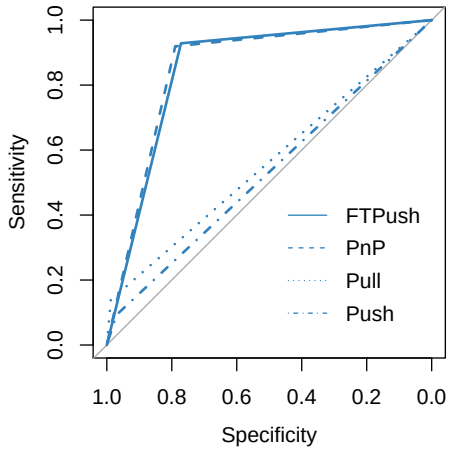
information gain k=3



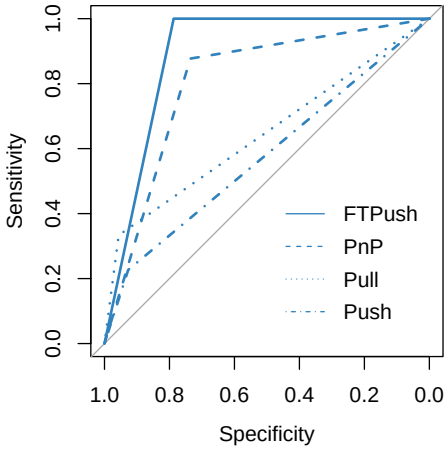
gain ratio k=1



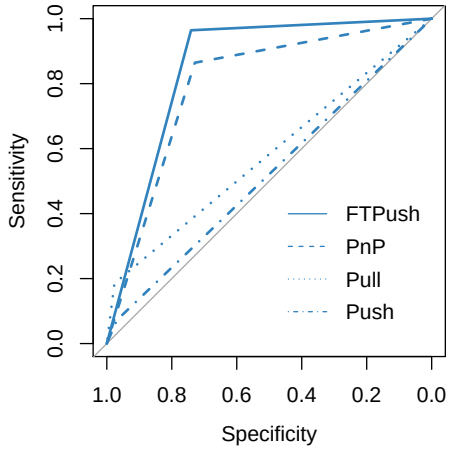
gain ratio k=3



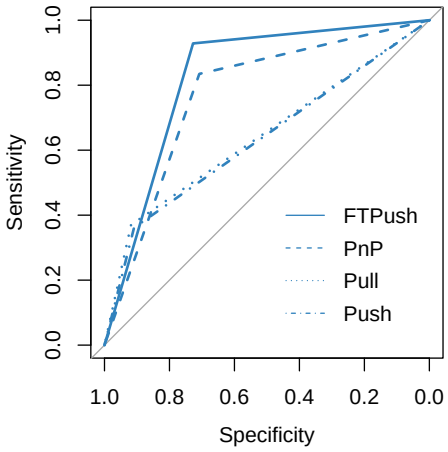
random forrest 1 k=1



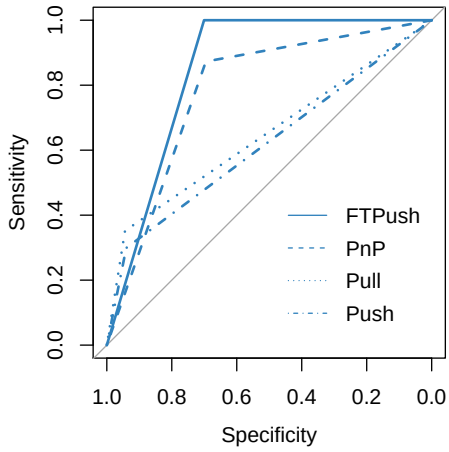
random forrest 1 k=3



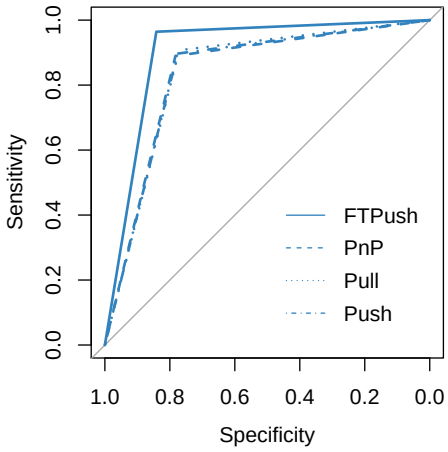
random forrest 2 k=1



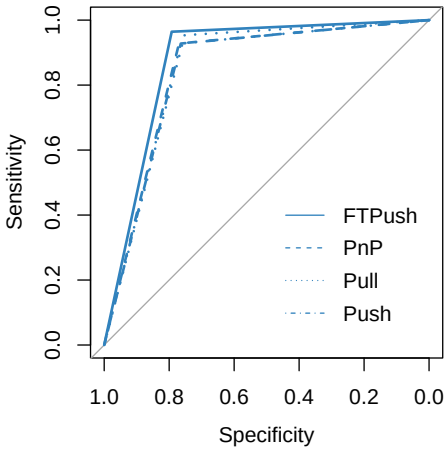
random forrest 2 k=3



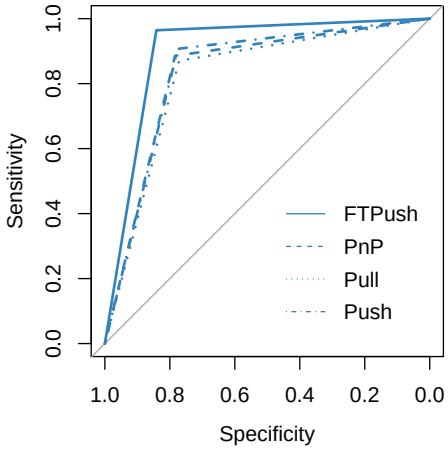
human weighting A k=1



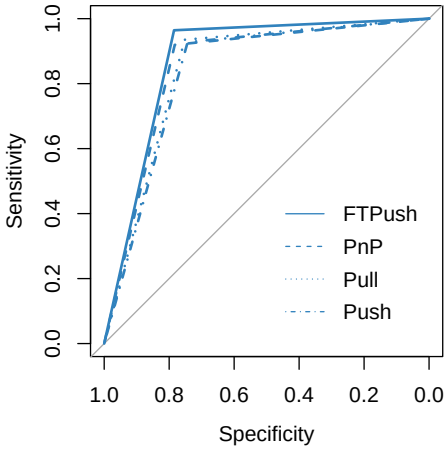
human weighting A k=3



human weighting B k=1



human weighting B k=3



C. Dimension Weights

	human weighting B	human weighting A	no weights	chi-squared test	information gain	gain ratio	RFI accuracy	RFI impurity
cHue	0,0	0,0	1,0	0,0	0,0	0,0	-1,205	73,8828
cSaturation	0,0	0,0	1,0	0,0	0,0	0,0	0,6091	76,2682
cValue	0,0	0,0	1,0	0,0	0,0	0,0	0,7452	76,4834
pX	1,0	1,0	1,0	0,0937	0,0086	0,0137	41,6504	88,0336
pY	1,0	1,0	1,0	0,1645	0,0285	0,0329	48,0507	103,3583
pZ	0,0	0,0	1,0	0,0	0,0	0,0	10,3835	8,5982
sShort	0,0	0,0	1,0	0,0	0,0	0,0	14,0496	9,8442
sLong	0,0	0,0	1,0	0,0	0,0	0,0	11,1404	8,5001
pose	0,0	0,0	1,0	0,0	0,0	0,0	14,5245	14,2379
delta.cHue	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
delta.cSaturation	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
delta.cValue	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
delta.pX	0,5	1,0	1,0	0,115	0,0128	0,0097	44,8063	85,1052
delta.pY	0,5	1,0	1,0	0,1063	0,0113	0,0088	41,8076	86,5968
delta.pZ	0,0	0,0	1,0	0,0461	0,0029	0,0164	41,0905	70,0613
delta.sShort	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
delta.sLong	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
delta.pose	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
final.cHue	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
final.cSaturation	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
final.cValue	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
final.pX	0,5	1,0	1,0	0,0817	0,0064	0,0108	50,7462	81,2218
final.pY	0,5	1,0	1,0	0,1388	0,0198	0,0249	37,818	106,7699
final.pZ	0,0	0,0	1,0	0,0	0,0	0,0	10,0571	8,4087
final.sShort	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
final.sLong	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
final.pose	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0

D. Instantiation and Substitution Results

instantiation

	succ1	succ2	succ3	succ4
guess	13	17	15	19
HWA3	47	16	1	0
HWB3	48	15	1	0
HWA1	49	13	1	1
NW1	49	14	1	0
HWB1	50	12	1	1
RFI1	54	8	2	0
NW3	54	9	1	0
RFI1	55	8	1	0
IG3	55	9	0	0
IG1	56	8	0	0
RFA3	58	4	2	0
CSq3	58	6	0	0
RFA1	58	6	0	0
GR1	59	5	0	0
GR3	59	5	0	0
CSq1	60	4	0	0

substitution

	2 in 2	2 in 3	2 in 4
guess	31	80	78
NW3	42	89	58
NW1	44	82	63
RFA3	64	78	47
RFI1	64	84	41
RFI3	68	84	37
GR3	72	80	37
RFA1	74	62	53
GR1	81	64	44
CSq3	99	68	22
CSq1	101	64	24
IG1	113	60	16
IG3	117	59	13
HWB1	126	56	7
HWA1	128	52	9
HWA3	130	49	10
HWB3	130	49	10

reduced training set					full training set					winn/loss			
instantiation													
	succ1	succ2	succ3	succ4		succ1	succ2	succ3	succ4	succ1	succ2	succ3	succ4
CSq1	60	4	0	0	CSq1	60	4	0	0	0	0	0	0
CSq3	58	6	0	0	CSq3	58	6	0	0	0	0	0	0
GR1	57	7	0	0	GR1	59	5	0	0	2	-2	0	0
GR3	58	6	0	0	GR3	59	5	0	0	1	-1	0	0
IG1	57	7	0	0	IG1	56	8	0	0	-1	1	0	0
IG3	55	9	0	0	IG3	55	9	0	0	0	0	0	0
RFA1	57	6	1	0	RFA1	58	6	0	0	1	0	-1	0
RFA3	56	7	1	0	RFA3	58	4	2	0	2	-3	1	0
RFI1	52	11	1	0	RFI1	54	8	2	0	2	-3	1	0
RFI1	55	8	1	0	RFI1	55	8	1	0	0	0	0	0
substitution													
	2 in 2	2 in 3	2 in 4			2 in 2	2 in 3	2 in 4		2 in 2	2 in 3	2 in 4	
CSq1	97	67	25		CSq1	101	64	24		4	-3	-1	
CSq3	93	74	22		CSq3	99	68	22		6	-6	0	
GR1	77	65	47		GR1	81	64	44		4	-1	-3	
GR3	71	72	46		GR3	72	80	37		1	8	-9	
IG1	106	64	19		IG1	113	60	16		7	-4	-3	
IG3	105	62	22		IG3	117	59	13		12	-3	-9	
RFA1	87	64	38		RFA1	74	62	53		-13	-2	15	
RFA3	87	80	22		RFA3	64	78	47		-23	-2	25	
RFI1	85	74	30		RFI1	64	84	41		-21	10	11	
RFI3	91	71	27		RFI3	68	84	37		-23	13	10	

The full training set contains far more failed instances than succeded. It also contains different numbers of instances for each behaviour. In section 5 the full training set is used as input for the weight calculation.

However, a second version that uses only a reduced training set could be considered. This table gives a comparison of the instantiation and substitution using a reduced (left) and the full training set (in the middle). The reduced training set contains 100 successful and 100 failed instances for each behaviour, it is the same set that is used for k-NN in the instantiation.

The right part shows the win/loss between the two versions.

E. Erklärung

Ich versichere an Eides statt, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit verwendet habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Ort, Datum	Unterschrift

Index

References

[1] Benjamin Adams and Martin Raubal. Conceptual Space Markup Language (CSML): Towards the Cognitive Semantic Web. In *Proceedings of the 3rd IEEE International Conference on Semantic Computing (ICSC 2009)*, pages 253–260, Berkeley, USA, September 14–16, 2009.

[2] Iman Awaad, Gerhard K. Kraetzschmar, and Joachim Hertzberg. Affordance-Based Reasoning in Robot Task Planning. In *Planning and Robotics Workshop (PlanRob), 23rd International Conference on Automated Planning and Scheduling (ICAPS)*, Rome, Italy, June 2013.

[3] Oliver Beyer, Philipp Cimiano, and Sascha Griffiths. Towards Action Representation within the Framework of Conceptual Spaces: Preliminary Results. In *Proceedings of the 8th International Cognitive Robotics Workshop at the AAAI*, Toronto, July 2012.

[4] Bonn-Rhein-Sieg University. b-it-bots Webside. www.b-it-bots.de/.

[5] Richard Cubek and Wolfgang Ertel. Learning and Application of High-Level Concepts with Conceptual Spaces and PDDL. In *3rd Workshop on Learning and Planning, ICAPS (21st International Conference on Automated Planning and Scheduling)*, Freiburg, Germany, 2011.

[6] Rosen Diankov. OpenRAVE Webside. http://openrave.org/.

[7] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[8] Stefan Edelkamp and Jörg Hoffmann. PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, 2004.

[9] Maria Fox and Derek Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, (20):61–124, 2003.

[10] Fraunhofer-Institut für Produktionstechnik und Automatisierung. Care-O-bot 3 Webside. www.care-o-bot.de/.

[11] Fraunhofer Institute for Intelligent Analysis and Information Systems IAIS. MACS - A Research Project for Exploring and Exploiting the Concept of Affordances for Robot Control – Webside. http://www.macs-eu.org/.

[12] Peter Gärdenfors. *Conceptual Spaces – The Geometry of Thought*. MIT Press, 2004.

[13] Peter Gärdenfors and Massimo Warglien. Using Conceptual Spaces to Model Actions and Events. *Journal of Semantics*, April 2012.

[14] Alfonso Gerevini and Derek Long. Plan Constraints and Preferences in PDDL3. Technical Report R. T. 2005-08-47, Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia, 2005.

[15] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning – Theory and Practice*. Morgan Kaufmann Publishers, May 2004.

[16] James J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, Boston, MA, 1979.

[17] Isabelle Guyon and André Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

[18] Ronny Hartanto. *Fusing DL Reasoning with HTN Planning as a Deliberative Layer in Mobile Robotics*. PhD thesis, Universität Osnabrück, March 2010.

[19] Daniel Höller and Christoph Mies. Integration of a PDDL Planning Framework into the Multi-Agent Simulation System LAMPSys. In Jürgen Sauer, Stefan Edelkamp, and Bernd Schattenberg, editors, *KI 2011. Workshop Proceedings, 26th PuK Workshop 'Planen, Scheduling und Konfigurieren, Entwerfen' (Online)*, October 4-7, 2011.

[20] Kurt Hornik, Christian Buchta, and Achim Zeileis. Open-source machine learning: R meets weka. *Computational Statistics*, 24(2):225–232, May 2009.

[21] International Planning Competition. Overview: Papers on the Planning Domain Definition Language. http://ipc.informatik.uni-freiburg.de/PddlResources.

[22] Gareth James, Daniela Witten, Travor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*, volume 103 of *Springer Texts in Statistics*.

[23] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, 2005.

[24] Christopher Lörken and Joachim Hertzberg. Grounding Planning Operators by Affordances. In *International Conference on Cognitive Systems (CogSys 2008)*, pages 79–84, Karlsruhe, Germany, April 2-4, 2008.

[25] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control, 1998.

[26] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, New York, USA, 1997.

[27] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, IJCAI'99, pages 968–973, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[28] Dana S. Nau. Current Trends in Automated Planning. *AI Magazine*, 28(4):43–58, 2007.

[29] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[30] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[31] Martin Raubal. Formalizing Conceptual Spaces. In Achille Varzi and Laure Vieu, editors, *Proceedings of the 3rd International Conference on Formal Ontology in Information Systems (FOIS 2004)*, pages 153–164, October 2004.

[32] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, Massachusetts, MA, 2001.

[33] Piotr Romanski. *FSelector: Selecting Attributes*, 2013. R package version 0.19.

[34] Erich Rome, Joachim Hertzberg, and Georg Dorffner, editors. *Proceedings of the 2006 international conference Towards Affordance-Based Robot Control*, Berlin, Heidelberg, 2008. Springer-Verlag.

[35] Erich Rome, Lucas Paletta, Erol Şahin, Georg Dorffner, Joachim Hertzberg, Ralph Breithaupt, Gerald Fritz, Jörg Irran, Florian Kintzler, Christopher Lörken, Stefan May, and Emre Uğur. The MACS Project: An Approach to Affordance-Inspired Robot Control. In *Proceedings of the 2006 international conference Towards Affordance-Based Robot Control*, pages 173–210, Berlin, Heidelberg, 2008. Springer-Verlag.

[36] ROS community. ROS Webside. www.ros.org/.

[37] Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. Filter Methods for Feature Selection – A Comparative Study. In Hujun Yin, Peter Tino, Emilio Corchado, Will Byrne, and Xin Yao, editors, *Proceedings of the 8th international conference on Intelligent Data Engineering and Automated Learning*, volume 4881 of *IDEAL 2007*, pages 178–187, Berlin, Heidelberg, 2007. Springer-Verlag.

[38] Bernd Schattenberg. *Hybrid Planning and Scheduling*. PhD thesis, Ulm University, 2009.

[39] Emre Ugur, Erol Şahin, and Erhan Oztop. Unsupervised Learning of Object Affordances for Planning in a Mobile Manipulation Platform. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA 2011)*, pages 4312–4317, Shanghai, China, May 9–13, 2011.

[40] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[41] Rainer Worst. KURT2: A Mobile Platform for Research in Robotics. In *Proceedings of the 2nd International Symposium on Autonomous Minirobots for Research and Edutainment: AMiRE 2003*, February 18–20, 2003.