Robotics and Autonomous Systems 99 (2018) 148–165

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Integrating multi-purpose natural language understanding, robot's memory, and symbolic planning for task execution in humanoid robots

Mirko Wächter [a,*], Ekaterina Ovchinnikova [a], Valerij Wittenbeck [a], Peter Kaiser [a], Sandor Szedmak [d], Wail Mustafa [c], Dirk Kraft [c], Norbert Krüger [c], Justus Piater [b], Tamim Asfour [a]

[a] *Karlsruhe Institute of Technology (KIT), Adenauerring 2, 76131, Karlsruhe, Germany*
[b] *University of Innsbruck (UIBK), Technikerstr. 21a, 6020 Innsbruck, Austria*
[c] *University of Southern Denmark (SDU), Campusvej 55, 5230 Odense M, Denmark*
[d] *Aalto University, Konemiehentie 2, 6020 Espoo, Finland*

ARTICLE INFO

Article history:
Received 9 February 2017
Received in revised form 11 September 2017
Accepted 18 October 2017
Available online 14 November 2017

Keywords:
Structural bootstrapping
Natural language understanding
Planning
Task execution
Object replacement
Humanoid robotics

ABSTRACT

We propose an approach for instructing a robot using natural language to solve complex tasks in a dynamic environment. In this study, we elaborate on a framework that allows a humanoid robot to understand natural language, derive symbolic representations of its sensorimotor experience, generate complex plans according to the current world state, and monitor plan execution. The presented development supports replacing missing objects and suggesting possible object locations. It is a realization of the concept of structural bootstrapping developed in the context of the European project *Xperience*. The framework is implemented within the robot development environment *ArmarX*. We evaluate the framework on the humanoid robot ARMAR-III in the context of two experiments: a demonstration of the real execution of a complex task in the kitchen environment on ARMAR-III and an experiment with untrained users in a simulation environment.

1. Introduction

One of the goals of the humanoid robotics research is to model human-like information processing and the underlying mechanisms for dealing with the real world. This especially concerns the ability to communicate and collaborate with humans, adapt to changing environments, apply available knowledge in previously unseen situations. The concept of *structural bootstrapping* introduced in the context of the *Xperience* project [1] addresses mechanisms of employing semantic and syntactic similarity to infer which entities can replace each other with respect to certain roles. For example, if the robot is asked to bring a lemonade but cannot find it in the kitchen, the robot can suggest to replace it with another beverage, e.g. a juice, based on the similarity of both objects being drinkable. Thus, structural bootstrapping allows the robot to reason about an observed novel entity and its potential functionality and features. Earlier experiments demonstrated how structural bootstrapping can be applied at different levels of a robotic architecture including a sensorimotor level, a symbol-to-signal mediator level, and a planning level [2–4].

Structural bootstrapping is related to the concept of *affordances* — latent "action possibilities" available to an agent towards an object, given agent capabilities and the environment [5]. For example, a bowl can afford pouring into it or stirring in it and a knife can afford cutting with it. Object affordances can also be used to support object categorization and infer potential object replacements. For example, if two containers afford pouring into them, then they are interchangeable towards this action. In the context of structural bootstrapping, the interplay between the symbolic encoding of the sensorimotor information, prior knowledge, planning, plan execution monitoring, and natural language understanding plays a significant role. Natural language (NL) commands and comments can be used to set goals for the robot, update its knowledge, and provide it with feedback. Symbolic representations of object affordances can be used for object replacement. Symbolic representations of robot's observations are required to generate realistic plans. Plan execution monitoring is needed to

[*] Corresponding author.
E-mail addresses: Waechter@kit.edu (M. Wächter), Ovchinnikova@kit.edu (E. Ovchinnikova), Wittenbeck@kit.edu (V. Wittenbeck), Kaiser@kit.edu (P. Kaiser), Sandor.Szedmak@aalto.fi (S. Szedmak), Mustafa@mmmi.sdu.dk (W. Mustafa), Kraft@mmmi.sdu.dk (D. Kraft), Norbert@mmmi.sdu.dk (N. Krüger), Justus.Piater@uibk.ac.at (J. Piater), Asfour@kit.edu (T. Asfour).

https://doi.org/10.1016/j.robot.2017.10.012
0921-8890/© 2017 Published by Elsevier B.V.

check if a plan was executed successfully and to solve encountered problems, e.g. replace missing objects. The main aspects and contributions of the manuscript are:

- We elaborate on a framework that allows the robot to understand natural language, generate symbolic representations of its sensorimotor experience, generate complex plans according to the current world state, and monitor plan execution. The framework is implemented within the robot development environment *ArmarX*,[1] see also [6]. The developed natural language understanding (NLU) pipeline is intended for a flexible multi-purpose human–robot communication. Given human utterances, it generates goals for a planner, symbolic descriptions of the world and human actions, and representations of human feedback. It grounds ambiguous NL constructions into the sensorimotor experience of the robot and supports complex linguistic phenomena, such as ambiguity, negation, anaphora, and quantification without requiring training data. The NLU component interacts with related components in a system architecture such as the robot's memory, a planner, a replacement manager.
- We address the mapping of sensorimotor data to symbolic representations required for linking the sensorimotor experience of the robot to NLU and symbolic planning and describe how a symbolic domain description is generated from the robot memory each time an NL utterance needs to be interpreted.
- We introduce the novel Replacement Manager (RM) component of the framework, which is responsible for finding possible locations of missing objects as well as replacing missing objects with suitable alternatives. The RM utilizes a variety of replacement strategies based on robot's previous experience, common-sense knowledge extracted from text corpora, visual object features, and human feedback.

Parts of the cognitive architecture presented in this manuscript rely on our previous work. The concept of structural bootstrapping is introduced in the European project Xperience and in [4]. The *ArmarX* framework is introduced in [6]. Our NLU pipeline and the mapping of sensorimotor data to symbolic representations are first presented in [8]. The common sense knowledge extraction from text is described in [9], while the first experiment on object replacement based on text-derived knowledge is described in [3]. Vision-based object replacement is in focus of [10]. The learning framework for computing probable replacements is outlined in [11,12]. The novelty of this manuscript lies in realization of the concept of structural bootstrapping within a control architecture of a humanoid robot and the demonstration of how this concept allows for grounded cognitive behaviors in a complex setting requiring human–robot communication and collaboration. More specifically, we introduce a novel component of the architecture — the Replacement Manager, which implements both previously developed and novel replacement strategies, see Section 5. We extend existing components of the architecture, such as NLU and plan execution and monitoring, with novel functionality allowing the components to interact with the RM, see Sections 4 and 6. For testing the framework, we design and conduct two novel experiments, see Section 7. We test the developed framework on the humanoid robot ARMAR-III [7] in a scenario requiring planning based on human–robot communication. Two experiments are described. First, we present a demonstration of the scenario execution on ARMAR-III in a kitchen environment. Second, we test how well the framework can be employed by untrained users. To do so, we ask the subjects to control the robot in a visual simulation environment by using natural language.

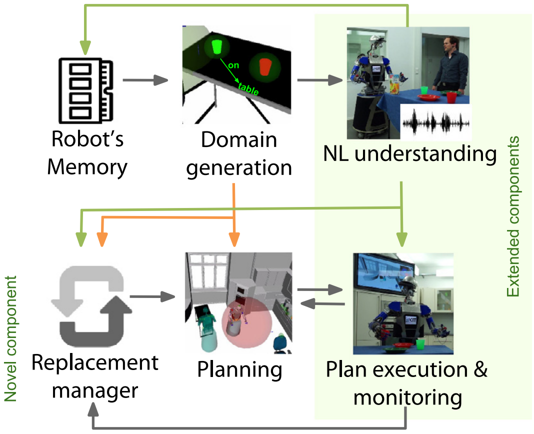


Fig. 1. The system architecture.

The remainder of the article is structured as follows. After presenting the general system architecture in Section 2, we present the domain description generation from the robot's memory (Section 3). Section 4 introduces the natural language understanding pipeline and Section 5 presents the Replacement Manager. Section 6 briefly presents the planner employed in the experiments and discusses how plan execution and monitoring are organized in our framework. Experiments on the humanoid robot ARMAR-III and in a visual simulation environment are presented in Section 7. Related work is discussed in Section 8, while Section 9 concludes the manuscript.

2. System architecture

The system architecture is implemented within the robot development environment *ArmarX* [6] and is shown in Fig. 1. The system architecture consists of six major building blocks: robot's memory, domain generation, natural language understanding, replacement manager, planning, as well as plan execution and monitoring.

The robot's memory is represented within *MemoryX*, one of the main modules of the *ArmarX* architecture (see Section 3). The memory stores and offers symbolic and sub-symbolic information about prior knowledge, long-term knowledge and knowledge about the current world state. Domain descriptions in symbolic form are generated from the robot's memory (see Section 3). The domain descriptions are also used by the NL understanding component for grounding and generating the domain knowledge base, see Section 4. The developed multi-purpose NLU framework can distinguish between (a) direct commands that can be executed without planning (*Go to the table*), (b) plans requiring commands that are converted into planner goals, which are processed by a planner (*Set the table*), (c) descriptions of the world that are added to the robot's memory and used by the planner (*The cup is on the table*), and (d) human feedback (*I'm fine with it*). The goal generated by the NLU component is passed to the Replacement Manager that checks for each object in the goal if this object and its location are in the domain description (Section 5). If an object or its location are unknown, then the component suggests available alternatives. NLU also provides the Replacement Manager with context-based affordances for mentioned objects. After suggesting a replacement, the Replacement Manager waits for a human confirmation or rejection, which is provided by the NLU component. If a replacement is confirmed, then the goal is rewritten and passed to the planner together with the domain description. The planner takes a domain

[1] https://armarx.humanoids.kit.edu.

description and a goal as an input. It generates a plan, which is a sequence of grounded atomic actions (see Section 6.1). Plan execution is performed by the Plan Execution & Monitoring component, which also verifies if the plan is executed correctly (see Section 6.2). Each time an NL utterance is registered and processed by the NLU pipeline and the planner, the robot's memory is updated and the required actions are added to the task stack to be processed by the Plan Execution & Monitoring component. If the plan execution fails e.g. because of missing objects, the component queries the Replacement Manager for suitable replacements. Otherwise, the planner is called to re-plan according to the current world state. Human comments and feedback can be used to update the world state in the robot's memory before or during action execution and are considered by the robot to adjust its plan accordingly.

2.1. Integration into the robotic platform

Integration of existing algorithms in a complex robotic platform poses several challenges. First of all, algorithms, e.g. for object localization, that are used on robotic service platforms in a dynamic environment, rather than in a controlled lab environment, have to be highly robust and real-time capable. Another challenge concerns the interfaces between the algorithms and the robotic hardware.

In the robot development environment *ArmarX* employed for the proposed system, generic interfaces were designed in an Interface Definition Language (IDL) to allow seamless exchange of implementations of different algorithms. Communication between different programming languages is realized with the middleware Internet Communication Engine [13], which provides transparent network communication and interfacing between many different programming languages with minimal overhead. New components can be easily added to the system with the minimal effort of implementing a corresponding interface. For example, new replacement strategies can by added to the Replacement Manager dynamically and will be used in future calls of the Replacement Manager. More details on *ArmarX* can be found in [6].

3. Robot's memory and domain description generation

The goal of the domain generation component is to map sensory data to symbolic representations. Since each symbol depends on a different combination of sensory data, we design the mapping procedure in a modular and extensible way. First, sensorimotor experience is turned into continuous sub-symbolic representations (e.g., coordinates of objects, the robot, and robot's hands) that are added to the robot's memory. These continuous representations are mapped to object and location names. Finally, symbolic representations describing the world state that are represented by predicates are generated.

3.1. Memory structure

MemoryX, a central part of the robot development environment *ArmarX*, is responsible for storing and representing different types of robot knowledge in different memories: prior knowledge memory, long-term memory, and working memory that provide symbolic entities like actions, objects, states, and locations. Each memory element, called *memory entity*, is represented by a name-value map. Memory entities are organized as a hierarchy, where every entity can be a parent of another entity. For example, the type `cup` can have a parent `container`, which has a parent `object`. If a parent has a feature attached, this feature is also available for its children. This hierarchy is also used as an ontology to define meta classes and specify common affordances or attributes of objects. For example, in order to specify that objects from a certain class are graspable, we add the class `graspable` as a parent for this class.

The *prior knowledge* contains persistent data inserted by the developer, e.g. accurate object 3D models, environmental models, and affordances as object-action associations extracted, for example, from text (see Section 5). The *long-term memory* consists of knowledge stored persistently, e.g. common object locations that are learned from the robot's experience during task execution and persistently stored as heat maps [14]. The *working memory* contains volatile knowledge about the current world state, e.g. object existence and position or relations between entities. The working memory serves as an intermediate data storage between sensorimotor experience and the symbolic high-level representation. The working memory is updated by components like the robot self-localization, object localization, or natural language understanding, whenever they receive new information.

To deal with uncertainties in sensory data, each memory entity value is accompanied by a probability distribution. In case of object locations, new data is fused with the data stored in the memory using a Kalman-filter.

3.2. Mapping sensorimotor data to symbols

In order to map sensorimotor experience to symbols, in this study, we employ the raw sensory data used by the components of self-localization, visual object recognition, and robot kinematics. For the self-localization, we use laser scanners and a 2D map of the environment. The self-localization is used to navigate on a labeled 2D graph, in which location labels are defined by a 2D pose of the location and a variance of the pose. For the visual object recognition, we use the RGB stereo vision with the texture-based [15] or color-based [16] approaches. The robot kinematics is used to calculate position of the robot's hands, which is used to determine if an object is grasped. Since the robot does not have sensors in its hands, we assume that the object is grasped if the pose of the object in the working memory is close enough to the robot hand pose.

The mapping of continuous sensory data into discrete symbolic data is done by *predicate providers*. Each world state predicate is defined in its own predicate provider module, which outputs a predicate state (unknown, true, false) by evaluating the content of the working memory or low-level sensorimotor data. Examples of the predicate providers are: `grasped` represents an object being held by an agent using a hand; `handEmpty` represents a state of a robot's hand; `objectAt` and `agentAt` represent object and robot locations, correspondingly; `leftgraspable` and `right-graspable` represent the fact that an object at a certain location can be grasped by the corresponding hand of the robot. Table 1 shows the full list of predicates that are calculated based on the sensor data.

Predicate providers can access other components (e.g., the working memory, robot kinematics information, long-term memory) to evaluate the predicate state. For example, the `objectAt` predicate provider uses the distance between the detected object coordinates and the center coordinates of the location label to determine if an object should be considered to be currently *at* this location. Only those objects that are required for fulfilling a particular task are tracked during the action execution. Higher level components operating on a symbolic level generate requests for a particular object to be recognized at a particular location. Other objects are not tracked to reduce the system load and avoid the false positive object recognition.

The validity of the generated predicates relies both on the sensor data and the memory state. Let us consider the `grasped` predicate as an example. The employed robot ARMAR-III does not have any sensor in his hands. Therefore, the `grasped` predicate

Table 1
Predicates calculated from sensor data.

Predicate with types	Description	Calculation description
inHand(object, hand, robot)	*object* is in *hand* of *robot*	Distance between object and hand $< \epsilon_1$
objectAt(object, location)	*object* is at landmark *location*	Distance between object and location $< \epsilon_2$
agentAt(robot, location)	*robot* is at landmark *location*	Distance between robot and location $< \epsilon_3$
handEmpty(robot, hand)	*hand* of *robot* is empty	Distance of hand to all objects $> \epsilon_4$
grasped(robot, hand, object)	*object* is grasped with *hand* of *robot*	Distance of hand to object $< \epsilon_5$
leftGraspable(object)	Left hand grasp is known for *object* and it is currently at a suitable location for the left hand	Object position in bounding box
rightGraspable(object)	Right hand grasp is known for *object* and it is currently at a suitable location for the right hand	Object position in bounding box

Table 2
Non-observable predicates.

Predicate with types	Description
open(door)	*door* is open
clean(location)	cleaning action was performed at *location*
stirred(container)	stirring action was performed in *container*
substanceIn(substance,container)	*substance* was poured into *container*
stackable(location)	more than one object can be located at *location* (e.g. sink)
toPutAway(object)	*object* needs to be put away into a predefined location (e.g. dirty utensils go into the sink)
inHandOfHuman(object,human)	*object* was handed to *human*

is defined by the distance between the object and the hand. The localization algorithms we apply are not precise enough to localize the grasped object in the hand, which makes it difficult to determine whether the object is grasped or not. We solve this problem by introducing the virtual attachment of the grasped object to the hand. By default, the grasping action is expected to be successful. Thus, it is assumed that the grasped object is virtually attached to the hand, i.e. it is moving synchronously with the hand with an increasing position uncertainty, until a new localization data is available. This strategy can fail if the grasping action was not successful. In this case, the robot keeps assuming that the object is in the hand, until the object is visually localized as still being at the original location.

Additionally, predicates that cannot be perceived by the robot, called non-observable predicates, are tracked and stored in the robot's memory. These predicates are either inserted by the human via speech or are inferred from the actions of the robot. An example of a non-observable predicate is *open* applied to a door. Currently, ARMAR-III cannot perceive a door being open or closed. However, it needs to access the state of the door in order to plan, for example, for grasping an object from the fridge. By default, the robot assumes all doors being closed. After it has performed the door opening action, it adds the predicate *open* applied to the door into its memory. The human can also update the memory of the robot by saying "The door of the fridge is open/closed". In the experiments described in Section 7, we use non-observable predicates that are listed in Table 2.

3.3. Domain description generation

Fig. 2 shows how the robot's memory is used to generate a symbolic domain description consisting of static symbol definitions and problem specific definitions. The symbol definitions consist of types, constants, predicate definitions, and action descriptions, while the problem definitions consist of the symbolic representation of the current world state represented by predicates and the goal state that should be achieved. Types enumerate available agents, hands, locations, and object classes stored in the prior knowledge. Constants represent available instances, on which actions can be performed, and are generated using entities in the working memory. Each constant can have multiple types, such that one is the actual type of the corresponding entity, and others are parents of that particular type including transitive parentship. For example, instances of the type `cup` are also instances of `graspable` and `object`. This type hierarchy is important for specifying actions over a particular set of types, e.g., the grasping action has the type `graspable` as a parameter to ensure that grasps are only planned on graspable objects. The domain generator derives action representations from the long-term memory, where they are associated with specific robot skills represented by statecharts as described in [17]. Each action is associated with a set of preconditions and effects represented by predicates.

The generated domain description is used by the NLU component as well as by the Replacement Manager and the planning component. The NLU component uses the domain description to create a knowledge base and to ground NL references and the Replacement Manager uses the description to check if objects in the planner goal and their locations are known to the robot. The planning component uses it as the knowledge base for plan generation.

4. Multi-purpose natural language understanding

The purposes of the NL understanding component are (a) to ground NL utterances to actions, objects, and locations stored in the robot's memory, (b) to distinguish between commands, descriptions of the world, and human feedback, (c) to provide context-based affordances for mentioned objects, and (d) to generate representations of each type of the NL input suitable for the downstream components (goal for the planner, object context for the Replacement Manager, feedback for the plan execution and monitoring component). Our approach is based on the abductive inference used for interpreting NL utterances as observations by linking them to known or assumed facts, see [18].

The NL understanding pipeline shown in Fig. 3 consists of the following processing modules. The speech input is processed by a speech recognition component[2] that converts it into text. The text is then processed by a semantic parser that outputs a logical representation of it. This representation together with observations stored in the robot's memory and the lexical and domain knowledge base constitute an input for an abductive reasoning engine that produces a mapping to the domain, i.e. symbolic labels known to the robot. The mapping is further classified and post-processed.

[2] In the experiments described in this manuscript, we used the speech recognition system presented in [19].

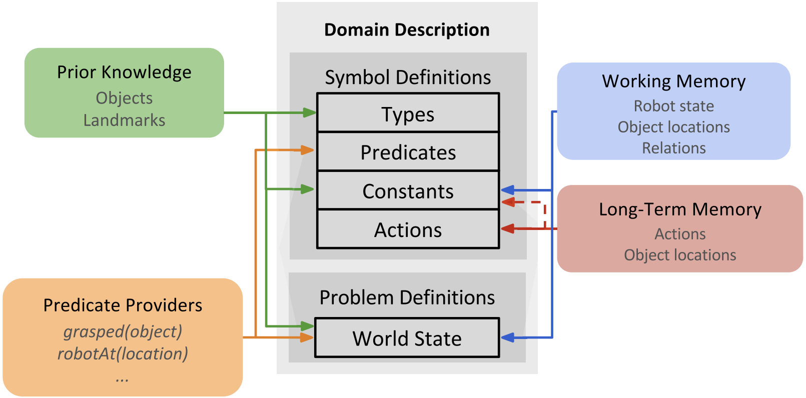


Fig. 2. Components involved in the domain description generation.

The pipeline is flexible, i.e. each component can be replaced by an alternative. We use the implementation of the abduction-based NLU that was developed in the context of knowledge-intensive large-scale text interpretation [20].

4.1. Logical form

We use logical representations of NL utterances described in [21]. In this framework, a *logical form* (LF) is a conjunction of propositions and variable inequalities, which have argument links showing relationships among phrase constituents. For example, the following LF corresponds to the command *Bring me the juice from the table*:

$\exists e_1, x_1, x_2, x_3, x_4$ (*bring-v* (e_1, x_1, x_2, x_3) $\wedge$ *thing* (x_1) $\wedge$ *person* (x_2) $\wedge$ *juice-n* (x_3) $\wedge$ *table-n* (x_4) $\wedge$ *from-p* (e_1, x_4)), where variables x_i refer to objects *thing*, *person*, *juice*, and *table* and variable e_1 refers to the eventuality of x_1 bringing x_2 to x_3; see [21] for more details. In the experiments described below, we used the *Boxer* semantic parser [22]. Alternatively, any dependency parser can be used if it is accompanied by an LF converter as described in [23].

4.2. Abductive inference

Abduction is inference to the best explanation. Formally, logical abduction is defined as follows:

Given: Background knowledge B, observations O, where both B and O are sets of first-order logical formulas,

Find: A hypothesis H such that $H \cup B \models O$, $H \cup B \not\models \perp$, where H is a set of first-order logical formulas.

Abduction can be applied to discourse interpretation [18]. In this framework, logical forms of the NL utterances represent observations, which need to be explained by the background knowledge. Where the reasoner is trying to link parts of the logical form to what is already known from the overall context and the background knowledge. The reasoner introduces assumptions, if it is provided with incomplete information. The reasoner prefers minimal hypotheses to those that introduce more assumptions.

Suppose the command *Bring me the juice from the table* is turned into an observation o_c. If the robot's memory contains an observation of a particular instance of `juice` being located on the table, this observation will be concatenated with o_c and the noun phrase *the juice* will be grounded to this instance by the abductive reasoner.

Another example is the disambiguation between a command (*Put the juice on the table*) and a world description (*The juice is on the table*), which depends on the presence of the action predicate.

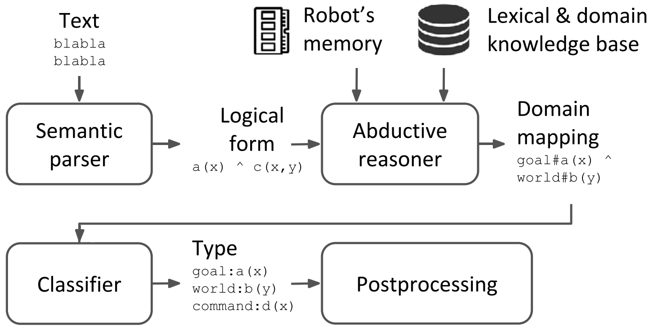


Fig. 3. Natural language understanding pipeline.

This disambiguation can be performed by using the following two background axioms:

$$goal\#objectAt\ (e_1, x_1, x_2) \rightarrow put\text{-}v\ (e_1, Robot, x_1) \wedge on\text{-}p\ (e_1, x_2)$$
$$world\#objectAt\ (x_1, x_2) \rightarrow on\text{-}p\ (x_1, x_2),$$

where prefixes *goal#* and *world#* indicate the type of information conveyed by the corresponding linguistic structures. In the case of the command, the first axiom will be applied, because it will explain more atomic observables (*put* and *on*). This axiom represents the fact that commands like *Robot, put* x_1 *on* x_2 imply that there is a goal of x_1 being located at x_2. In the case of the world description represented by the bare *on* prepositional phrase, the second axiom will be applied. This axiom describes the semantics of the bare *on* prepositional phrase not attached to a verb and represents the fact that x_1 is located at x_2.

We use a tractable implementation of abduction based on Integer Linear Programming (ILP) [20]. The reasoning system converts a problem of abduction into an ILP problem, and solves the problem by using efficient techniques developed by the ILP research community. Typically, there exist many hypotheses explaining an observation. In the experiments described below, we use the framework of *weighted abduction* [18] to rank hypotheses according to plausibility and select the best hypothesis. This framework allows us to define assumption costs and axiom weights that are used to estimate the overall cost of the hypotheses and rank them. As the result, the framework favors minimal (shortest) hypotheses as well as hypotheses that link parts of observations together and support discourse coherence, which is crucial for language understanding, see [24]. However, any other abductive framework and reasoning engine can be integrated into the pipeline.

4.3. Lexical and domain knowledge base

In our framework, the background knowledge B is a set of first-order logic formulas of the form

$$P_1^{w_1} \wedge \cdots \wedge P_n^{w_n} \rightarrow Q_1 \wedge \cdots \wedge Q_m,$$

where P_i and Q_j are predicate-argument structures or variable inequalities and w_i are axiom weights.[3]

Lexical knowledge used in the experiments described below was generated automatically from the lexical-semantic resources WordNet [25] and FrameNet [26]. First, verbs and nouns were mapped to the synonym classes. For example, the following axiom maps the verb *bring* to the class of giving:

$$\textit{action\#give}\,(e_1, \textit{agent}, \textit{recipient}, \textit{theme}) \rightarrow$$
$$\textit{bring-v}\,(e_1, \textit{agent}, \textit{theme}) \wedge \textit{to-p}\,(e_1, \textit{recipient})$$

Prepositional phrases were mapped to source, destination, location, instrument, etc., predicates. Different syntactic realizations of each predicate for each verb (e.g., *from X*, *in X*, *out of X*) were derived from syntactic patterns specified in FrameNet that were linked to the corresponding FrameNet roles. See [27] for more details on the generation of lexical axioms. A simple spatial axiom was added to reason about locations, which states that if an object is located at a part of a location (*corner*, *top*, *side*, etc.), then it is located at the location.

The synonym classes were further manually axiomatized in terms of domain types, predicates, constants, and actions. For example, the axiom below is used to process constructions like *bring me X from Y*:

$$\textit{goal\#inHandOfHuman}\,(e_1, \textit{theme}) \wedge$$
$$\textit{world\#objectAt}\,(\textit{theme}, \textit{loc}) \rightarrow$$
$$\textit{action\#give}\,(e_1, \textit{Robot}, \textit{recipient}, \textit{theme}) \wedge$$
$$\textit{location\#source}\,(e_1, \textit{loc}),$$

which represents the fact that the command evokes the goal of the given object being in the hand of the human and the indicated source is used to describe the location of the object in the world. The prefixes (e.g., *goal#*, *world#*) indicate the type of information conveyed by the corresponding linguistic structures. The framework can also handle numerals, negation, quantifiers represented by separate predicates in the axioms (e.g., *not*, *repeat*). For example, the following axiom is used to process constructions like *put N Xs on Y*:

$$\textit{goal\#objectAt}\,(e_1, \textit{theme}, \textit{loc}) \wedge \textit{\#repeat}\,(\textit{theme}, n) \rightarrow$$
$$\textit{action\#puton}\,(e_1, \textit{Robot}, \textit{theme}, \textit{loc}) \wedge \textit{card}\,(\textit{theme}, n)$$

The *#repeat* predicate is further used by the post-processing component that multiplies predicates containing the corresponding variable (*theme*) *n* times.

Negation is represented by the predicate *not*, e.g., the following axiom maps the adjective *dirty* to the domain:

$$\textit{not}\,(e_1) \wedge \textit{world\#clean}\,(e_1, x) \rightarrow \textit{dirty-a}\,(e_2, x)$$

Quantification is also represented by a separate predicate. The repetition, negation, and quantification predicates are further treated by the post-processing component.

The hierarchy axioms (`red_cup`→`cup`) and inconsistency axioms (`red_cup`*xor* `green_cup`) were generated automatically from the domain description. Every type-parent relation in the description was converted into a hierarchy axiom. If two types share the same parent and do not share any instances, they were declared to be inconsistent in the knowledge base.

4.4. Object grounding

If objects are described uniquely, then they can be directly mapped to the constants in the domain. For example, *the red cup* in the utterance *Give me the red cup* can be mapped to the constant `red_cup` if there is only one red cup in the domain. However, redundant information that can be recovered from the context is often omitted in the NL communication, see [28]. In our approach, grounding of underspecified references is naturally performed by the abductive reasoner interpreting observations by linking their parts together, see Section 4.2. For example, given the text fragment *The red cup is on the table. Give it to me*, the pronoun *it* in the second sentence will be linked to *red cup* in the first sentence and grounded to `red_cup`. To link underspecified references to earlier object mentions in a robot–human interaction session, we keep all mentions and concatenate them with each new input LF to be interpreted. Predicates describing the world from the robot's memory are also concatenated with LFs to enable grounding. Given *Bring me the cup from the table*, the reference *the cup from the table* will be grounded to an instance of `cup` observed as being located on an instance of `table`.

If some arguments of an action remain underspecified or not specified, then the first instance or the corresponding type will be derived from the domain description. For example, the execution of the action of putting things down requires a hand to be specified. In the NL commands this argument is often omitted (*Put the cup on the table*), because for humans it does not matter, which hand the robot will use. The structure `putdown(cup,table,hand)` is generated by the NLU pipeline for the first command above. The grounding function then selects the first available instance of the underspecified predicate. In future, we consider using a clarification dialog, as proposed, for example, in [29].

4.5. Context-based affordances

Object replacement depends on the corresponding object affordances. For example, a spoon can be replaced by a fork in the context of eating or by a knife in the context of stirring. Potential affordances can be extracted from the dialog context. For example, if the human says *I'd like to drink something. Bring me some juice*, then the affordance of the juice is drinking. In order to provide the Replacement Manager with this information, we store all verbs mentioned in the dialog session. When the NLU component generates a new planner goal, for each domain object label represented by a noun phrase, it selects a verb possibly representing its affordance. Those verbs are selected, which have the highest weights in the common-sense affordance database generated from corpora as described in Section 5.1.1. For object labels, for which there is no appropriate verb in the dialog context, the top affordance from the common-sense affordance database is provided. Basing the context-based affordance extraction on verbs is a clear limitation, because any part of speech can refer to an affordance, e.g., *I'm thirsty*. *Bring me some juice*. In future, we plan to employ lexical-semantic and world knowledge to reason about affordances.

4.6. Classifier

The classifier takes into account prefixes assigned to the inferred predicates. For example, the abductive reasoner returned the following mapping for the command *Bring me the cup from the table*:

$$\textit{action\#give}\,(e_1, x_1, x_2, x_3) \wedge \textit{location\#source}\,(e_1, x_4) \wedge x_1 = \textit{Robot} \wedge x_2 = \textit{Human} \wedge \textit{goal\#inHandOfHuman}\,(e_1, x_3, x_2) \wedge \textit{world\#objectAt}\,(x_3, x_4)$$

[3] See [23] for a discussion of the weights.

The classifier extracts predicates with prefixes and predicates related to the corresponding arguments. The following structures will be produced for the mapping above:

`[goal: inHandOfHuman(cup,Human), world: objectAt(cup,table)]`

Actions that do not evoke goals or world descriptions are interpreted by the classifier as direct commands or human action descriptions depending on the agent. For example, *action#grasp* (*Human*, *cup*) (*I'm grasping the cup*) will be interpreted as a human action description, while *action#grasp* (*Robot*, *cup*) (*Grasp the cup*) is a direct command.

The classifier can also handle nested predicates. For example, the utterances 1) *Help me to move the table*, 2) *I will help you to move the table*, 3) *I will help you by moving the table* will be assigned the following structures, correspondingly:

1. `[direct_command: helpRequest:[requester: Human, action: move(Robot,table)]`
2. `[human_action: help:[helpInAction: move (Robot,table)]`
3. `[human_action: help:[helpByAction: move (Human,table)]`

The human feedback is currently typed as `agreement` (e.g., *I'm fine with it*), `disagreement` (e.g., *No*), or `no_information` (e.g., *I don't know*).

4.7. Post-processing

The post-processing component converts the extracted data into the format required by the downstream modules. Direct commands and human feedback are immediately processed by the Plan Execution & Monitoring component. Object context is used by the Replacement Manager. World descriptions are added to the robot's working memory. Goals extracted from utterances are converted into a planner goal format, so that *not* predicate is turned into the corresponding negation symbol, predicates that need multiplication (indicated by the *#repeat* predicate) are multiplied, and quantification predicates are turned into quantifiers. For example, the commands 1) *Put two cups on the table* and 2) *Put all cups on the table* can be converted into the following goal representations in the PKS syntax [30], correspondingly:

1. `(existsK(?x1: cup, ?x2: table) K(objectAt (?x1,?x2)) & (existsK(?x3: cup) K(objectAt (?x3,?x2)) & K(?x1 != ?x3)))`
2. `(forallK(?x1: cup) (existsK(?x2: table) K(objectAt(?x1,?x2))))`

4.8. Processing examples

In the following, we present processing steps for two example sentences. For simplicity, we do not demonstrate object grounding in these examples. As described in Section 4.4, the grounding is performed by concatenating logical forms produced by the semantic parser with the earlier object mentions as well as predicates corresponding to objects known to the robot.

In the example below, two background axioms are employed by the abductive reasoner. Lexical axiom L1 is derived from FrameNet and maps the phrase *bring from* to the giving action class and location indication. Domain axiom D1 created manually is used to map the giving action class to the goal of the object being in the hand of a human.

Analysis step	**Output representation**
Text	*Bring me the juice from the table*
Logical form	∃e_1, x_1, x_2, x_3, x_4 (*bring-v* (e_1, x_1, x_2, x_3) ∧*thing* (x_1) ∧ *human* (x_2) ∧*juice-n* (x_3) ∧*table-n* (x_4) ∧*from-p* (e_1, x_4))
Axioms applied	L1: *action#give* (e_1, *agent*, *recipient*, *theme*) ∧ *world#objectAt* (*theme*, *location*) → *bring-v* (e_1, *agent*, *recipient*, *theme*) ∧*from-p* (e_1, *location*) D1: *goal#inHandOfHuman* (e_1, *theme*, *recipient*) → *action#give* (e_1, *robot*, *recipient*, *theme*)
Abductive inference	*goal#inHandOfHuman* (e_1, x_3, x_2) ∧*world#objectAt* (x_3, x_4) ∧*juice-n* (x_3) ∧*table-n* (x_4)∧*human* (x_2)
Classifier	`[goal: inHandOfHuman(juice,human), world: objectAt(juice,table)]`
Post-processor	`[goal: (existsK(?x1 : juice, ?x2: human) K(inHandOfHuman(?x1,?x2)), world: objectAt(juice,table)]`

The next example illustrates the use of the *#repreat* predicate used to multiply goad predicates.

Analysis step	**Output representation**
Text	*Put two the glasses on the table*
Logical form	∃e_1, x_1, x_2, x_3 (*put-v* (e_1, x_1, x_2) ∧*thing* (x_1) ∧ ∧*glass-n* (x_2) ∧*card* (x_2, 2) ∧ *table-n* (x_3) ∧*on-p* (e_1, x_3))
Axioms applied	L1: *action#puton* (e_1, *agent*, *theme*, *location*)→ *put-v* (e_1, *agent*, *theme*) ∧*on-p* (e_1, *location*) D1: *goal#objectAt* (e_1, *theme*, *location*) ∧*#repeat* (*theme*, *n*) → *action#puton* (e_1, *robot*, *theme*, *location*) ∧*card* (*theme*, *n*)
Abductive inference	*goal#objectAt* (e_1, x_2, x_3) ∧*#repeat* (x_2, 2) ∧ *glass-n* (x_2) ∧*table-n* (x_3)
Classifier	`[goal: objectAt(glass, table) & repeat (glass,2)]`
Post-processor	`[goal: (existsK(?x1 : glass, ?x2 : table) K(objectAt(?x1, ?x2)) & (existsK(?x3 : glass) K(objectAt(?x3, ?x2)) & K(?x1 != ?x3)))]`

5. Replacement manager

The Replacement Manager (RM) has two aims: 1) to replace missing objects with alternatives and 2) to suggest new potential locations for missing objects. A welcome side product of the RM is that it serves as a preliminary feasibility checker for the task before the planning process is started. If objects mentioned in the goal are not present in the generated domain, the planner will not be able to find a valid plan and thus does not need to be called.

The RM is evoked after the NLU component produces a goal for the planner or if the plan execution fails because of a missing object (see Section 6.2). For each object and location name occurring in the goal, the RM checks if they are contained in the robot's memory (*MemoryX*), i.e. the names can be converted into *MemoryX* types that have instances with specified locations. If an instance or its location is missing, the RM attempts a replacement and rewrites the goal before passing it to the planner.

Fig. 4 shows how the RM interacts with other components. A speech command is processed by the NL Understanding component that generates a goal for the planner and affordances for each object mentioned in the goal. The RM queries the domain generator and replaces unknown objects in the goal with the known ones and makes sure that there are valid locations for instances of all objects mentioned in the goal by inserting object instance hypotheses into the working memory. The planner will treat these the same as confirmed object instances, but all actions using an object instance will fail during execution if the object instance hypothesis is wrong.

If a suitable replacement has been found, the RM rewrites the goal. The component passes the goal to the planner that generates a plan. The plan execution is supervised by the Plan Execution

& Monitoring component. If the plan execution fails because of a missing object, the RM is called again.

The RM is using different replacement strategies, which vary with respect to the considered input data and range from visual shape estimation to evaluation of large text corpora. The replacement strategies are subdivided into object and location replacement strategies as follows.

5.1. *Object replacement*

Object replacement is performed when (a) an object type mentioned in the goal is unknown or (b) a suitable object could not be found at any known location of the object during the plan execution. We employ two object replacement strategies based on shared common-sense affordances and shared visual features as described below. Object replacement requires human feedback. Therefore the RM generates a confirmation question for the human and proceeds with the replacement only if it was confirmed.

5.1.1. *Common-sense affordances strategy*

The strategy based on shared common-sense affordances employs typical object affordances generated from textual corpora as described in [9]. For each noun referring to an object in the domain, we extract affordances expressed by verbs with assigned scores. This is done as follows. We use a parsed text corpus[4] to extract verbs co-occurring with a given noun in the instrument role patterns: "`VERB with (a/the)? NOUN`" (*cut with a knife*), "`NOUN for VERBing`" (*knife for cutting*) and in the patient role patterns: "`VERB (a/the)? NOUN`" (*cut the bread*). Stop words are excluded from consideration. As a result, we generate an affordance database containing entries of the form ⟨ *object*, *affordance*, *role*, *norm_freq* ⟩, where *role* can be *instrument* or *patient* and *norm_freq* is a normalized frequency of the co-occurrence of *object* and *affordance* in the patterns, e.g., ⟨*juice*, *drink*, *patient*, 0.866⟩. The affordance database is used to generate a replacement database consisting of tuples of the form ⟨ *object1*, *object2*, *affordance*, *score* ⟩ indicating that *object1* can be replaced by *object2* towards *affordance* with the confidence equal to *score*. For example, a spoon is most likely to be replaced by a fork towards eating, while it is most likely to be replaced by a stirrer towards stirring.

The confidence *score* is computed by a relational learning framework in two steps. First, similarity for object pairs towards affordances is computed. Second, a function is learned, which generalizes the known relations to all possible object pairs not observed earlier. The similarity measure is defined as follows.

$$r(o_1, o_2|a) = \begin{cases} n\_f(o_1, a) & \\ \quad *n\_f(o_2, a) & \text{if } (o_1, a) \in \mathcal{D} \text{ and } (o_2, a) \in \mathcal{D} \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $\mathcal{D} \subset \mathcal{O} \times \mathcal{A}$ is a set of object-affordance pairs such that $\mathcal{O}$ is a set of objects and $\mathcal{A}$ is a set of affordances, $n\_f(o, a)$ is the normalized frequency of the object o and affordance a co-occurrence. Based on this similarity measure a feature vector for all object pairs can be constructed, $\phi(o_1, o_2) = (r(o_1, o_2|a), a \in \mathcal{A})$. Note that ϕ can be defined for cases not represented in $\mathcal{D}$.

At the second step, a set of functions connecting the object pairs to the affordances is learned. This step is needed to propose replacements for object o towards affordance a even if o and a do not co-occur in the affordance database. It is also needed to learn replacements towards an unknown affordance. In [3,11,12], the learning procedure and related applications are described in detail. Let us sketch the main idea in this section.

In this approach, the similarity measure $r(o_1, o_2|a)$ is represented by probability density functions of Gaussian distribution with expected value $r(o_1, o_2|a)$ and with a common standard deviation σ, i.e. $\psi(r(o_1, o_2|a)) = p(.|r(o_1, o_2|a), \sigma)$. The functions are defined for all affordances $a \in \mathcal{A}$ as $f_a : \mathcal{O} \times \mathcal{O} \rightarrow \mathcal{F}_\sigma$, where $\mathcal{F}_\sigma$ is the set of all probability functions of normal distribution with standard deviation σ. These functions can express the uncertainty of the similarity measure. The functions $\{f_a | a \in \mathcal{A}\}$ are then represented by a linear operator that maps feature vectors $\phi(o_1, o_2,)$ of the object pairs into the space of $\mathcal{F}_\sigma$.

The learning problem consists in maximizing the inner product between the predicted and the given feature vectors of the similarity scores for all observed object pairs and affordances. This optimization problem is an extension of the Support Vector Machine, and the Maximum Margin Markov Networks developed for structured output learning frameworks, see a description and several alternatives in [31]. This optimization problem can be solved via its dual form, the detailed procedure is provided by [32]. After solving the optimization problem, the predicted similarity score for tuple $\langle o_1, o_2, a \rangle$ can be computed as follows

$$\psi(r(o_1, o_2|a)) = \mathbf{W}_a^* \phi(o_1, o_2), \tag{2}$$

where $\mathbf{W}_a^*$ is the optimal solution for affordance a. Since ψ as a feature vector is a probability density function, the replacement score measure for an object pair towards an affordance can be derived by taking the expectation with respect to ψ, i.e.

$$\tilde{r}(o_1, o_2|a) = E[\psi(r(o_1, o_2|a))]. \tag{3}$$

The NL Understanding component generates an affordance for each object mentioned in the goal as described in Section 4. For the object that needs to be replaced, the shared common-sense affordances strategy searches for the replacement towards the LU generated affordance for this object, which has the highest score in the replacement database. Since the replacement database is generated from textual corpora, it contains only object names represented by nouns ("cup" instead of "bluecup"). The strategy uses the type hierarchy (e.g., "bluecup" → "cup" → "container") for finding the replacement. For example, if "bluecup" needs to be replaced, it will search for replacement options specified for its parent in the hierarchy. Similarly, if "cup" is suggested as a replacement, the strategy will select its leaf child in the hierarchy as an actual replacement candidate.

5.1.2. *Visual features strategy*

This strategy compares precomputed affordances of the object to be replaced, which are based on its visual features, with affordances extracted from point cloud data during run-time in the robot's current field of view. In [10], we introduced, evaluated and discussed in more detail the strategy presented here. The affordances are estimated on the basis of object shapes. More specifically, we utilize the shape representation based on global 3D descriptors to predict functional properties of objects [33]. For instance, a container shape affords pouring into it, dropping into it, etc. Fig. 5 shows ARMAR-III applying the visual features strategy. The perceived point cloud data is projected into the memory view of the robot. For each dense point cloud cluster (colored point cloud data), the affordances based on the shape of the cluster are estimated. In this example, the bowl as well as the basked afford *pouring into*, *stirring*, and *dropping into*. The gray bowl represents the perceived pose of the bowl produced by the object recognition system.

Object shapes are described using histograms of relations between pairs of 3D features. First, we segment the scene using RGBD data obtained from the Kinect sensor. For each segment, we extract planar 3D surface features called *3D texlets* [34] that contain position and orientation. Objects are represented by sets of pairwise

[4] In the experiments described below, the Google Books corpus was used, https://storage.googleapis.com/books/syntactic-ngrams/index.html.

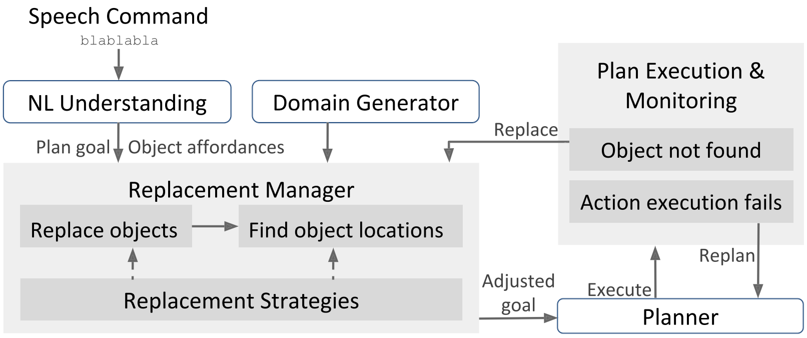


Fig. 4. Interaction between the replacement manager and other components.

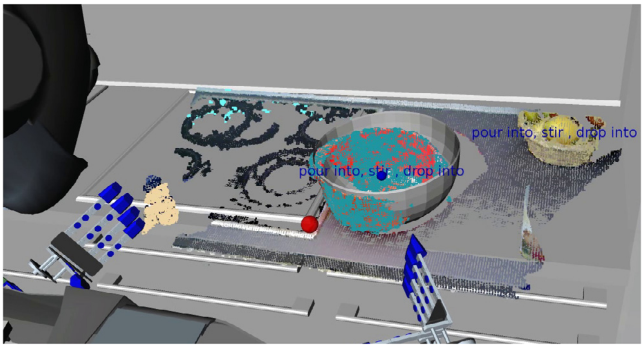


Fig. 5. Affordances predicated based on shape representations.

relations, defined globally, for all pairs of texlets. We compute geometric relations of two attributes: angle and scale-invariant distance (i.e., normalized relative to the object size) between 3D texlets. The distance relation is chosen to be scale-invariant, because what defines a object affordance is usually independent of scale. The final object descriptor is obtained by binning these two relations in a 2D histogram, which models the distributions of the relations in fixed-sized feature vectors while considering their co-occurrence. According to previous investigations [33], the binning size is set to 12 in both dimensions resulting in a feature vector of 144 dimensions. The resulting descriptor is highly discriminative, leading to fast learning and accurate estimation.

Affordance labels are learned using JointSVM [10]. JointSVM is equivalent to Structural SVM, which is an extension of SVM for predicting structured outputs, with a linear output kernel plus a regularization term on the kernel [35]. As input kernels, we choose polynomial kernels based on previous tests, cf. [10]. The estimation of both the kernel parameters and the internal parameters of JointSVM is performed by cross-validation.

JointSVM outputs an indicator vector, i.e. a set of binary labels, defined on the objects appearing on a scene. The learner treats the full output vector as one entity, i.e. simultaneously predicts all object labels. The estimation of the confidence of each affordance label is based on the assumption that if the predicted vector is close to a known label vector in the training data, then the confidence should be high, otherwise it is low; see [10] for the implementation.

For estimating if an unknown object with predicated affordances can replace a known object with known affordances, we compute object similarity. The similarity measure is defined as follows:

$$S(y(x)) = S_p(y(x)) - S_n(y(x)), \tag{4}$$

where $y(x)$ are the predicted affordances, and $S_p(y(x))$, $S_n(y(x))$ are the positive and the negative similarity metrics, respectively, cf. [10]. The positive similarity accounts for the true positive predictions, $y_p(x)$, while the negative similarity accounts for the false positive predictions, $y_n(x)$. Both $y_p(x)$ and $y_n(x)$ are derived from $y(x)$ by considering the known affordances. Then, $S_p(y(x))$ and $S_n(y(x))$ are defined as follows:

$$S_p(y(x)) = AVG(y_p(x)) \times TPR(y(x)) \tag{5}$$

$$S_n(y(x)) = AVG(y_n(x)) \times FPR(y(x)) \tag{6}$$

where *AVG* indicates the mean value and *TPR* and *FPR* indicate the true positive rate (recall) and the false positive rate (fall-out), respectively. Based on the pre-defined threshold, a potential replacement is estimated to be acceptable or not. In the experiments described below, we used the threshold of 0.3 estimated as described in [10].

Because it is possible to extract affordances during run-time, the robot can select an object as a replacement alternative, for which it otherwise does not have any knowledge. It can also react to changes in the known objects. For example, a container can be closed or open, which changes its affordances, e.g., it can be poured from or into only when it is open.

5.2. Location replacement

Location replacement happens when the location of all instances of an object type mentioned in the goal is unknown or when an object could not be found during the plan execution. For the location replacement, the RM manipulates the current working memory of the robot and inserts object instances as unconfirmed hypotheses at the suggested location from the replacement strategy. We employ three location replacement strategies based on 1) common locations learned from the previous experience of the robot, 2) common-sense locations obtained from textual corpora, 3) the human feedback.

5.2.1. Common locations strategy

The strategy based on common locations uses the feature of *ArmarX* that allows robots to learn typical locations of the objects from its experience [14]. Since the information about object locations is stored in the robot's memory as a set of density distributions of points (see Fig. 6), the distributions have to be mapped to symbolic location labels that can be processed by the planner. To do so, we link a location label with the expected value of the corresponding distribution. When the robot is close enough to the object and can actually see it, then the assumed location is replaced by the actual observed object position.

This strategy is the most used location strategy, since the confidence of the location hypotheses is high. When the robot is initialized, the locations of the objects are unknown. Thus, when a command is received, a location hypothesis needs to be generated for each implied object.

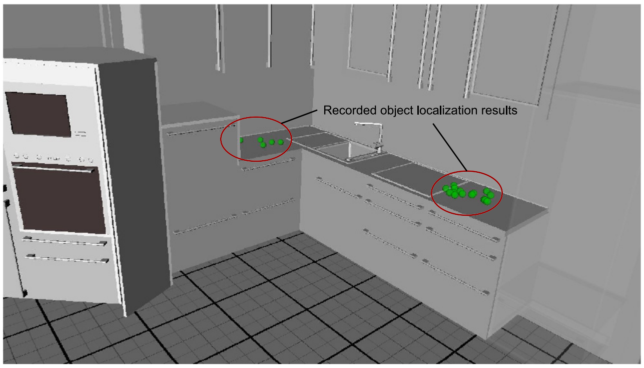


Fig. 6. Previously seen locations of an object (green spheres), which are clustered to density distributions in order to generate common object locations. The cluster on the right side represents the top location hypothesis, since the object has been seen there more frequently. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.2.2. *Common-sense locations strategy*

The strategy based on common-sense knowledge obtained from textual corpora employs the method for extracting typical object locations from text as described in [9]. For each pair of (*object_label*, *location_label*) in our domain, such that the labels are expressed by nouns, we search in the corpus for the patterns "`OBJECT_NOUN (be)? loc_prep LOC_NOUN`", where `loc_prep` is a location preposition, e.g., *on*, *in*, *at*. Using this method, we generate a location database consisting of the tuples of the form ⟨ *object*, *location*, *score* ⟩. Each tuple proposes a location for a given object, with a confidence score corresponding to the normalized frequency of their co-occurrence in the corpus. To increase the likelihood of finding objects, this strategy queries the database not only for the actual object type, but also all parents of that object type in the type hierarchy.

5.2.3. *Human feedback strategy*

The strategy based on the human feedback uses object locations communicated by the human, e.g., *The corn is in the fridge*. As described in Section 4, the NL Understanding component handles world state descriptions including location descriptions and updates the *MemoryX* working memory correspondingly. Thus, the strategy consists of generating a question for the human asking for a location of the missing object and monitoring the *MemoryX* working memory updates. After *MemoryX* was updated, the RM invokes the planner, which replans given the new information. This strategy is a fall-back mechanism that is used only if all other strategies fail.

6. Planning and plan execution & monitoring

6.1. *Planning*

We define a plan as a sequence of actions $P = \langle a_1, \ldots, a_n \rangle$ with respect to the initial state s_0 and the goal G such that $\langle s_0, P \rangle \models G$. In the experiments described in Section 7, we used the state-of-the-art *PKS* planner [30]. Any other symbolic planner can be used instead.

The domain generation and NLU components of our system provide the planner with a domain description (Section 3) and a goal (Section 4) represented in the *PKS* syntax as input. Below we show the PKS representation of the state of the world description corresponding to the robot being in the center of the kitchen and two cups being on the countertop.[5]

```
agentAt(robot, kitchen_center),
handEmpty(robot, rightHand), handEmpty(robot,
    leftHand),
objectAt(cup1, countertop), objectAt(cup2,
    countertop),
leftGraspable(cup1), rightGraspable(cup1),
leftGraspable(cup2), rightGraspable(cup2)
```

The example below shows the PKS definition of the `grasp` action.

```
grasp(?a : robot, ?h : hand, ?l : surface, ?o :
graspable) {
  preconds:
    (K(rightGraspable(?o)) & (existsK(?y :
        rightHand) K(?y = ?h)) |
    (K(leftGraspable(?o)) & (existsK(?y :
        leftHand) K(?y = ?h)))) &
    K(agentAt(?a, ?l)) &
    K(objectAt(?o, ?l)) &
    K(handEmpty(?a, ?h))
  effects:
    add(Kf, grasped(?a, ?h, ?o)),
    del(Kf, objectAt(?o, ?l)),
    del(Kf, handEmpty(?a, ?h))
}
```

The PKS planner returns sequences of grounded actions with their pre- and post-conditions. Given the domain description example above and the following goal: `(existsK(?x1 : cup, ?x2 : table) K(objectAt(?x1, ?x2)) & (existsK(?x3 : cup) K(objectAt(?x3, ?x2)) & K(?x1 != ?x3)))` corresponding to the command *Put two cups on the table*, it generates the plan below.[6]

```
move(robot, kitchen_center, countertop)
  pre:  agentAt(robot, kitchen_center)
        (kitchen_center != countertop)
  post: agentAt(robot, countertop)
grasp(robot, leftHand, countertop, cup1)
  pre: agentAt(robot, countertop)
       objectAt(cup1, countertop)
       handEmpty(leftHand)
       leftGraspable(cup1)
  post: grasped(robot, leftHand, cup1)
        !objectAt(cup1, countertop)
        !handEmpty(leftHand)
grasp(robot,rightHand,countertop, cup2)
  ...
move(robot,countertop,table)
  ...
putdown(robot,leftHand,table, cup1)
  pre:  agentAt(robot, table)
        grasped(robot, leftHand, cup1)
  post: handEmpty(robot, leftHand, cup1)
        objectAt(cup1, table)
        !grasped(robot, leftHand, cup1)
putdown(robot,rightHand,table, cup2)
  ...
```

6.2. *Plan execution and monitoring*

The components described in the previous sections are employed by the Plan Execution and Monitoring (PEM) component. The PEM is the central coordination component for the command execution. A simplified control flow for execution of a planning

[5] For simplicity, in this example we skip definitions of types, predicates, and constants required by *PKS*.

[6] The plan description is shortened for better readability.

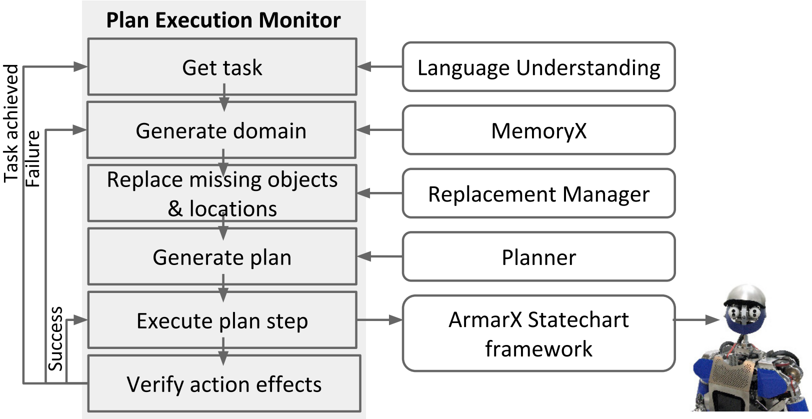


Fig. 7. Plan generation, execution, and monitoring.

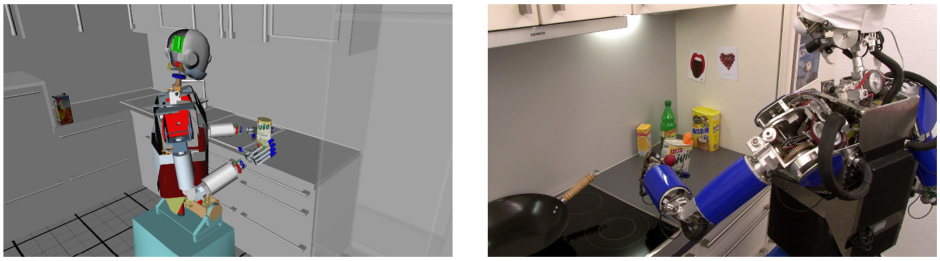


Fig. 8. Visualization of the robot's working memory (left) during action execution and the robot in the real world state (right) at the same moment. The working memory is continuously updated with perceived and predicted data. Only objects relevant for the current task or that have been relevant for a previous task are tracked in the working memory.

task is shown in Fig. 7. The PEM is evoked when a new task is sent from an external component (e.g., the NLU component). Different types of tasks are accepted. For each type of task, the PEM has an implementation of a *ControlMode* interface, which knows how to execute this task type. Currently, a task can be a single command or a list of goals that should be achieved. Here, we are focusing on the goal task type, which requires the planner to produce a plan. After a new task was received, the PEM calls the Domain Generator to generate a new domain description based on the current world state (Section 3). This domain description together with the received goal are then passed to the Replacement Manager, which checks if every object in the goal and its location are in the domain description. In case of missing objects or object locations, it replaces them and rewrites the goal. Then the domain description and the rewritten goal are passed to the planner. If a plan could not be found, the PEM synthesizes a feedback indicating the failure. A successful plan consists of a sequence of actions with bound variables as well as pre- and post-conditions of the actions that are passed back to PEM. These actions are executed one by one.

Each action that corresponds to a symbolic planning operator is associated with an *ArmarX* statechart, which controls the action execution. The *ArmarX* statecharts allow us to model robot actions in a hierarchical manner and to specify control and data flow visually. Due to the hierarchy of statecharts, it is possible to compose complex skills by using elementary or primitive skills, e.g. following a trajectory with the tool center point of a robot arm. The primitive skills are based on services provided by the robot development environment *ArmarX* [6] such as inverse kinematics, motion planning, and robot's memory. Each top-level action, i.e. the execution of a planning operator such as *grasp*, is manually designed with respect to the available sensors and algorithms. For example, the action *open* uses force–torque-based sensing for grasping the door handle and impedance control to open the door. The action *grasp* uses visual-servoing [36] for precise execution with respect to the object localization. The actions *stirring* and *wiping* use motion learned from demonstration with a specialization of the action formalism Dynamic Movement Primitives presented in [37]. A detailed description of the *ArmarX* statecharts can be found in [17].

Action execution might fail because of uncertainties in perception and execution or changes in the environment. To account for the changes, pre-conditions of an action before the execution and its effects after the execution are verified by the PEM. If action execution failed because of a missing object, then the PEM calls the Replacement Manager to replace the object or its location. The world state is continuously updated as well as the working memory based on sensor-data or prediction models. Fig. 8 shows the visualization of the robot working memory and the robot in the real world at the same moment during the execution of the skill `putdown`. The world state observer component is queried for the current world state after each action. If any mismatches between a planned world state and a perceived world state are detected, the plan execution is considered to have failed and re-planning is triggered based on the current world state. Additionally, the statecharts report if they succeeded or failed; failing leads to re-planning. If an action was successfully executed, the next action is selected and executed. After the task completion the robot goes idle and waits for the next task.

7. Experiments

In this section, we describe two experiments: a demonstration of the humanoid robot ARMAR-III and a visual simulation experiment involving untrained human subjects. Several specific components involved in our system have been evaluated in our previous work. The accuracy of the NLU pipeline was evaluated in [8]. Common-sense knowledge extraction was evaluated in [9]. In [3], object replacement based on common-sense knowledge was

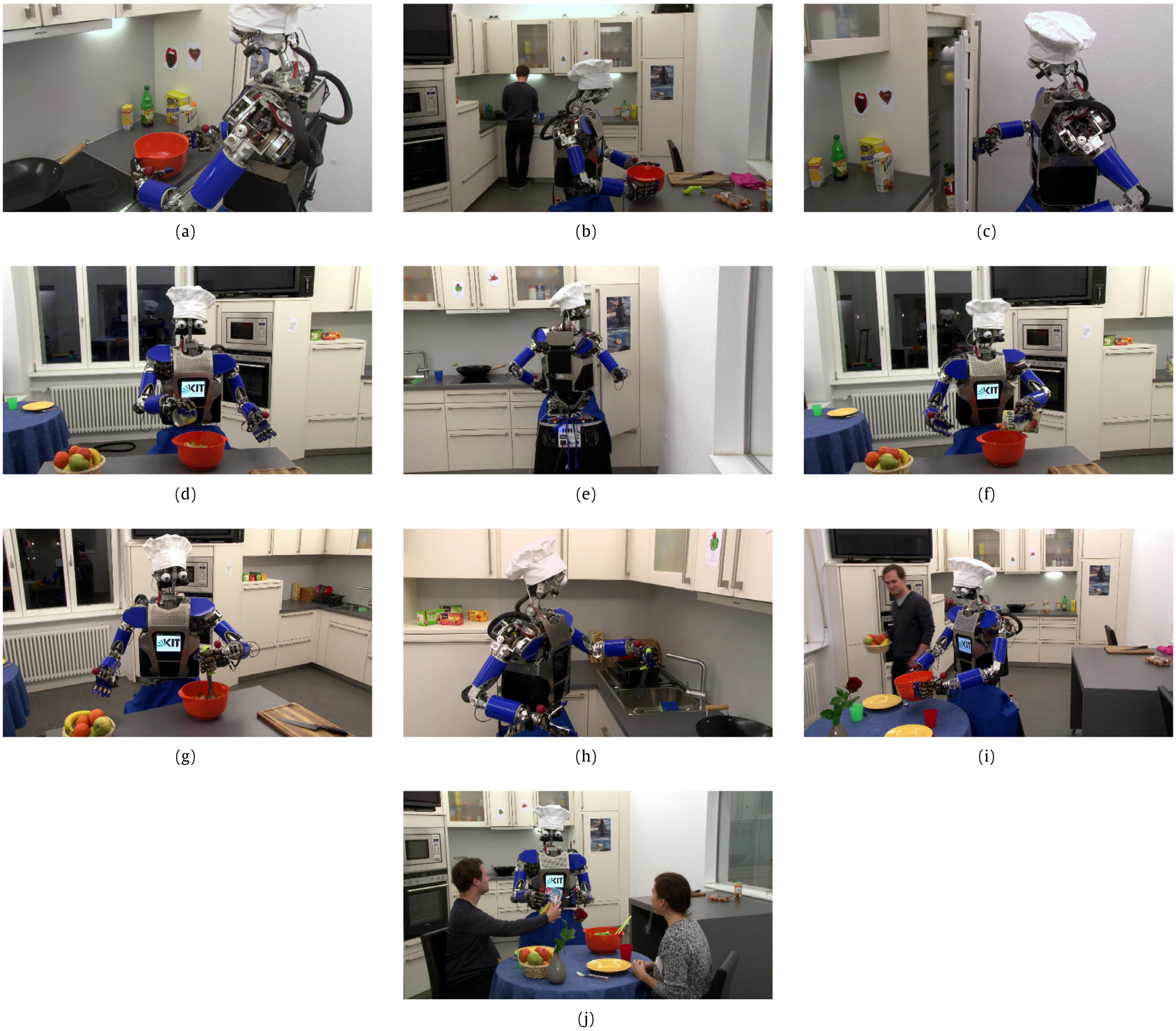

(a) (b) (c) (d) (e) (f) (g) (h) (i) (j)

Fig. 9. Snapshots of cooperative rearranging the room and preparing a dinner.

tested. In [10], an evaluation of the visual features replacement strategy was presented. The object localization was evaluated in [15,16]. In this study, we test how all components work in combination in a complex setting requiring human–object communication and collaboration.

7.1. Execution on ARMAR-III

We tested our approach on the humanoid robot ARMAR-III [7] in a kitchen environment. The case study elaborates on the dinner preparation scenario. In this section, we describe two parts of the *Xperience* project demo scenario, which are relevant for this manuscript: 1) salad preparation, 2) bringing a drink. The accompanying video can be found at https://youtu.be/PyJ5hCW3zQM. The video of the full *Xperience* project demonstration can be found at https://youtu.be/-8oC-WW5P1I. Apart from the video, the system was also shown in a live demonstration for the project review of the *Xperience* project.

In this experiment, the following robot skills were involved: moving, grasping, placing, stirring, pouring, door opening and closing, handing, receiving. Fig. 9 shows snapshots of the scenario in chronological execution order.

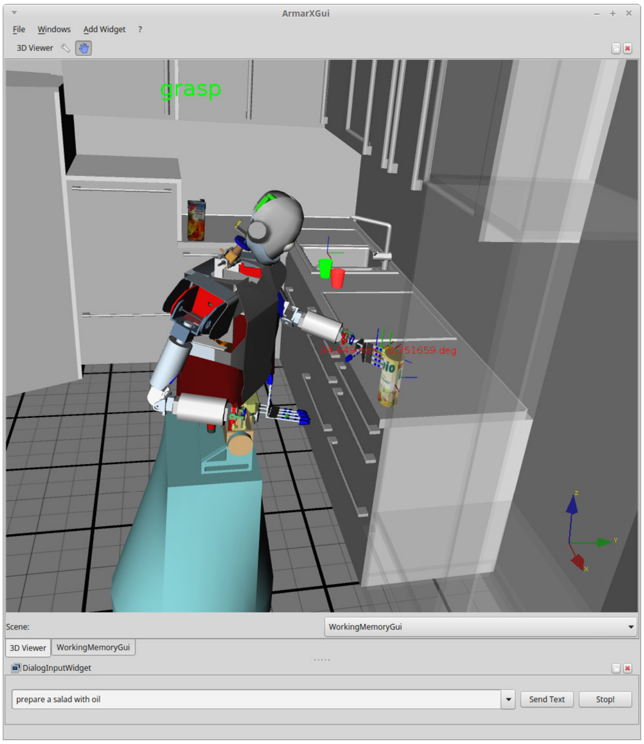
In the **salad preparation** part, the human first asks the robot to put a salad bowl on the sideboard. The command is processed by the NLU component, which generates a planner goal. The goal and the domain description are processed by the planner, which generates a multi-step plan. According to the plan, the robot moves to the location of the salad bowl, but does not find the required object at the location. The Plan Execution & Monitoring component reports the failure in the plan execution. The Replacement Manager is evoked. It finds a container that has a similar shape to the original known bowl, Fig. 9(a). The RM suggests the new container as a replacement based on shared visual features. The goal is rewritten accordingly and passed to the planner, which produces a new plan that can be executed successfully, Fig. 9(b). When the bowl is on the sideboard, the human asks the robot for help in preparing a salad with corn and oil. The command is processed by the NLU and a goal is generated, which implies the corn and the oil being in the bowl and the salad being stirred in the bowl. The RM finds that the location of the corn is unknown. Since other location replacement strategies fail, it generates a question for the human inquiring for the corn location. The human tells the robot that the corn is located in the fridge. The utterance is processed by the NLU and *MemoryX* is updated correspondingly. After obtaining the feedback from human, the RM evokes the planner that generates a plan. The robot moves to the fridge, opens it (Fig. 9(c)), moves to the sideboard, pours the corn into the bowl (Fig. 9(d)), puts the empty can into the sink, and returns to the fridge to close it (Fig. 9(e)). The actions of putting the can into the sink and closing the fridge are planned, since we introduce symbolic rules stating that dirty objects should go into the sink after being manipulated and that the fridge door should be closed at the end of each plan execution. After adding the corn, the robot moves to the oil location, grasps the oil, pours it into the bowl (Fig. 9(f)), and puts the oil bottle away. The latter action is also planned, since we define a symbolic rule requiring robot's hands to be empty at the end of each plan execution. In the meanwhile, the human is cutting other salad ingredients and pouring them into the bowl. In order to mix the salad, the robot requires a stirrer. It moves to the assumed stirrer location, but it cannot grasp the stirrer. Instead of grasping, the planner plans a speech request from the human. The human passes the stirrer to the robot. The robot returns to the bowl, stirs the salad (Fig. 9(g)), and puts the stirrer into the sink (Fig. 9(h)). Finally, the human asks the robot to put the bowl on the dinning table, which is performed by the robot (Fig. 9(i)).

In the second part of the scenario, the human is **asking for a drink** saying *I'd like to drink something. Could you please bring me a lemonade*. This command is processed by the NLU component and a goal of the lemonade being in the hand of the human is generated. Apart from generating the goal, the NLU extracts the affordance of drinking for the object "lemonade". The goal and the predicted affordances are passed to the Replacement Manager. The RM finds that the object "lemonade" is unknown and attempts a replacement by using the affordance "drink". The object "multivitamin juice" is proposed as a possible replacement. The RM generates a confirmation utterance *Sorry, I have no lemonade. But I can bring you a multivitamin juice*. After the human confirms the replacement, the RM rewrites the goal and passes it to the planner. According to the generated plan, the robot moves to the assumed location of the juice, but does not find it there. Plan execution fails and the RM is evoked. The RM component derives another potential location of the juice from the database of common locations. The robot finds the juice at the new location, grasps it, moves to the location of the human, and hands the juice over to the human (Fig. 9(j)).

7.2. Simulation experiment

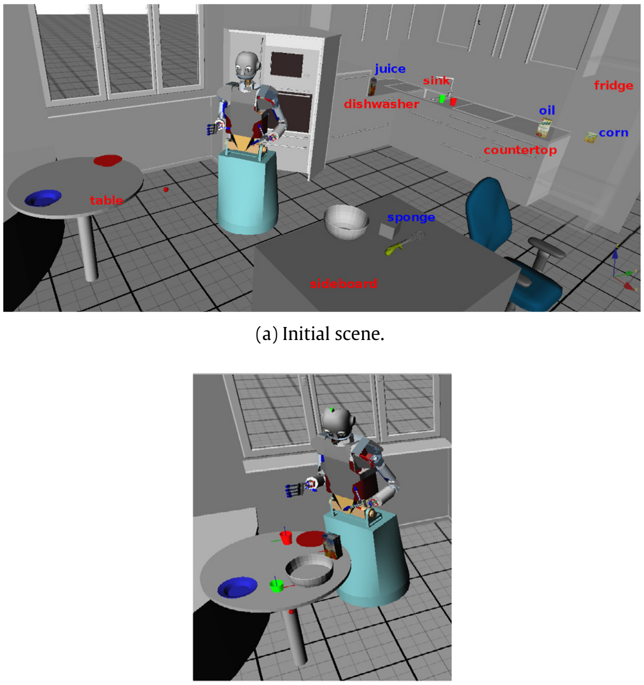
In the simulation experiment, we aimed at testing if the framework can be employed by untrained users. Due to the usage of generic interfaces in *ArmarX*, the programs used on the real robot can also be executed without changes in a simulation environment. To provide the user with information about the simulation, *ArmarX* offers a visualization of the simulated scene and the user can track the robot's action execution and the effects they have on the scene. This Graphical User Interface is shown in Fig. 10. The user can interact with the simulated robot by typing in the text dialog window or by speaking into a microphone.

We asked the subjects to achieve a given goal by controlling the robot with natural language commands in the simulation environment. The instructions were formulated as follows. First, users were advised to communicate with the robot by typing sentences in the dialog widget. The users were shown the initial scene with labels assigned to available locations and some of the objects (Fig. 11(a)), which could not be easily visually recognized. They were also shown the goal table setup (Fig. 11(b)) and asked to achieve it by controlling the robot. In addition, we specified that the drink on the table can be chosen from a predefined list of drinks represented by images (e.g. beer, juice, milk) and the salad should contain ingredients represented by images of corn and oil.



prepare a salad with oil

Fig. 10. Graphical user interface used for the simulation experiments.



(a) Initial scene.

(b) Goal table setup.

Fig. 11. Instruction scenes.

The experiment was run offline on site, with no time constraints. In total, eight subjects (four male, four female, age 20–60) took part in the experiment. All subjects had no background in robotics or related areas. The experimenter was present during the whole time of the experiment to record the experiment transcript,

Table 3
Example experiment transcript. For each input utterance, the transcript records if each of the evoked components worked correctly. For the first command, the language understanding component produced correct goals and passed them to the replacement manager, which produced the suggestion to replace the unknown object "glass" with the known object "cup" and the confirmation question for the user. The next utterance "yes" was correctly processed by LU; given the confirmation, the RM correctly rewrote the goals; there was no memory update; the planner produced a suboptimal goal, which implied removing the juice from the table to free space for putting the cups, while the user asked for the juice to be placed there earlier; the plan was correctly executed on the robot; the user was confused by the fact that the juice was removed from the table.

User id	Text	LU	RM	Memory update	Planner	Execution	User feedback
M2	"Take two glasses and put them on the table"	yes	yes: "glass" → "cup"	–	–	–	–
	"yes"	yes	yes	–	suboptimal: juice removed from the table	yes	user confused

Table 4
Examples of the correct (C), partially correct (P), and incorrect (I) output of the language understanding component.

	Text	LU output	Comment
C	"Can you please set the table for two people"	[goal: (existsK(?x1 : cup, ?x2 : placesetting) K(objectAt(?x1, ?x2)) & (existsK(?x3 : cup, ?x4 : placesetting) K(objectAt(?x3, ?x4)) & K(?x1 != ?x3) & K(?x2 != ?x4))), context: (table,set)]	
P	"Bring a glass and put it on the table"	[goal: ((existsK(?x1 : glass, ?x2 : human) K(inHandOfHuman(?x1,?x2)), (existsK(?x1 : glass, ?x2 : table) K(objectAt(?x1,?x2))), context: ((glass,bring), (table,put))]	"bring" interpreted as "bring to human" instead of "bring to location"
I	"I need milk"	[]	Underspecified command, modality ("need") not recognized

see Table 3. The experimenter was not giving any instructions or answering questions.

All subjects were able to solve the tasks using 12.5 utterances on average. Some subjects provided general task descriptions, e.g. *Make a salad with corn and oil*. Others gave detailed commands, e.g. *Go to the fridge. Open the fridge. Take the corn…* In total, 103 natural language utterances were processed in this experiment. The Language Understanding component processed 87 utterances fully correctly, 3 – partially correctly, and 13 – incorrectly; see examples in Table 4.

The main source of errors are underspecified commands, e.g. *I need milk, Prepare a salad*). Introducing clarification questions asked by the robot is a possible solution to this problem. Another issue concerns processing of the partial input. For example, if the robot asks about the corn location and the user answers *in the fridge* instead of *the corn/it is in the fridge*, such answer is not processed correctly. Currently, we do not have a dialog component that waits for a particular type of input. Instead, we process all types of input all the time. This problem can be solved in our framework by assigning higher assumptions weights to certain syntactic constructions (e.g. prepositional phrases without noun), which will force the abductive reasoner to link them to the previous discourse. A related issue concerns context-dependent metonymy. For example, in the utterance *Put the salad on the table*, the reference *salad* should be linked to the bowl, in which the salad has been prepared. Additional domain axioms are needed to establish this link. One more issue is related to missing lexical items or their meanings in the knowledge base. For example, the verb *put* was defined in our knowledge base as referring to moving an object from one location to another, while some of the users were using it in the sense of *pour*, e.g. *put the corn into the bowl*.

The Replacement Manager component suggested 6 object replacements, 4 of which were accepted by the users. It has also generated location hypotheses for all known objects and asked the users for unknown object locations. One issue we have encountered with this component were the cases when several replacements of the same type were required in a goal expression. For example, the command *Put two glasses on the table* implies that there should be two different glasses on the table. Since no glasses were available in our kitchen environment, the RM suggested replacing them with cups and has chosen the type red_cup as a replacement. Since there was only one instance of red_cup, the rewritten goal implying two different instances could not be fulfilled. To tackle this problem, the RM needs to consider the number of instances of the replaced type and make the replacement only if the required instances are available.

Table 5
Runtime of the main components in simulation, in seconds.

Component	Average	Maximum	Minimum
Language understanding	0.26 s	0.79 s	0.13 s
Visual features strategy	2.61 s	4.61 s	1.66 s
Planning system	0.89 s	7.99 s	0.13 s
Action execution	20.59 s	42.01 s	4.44 s

The working memory was successfully updated during the execution and after processing human utterances. Given a goal that could be fulfilled and a domain description, the planner was producing relevant plans. An issue that we have encountered is that our framework is currently lacking a mechanism to estimate pragmatic context-dependent relevance of the generated plans. For example, a user has commanded the robot to put a drink on the table, which the robot has performed. Some of the following commands may result in a plan that requires the robot to remove the drink, which may contradict the original intention of the user. The ability of storing previous goals and generating new plans without overwriting these goals is a desired functionality for our framework.

The runtime of the main components is depicted in Table 5 and the timing of the action execution on the real robot is shown in Table 6. The host pc used for the measurements was an Intel Core i7-6700HQ CPU @ 2.60 GHz with 16 GB RAM. The tasks from this simulation experiment were used to measure the runtime. The NLU component performed its analysis reliably in under one second (maximum 0.79 s). No input data produced a noticeable delay in the task triggering. The RM and its strategies, except the visual features strategy, are table look ups of precomputed information,

Table 6
Runtime of the action execution in seconds on the real robot.

Action name	Average	Std. Dev. σ
Grasp	38.9 s	2.2 s
Move	8.5 s	2.6 s
Open	121.8 s	1.7 s
Pour	11.5 s	0.5 s
Putdown	20.5 s	0.7 s
Request from human	13.7 s	1.5 s
Stir	55.6 s	1.15 s

e.g. common sense object replacements, and thus consumed no significant CPU time. These strategies were omitted in Table 5. The visual features strategy on the other hand is computationally expensive and required 2.61 s on average. The run time mainly depends on how many point cloud clusters (i.e. objects) have been found in the current scene. The runtime of the planning component highly depends on the specific goal and the current state of the memory and varies between 0.00015 s and 7.99 s for the given tasks. The minimum of 0.00015 s was achieved when the goal was already fulfilled before starting planning. The maximum of 7.99 s was required for the task *set the table for two people*, with all objects except the corn already being in the working memory. The runtime of the action execution was measured per plan step in the simulation. This measurement includes the full duration of the robot's action execution. The runtime is similar on the real robot, but slightly higher due to smaller joint angle velocities and a higher perception time. The runtime highly depends on the executed action; in case of the *move* action it depends on the traveling distance.

8. Related work

In this section, the related work is organized according to specific aspects relevant for this article.

Structural bootstrapping and replacement. The concept of structural bootstrapping was introduced in the context of the *Xperience* project [1]. In [4], it was demonstrated how structural bootstrapping can be performed at different levels of a robotic architecture consisting of a planning level, a symbol-to-signal mediator level, and a sensorimotor level. The concept was applied to acquisition of action knowledge [38], learning action skills based on exploration and interaction with the environment [2] and replacement of missing objects [3,4,10].

In [39], object replacement is carried out based on a similarity measure utilizing (a) object classes with features and affordances coded manually and (b) visual features such as shape and color intensity. In [3], object replacement is performed by employing the ROAR database of objects with their affordances and a learning method predicting unobserved object affordances. Along these lines, we employ structural bootstrapping for object and location replacement based on object affordances and predicted object locations.

Affordance estimation. Early work on affordance estimation followed a function-based approach to object recognition for 3D CAD models of objects such as chairs [40]. Another approach models affordances of objects as a function of human–object interactions. For example, in [41], 3D geometric properties are computed for tracked humans and objects in order to describe human–object interactions. In [42], actions and objects in human demonstrations are recognized and dependencies between them are modeled. In [43], object affordances are represented by clustered spatial configurations of human–object interactions. Other researchers focus on defining a subset of attributes for predicting new object categories [44–46]. For example, by learning 2D shape and color patterns, attributes of novel objects can be recognized – [47]. In the field of NL processing, there exist developed methods for extracting common-sense knowledge from textual corpora, which can also be applied for mining object affordances, cf. [48–50]. For example, [51] use verb–noun co-occurrences in Google Syntactic N-Grams as well as Latent Semantic Analysis and Word2Vec semantic vectors to extract verbs that are most likely to refer to affordances of objects represented by given nouns. In our work, we extract object affordances from visual features, textual corpora, and dialog context.

Grounding NL. Approaches to grounding NL into actions, relations, and objects known to the robot can be roughly subdivided into symbolic and statistical. Symbolic approaches rely on sets of rules to map linguistic constructions into pre-specified action spaces and sets of environmental features. In [52], simple rules are used to map NL instructions having a pre-defined structure to robot skills and task hierarchies. In [53], NL instructions are processed with a dependency parser and background axioms are used to make assumptions and fill the gaps in the NL input. In [54], background knowledge about robot actions is axiomatized using Markov Logic Networks. In [55], a knowledge base of known actions, objects, and locations is used for a Bayes-based grounding model. Symbolic approaches work well for small pre-defined domains, but most of them employ manually written rules, which limits their coverage and scalability. In order to increase the linguistic coverage, some of the systems use lexical-semantic resources like WordNet, FrameNet, and VerbNet [54,56]. In this study, we follow this approach and generate our lexical axioms from Wordnet and FrameNet.

Statistical approaches rely on annotated corpora to learn mappings between linguistic structures and grounded predicates representing the external world. In [57], reinforcement learning is applied to interpret NL directions in terms of landmarks on a map. In [58], machine translation is used to translate from NL route instructions to a map of an environment built by a robot. In [59], Generalized Grounding Graphs are presented that define a probabilistic graphical model dynamically according to linguistic parse structures. In [28], a verb-environment-instruction library is used to learn the relations between the language, environment states, and robotic instructions in a machine learning framework. Statistical approaches are generally better at handling NL variability. An obvious drawback of these approaches is that they generate noise and require a significant amount of annotated training data, which can be difficult to obtain for each new application domain and set of action primitives.

Some recent work focuses on building joint models explicitly considering perception at the same time as parsing [60,61]. The framework presented in this article is in line with this approach, because abductive inference considers both the linguistic and perceptual input as an observation to be interpreted given the background knowledge.

Our approach to grounding is also in line with [62] proposing to ground language and other kind of symbolic information to so called Object Action Complexes (OACs). OACs provide a framework, in which experience is systematically structured and linked to specific actions, which on the one hand are described by symbolic state transitions allowing for planning, and on the other hand, by the relevant sub-symbolic information allowing for reasoning in continuous and ambiguous signal space. Our mapping of linguistic structures to predicate providers (Section 3) and statecharts (Section 6.2) can be seen as a realization of the OAC framework.

Planning. With respect to the action execution, the existing approaches can be classified into those directly mapping NL instructions into action sequences [54,63,64] and those employing a planner [53,56,65,66]. We employ a planner, because it allows us to account for the dynamically changing environment, which is essential for the human–robot collaboration. Similar to [66], we translate a NL command into a goal description.

World descriptions. Although most of the NLU systems in robotics focus directly on instruction interpretations, there are a few systems detecting world descriptions implicitly contained in human commands [53,55,67]. These descriptions are further used in the planning context, as it is done in our approach. In addition, we detect world descriptions not embedded into the context of an instruction and process human action descriptions and feedback.

Linking planning, NL, and sensorimotor experience. Interaction between NL instructions, resulting symbolic plans, and sensorimotor experience during plan execution has been previously explored in the literature. In [63], symbolic representations of objects, object locations, and robot actions, are mapped on the fly to the sensorimotor information. During the execution of the predefined plans, the plan execution monitoring component evaluates the outcome of each robot's action as success or failure. In [68], symbolic representations are generated based on several sensorimotor features needed for segmentation and inference of tasks. In [53], the planner knowledge base is updated each time a NL instruction related to the current world state is provided and the planner re-plans taking into consideration the new information. In [65], symbolic planning is employed to plan a sequence of motion primitives for executing a predefined baking primitive given the current world state. Replacement of missing objects and re-planning is performed in [39,69]. In line with these studies, mapping sensorimotor data to symbols, plan execution monitoring, as well as object replacement is a part of our system.

9. Conclusion

We have presented a realization of the concept of structural bootstrapping in a framework integrating sensorimotor experience, robot's memory, natural language understanding, and planning in a robotic architecture developed in the context of the *Xperience* project. We showed that the developed framework is flexible enough to be used for action execution on a humanoid robot in a complex domain and can process input from untrained users in a scenario requiring human–robot interaction and collaboration. The limitations of each system component are discussed in detail in the corresponding sections. The main issues that need to be addressed in the future work are (a) processing of the underspecified/partial linguistic input, (b) incorporating pragmatic context-dependent relevance of the generated plans, and (c) equipping the knowledge base with deeper domain knowledge required both for language understanding and for relevant planning and replacement. Another current limitation concerns the inability of the framework to learn new objects and environment models. New objects and information required for interacting with them, e.g. 3D mesh models, object localization descriptors and grasp information, are currently manually added to the prior knowledge database. Similarly, a semantic model of the environment has to be developed manually. The adjustment of the system to a new environment requires landmarks with a suitable pose for performing various manipulation actions.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement No. 270273 (*Xperience*). We would also like to thank M. Do, C. Geib, M. Grotz, M. Kröhnert, D. Schiebener, and N. Vahrenkamp for their various contributions to the underlying system, which made this article possible.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2017.10.012.

References

[1] Xperience project, website, available online at http://www.xperience.org.
[2] M. Do, J. Schill, J. Ernesti, T. Asfour, Learn to wipe: A case study of structural bootstrapping from sensorimotor experience, in: Proc. of ICRA, 2014.
[3] A. Agostini, M. Javad Aein, S. Szedmak, E.E. Aksoy, J. Piater, F. Worgotter, Using structural bootstrapping for object substitution in robotic executions of human-like manipulation tasks, in: Proc. of IROS, 2015, pp. 6479–6486.
[4] F. Wörgötter, C. Geib, M. Tamosiunaite, E.E. Aksoy, J. Piater, H. Xiong, A. Ude, B. Nemec, D. Kraft, N. Krüger, M. Wächter, T. Asfour, Structural bootstrapping - a novel concept for the fast acquisition of action-knowledge, IEEE Trans. Auton. Ment. Dev. 7 (2) (2015) 140–154.
[5] J.J. Gibson, The theory of affordances, Hilldale, 1977.
[6] N. Vahrenkamp, M. Wächter, M. Kröhnert, K. Welke, T. Asfour, The ArmarX framework – supporting high level robot programming through state disclosure, Inf. Technol. 57 (2) (2015) 99–111.
[7] T. Asfour, K. Regenstein, P. Azad, J. Schroder, A. Bierbaum, N. Vahrenkamp, R. Dillmann, ARMAR-III: An integrated humanoid platform for sensory-motor control, in: Proc. of Humanoids, 2006, pp. 169–175.
[8] E. Ovchinnikova, M. Wächter, V. Wittenbeck, T. Asfour, Multi-purpose natural language understanding linked to sensorimotor experience in humanoid robots, in: Proc. of Humanoids, 2015, pp. 365–372.
[9] P. Kaiser, M. Lewis, R.P.A. Petrick, T. Asfour, M. Steedman, Extracting common sense knowledge from text for robot planning, in: Proc. of ICRA, 2014, pp. 3749–3756.
[10] W. Mustafa, M. Wächter, S. Szedmak, A. Agostini, D. Kraft, T. Asfour, J. Piater, F. Wörgötter, N. Krüger, Affordance estimation for vision-based object replacement on a humanoid robot, in: Proc. of ISR, 2016, p. 2017.
[11] S. Szedmak, E. Ugur, J. Piater, Knowledge propagation and relation learning for predicting action effects, 2014.
[12] S. Krivic, S. Szedmak, H. Xiong, J. Piater, Learning missing edges via kernels in partially-known graphs, in: European Symposium on Artificial Neural Networks ESANN, Computational Intelligence and Machine Learning, 2015.
[13] M. Henning, A new approach to object-oriented middleware, IEEE Internet Comput. 8 (1) (2004) 66–75. http://dx.doi.org/10.1109/MIC.2004.1260706.
[14] K. Welke, P. Kaiser, A. Kozlov, N. Adermann, T. Asfour, M. Lewis, M. Steedman, Grounded spatial symbols for task planning based on experience, in: Proc. of Humanoids, 2013, pp. 484–491.
[15] P. Azad, T. Asfour, R. Dillmann, Combining Harris interest points and the SIFT descriptor for fast scale-invariant object recognition, in: Proc. of IROS, 2009, pp. 4275–4280.
[16] P. Azad, D. Münch, T. Asfour, R. Dillmann, 6-dof model-based tracking of arbitrarily shaped 3d objects, in: Proc. of ICRA, 2011, pp. 5204–5209.
[17] M. Wächter, S. Ottenhaus, M. Kröhnert, N. Vahrenkamp, T. Asfour, The ArmarX statechart concept: Graphical programming of robot behaviour, Frontiers - Software Architectures for Humanoid Robotics.
[18] J.R. Hobbs, M.E. Stickel, D.E. Appelt, P.A. Martin, Interpretation as abduction, Artif. Intell. 63 (1–2) (1993) 69–142.
[19] H. Soltau, F. Metze, C. Fügen, A. Waibel, A one-pass decoder based on polymorphic linguistic context assignment, in: Proc. of ASRU, 2001, pp. 214–217.
[20] N. Inoue, E. Ovchinnikova, K. Inui, J.R. Hobbs, Weighted abduction for discourse processing based on integer linear programming, in: Plan, Activity, and Intent Recognition, 2014, pp. 33–55.
[21] J.R. Hobbs, Ontological promiscuity, in: Proc. of ACL, 1985, pp. 60–69.
[22] J. Bos, Wide-coverage semantic analysis with boxer, in: Proc. of STEP, Research in Computational Semantics, 2008, pp. 277–286.
[23] E. Ovchinnikova, R. Israel, S. Wertheim, V. Zaytsev, N. Montazeri, J. Hobbs, Abductive inference for interpretation of metaphors, in: Proc. of ACL Workshop on Metaphor in NLP, 2014, pp. 33–41.
[24] E. Ovchinnikova, A.S. Gordon, J. Hobbs, Abduction for discourse interpretation: A probabilistic framework, in: Proc. of JSSP, 2013, pp. 42–50.
[25] C. Fellbaum, WordNet: An electronic lexical database, 1998.
[26] C.F. Baker, C.J. Fillmore, J.B. Lowe, The Berkeley FrameNet project, in: Proc. of COLING-ACL, 1998, pp. 86–90.
[27] E. Ovchinnikova, Integration of World Knowledge for Natural Language Understanding, Springer, 2012.
[28] D.K. Misra, J. Sung, K. Lee, A. Saxena, Tell me dave: Context-sensitive grounding of natural language to manipulation instructions, in: Proc. of RSS.
[29] R. Ros, S. Lemaignan, E.A. Sisbot, R. Alami, J. Steinwender, K. Hamann, F. Warneken, Which one? grounding the referent based on efficient human–robot interaction, in: Proc. of RO-MAN, 2010, pp. 570–575.
[30] R.P. Petrick, F. Bacchus, A knowledge-based approach to planning with incomplete information and sensing, in: Proc. of AIPS, 2002, pp. 212–222.

[31] G. Bakir, T. Hofman, B. Schölkopf, A.J. Smola, B. Taskar, S.V.N. Vishwanathan (Eds.), Predicting Structured Data, MIT Press, 2007.
[32] M. Ghazanfar, A. Prugel-Bennett, S. Szedmak, Kernel mapping recommender system algorithms, Inf. Sci. 208 (2012) 81–104.
[33] W. Mustafa, N. Pugeault, A.G. Buch, N. Krüger, Multi-view object instance recognition in an industrial context, Robotica (2015) 1–22.
[34] D. Kraft, W. Mustafa, M. Popović, J.B. Jessen, A.G. Buch, T.R. Savarimuthu, N. Pugeault, N. Krüger, Using surfaces and surface relations in an early cognitive vision system, Mach. Vis. Appl. 26 (7–8) (2015) 933–954.
[35] T. Joachims, T. Finley, C.-N.J. Yu, Cutting-plane training of structural svms, Mach. Learn. 77 (1) (2009) 27–59.
[36] N. Vahrenkamp, S. Wieland, P. Azad, D. Gonzalez-Aguirre, T. Asfour, R. Dillmann, Visual servoing for humanoid grasping and manipulation tasks, in: IEEE/RAS International Conference on Humanoid Robots (Humanoids), 2008, pp. 406–412.
[37] J. Ernesti, L. Righetti, M. Do, T. Asfour, S. Schaal, Encoding of periodic and their transient motions by a single dynamic movement primitive, in: IEEE/RAS International Conference on Humanoid Robots (Humanoids), 2012, pp. 57–64.
[38] E.E. Aksoy, M. Tamosiunaite, R. Vuga, A. Ude, C. Geib, M. Steedman, F. Worgotter, Structural bootstrapping at the sensorimotor level for the fast acquisition of action knowledge for cognitive robots, in: Proc. of ICDL, 2013, pp. 1–8.
[39] I. Awaad, G.K. Kraetzschmar, J. Hertzberg, Finding ways to get the job done: An affordance-based approach, in: Proc. of ICAPS, 2014.
[40] L. Stark, K. Bowyer, Function-based generic recognition for multiple object categories, CVGIP: Image Understanding 59 (1) (1994) 1–21.
[41] H.S. Koppula, R. Gupta, A. Saxena, Learning human activities and object affordances from RGBD videos, Int. J. Robot. Res. 32 (8) (2013) 951–970.
[42] H. Kjellström, J. Romero, D. Kragić, Visual object-action recognition: Inferring object affordances from human demonstration, Comput. Vis. Image Underst. 115 (1) (2011) 81–90.
[43] B. Yao, J. Ma, L. Fei-Fei, Discovering object functionality, in: Proc. of ICCV, 2013, pp. 2512–2519.
[44] D. Parikh, K. Grauman, Relative attributes, in: PrOC. of ICCV, IEEE, 2011, pp. 503–510.
[45] C.H. Lampert, H. Nickisch, S. Harmeling, Learning to detect unseen object classes by between-class attribute transfer, in: PrOC. of CVPR, IEEE, 2009, pp. 951–958.
[46] X. Yu, Y. Aloimonos, Attribute-based transfer learning for object categorization with zero/one training example, in: Proc. of ECCV, 2010, pp. 127–140.
[47] V. Ferrari, A. Zisserman, Learning visual attributes, in: Proc. of NIPS, 2007, pp. 433–440.
[48] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E.R. Hruschka Jr., T.M. Mitchell, Toward an architecture for never-ending language learning, in: Proc. of AAAI, 2010.
[49] C.L. Teo, Y. Yang, H. Daumé III, C. Fermüller, Y. Aloimonos, A corpus-guided framework for robotic visual perception, in: Proc. of Workshop on Language-Action Tools for Cognitive Artificial Agents, 2011, pp. 36–42.
[50] K. Zhou, M. Zillich, H. Zender, M. Vincze, Web mining driven object locality knowledge acquisition for efficient robot behavior, in: Proc. of IROS, IEEE, 2012, pp. 3962–3969.
[51] Y.-W. Chao, Z. Wang, R. Mihalcea, J. Deng, Mining semantic affordances of visual object categories, in: Proc. of CVPR, 2015, pp. 4259–4267.
[52] P.E. Rybski, K. Yoon, J. Stolarz, M.M. Veloso, Interactive robot task training through dialog and demonstration, in: Proc. of HRI, 2007, pp. 49–56.
[53] R. Cantrell, K. Talamadupula, P.W. Schermerhorn, J. Benton, S. Kambhampati, M. Scheutz, Tell me when and why to do it!: run-time planner model updates via natural language instruction, in: Proc. of HRI, 2012, pp. 471–478.
[54] D. Nyga, M. Beetz, Everything robots always wanted to know about housework (but were afraid to ask), in: Proc. of IROS, 2012, pp. 243–250.
[55] T. Kollar, V. Perera, D. Nardi, M. Veloso, Learning environmental knowledge from task-based human–robot dialog, in: Proc. of ICRA, 2013, pp. 4304–4309.
[56] D.J. Brooks, C. Lignos, C. Finucane, M.S. Medvedev, I. Perera, V. Raman, H. Kress-Gazit, M. Marcus, H.A. Yanco, Make it so: Continuous, flexible natural language interaction with an autonomous robot, in: Proc. of the Grounding Language for Physical Systems Workshop at AAAI, 2012.
[57] A. Vogel, D. Jurafsky, Learning to follow navigational directions, in: Proc. of ACL, 2010, pp. 806–814.
[58] C. Matuszek, D. Fox, K. Koscher, Following directions using statistical machine translation, in: Proc. of ACM/IEEE, 2010, pp. 251–258.
[59] T. Kollar, S. Tellex, M.R. Walter, A. Huang, A. Bachrach, S. Hemachandra, E. Brunskill, A. Banerjee, D. Roy, S. Teller, Generalized grounding graphs: A probabilistic framework for understanding grounded language, JAIR.
[60] J. Krishnamurthy, T. Kollar, Jointly learning to parse and perceive: Connecting natural language to the physical world, Trans. of ACL 1 (2013) 193–206.
[61] C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, D. Fox, A joint model of language and perception for grounded attribute learning, arXiv preprint arXiv:1206.6423.
[62] N. Krüger, C. Geib, J. Piater, R. Petrick, M. Steedman, F. Wörgötter, A. Ude, T. Asfour, D. Kraft, D. Omrcen, A. Agostini, R. Dillmann, Object-action complexes: Grounded abstractions of sensori-motor processes, Robot. Auton. Syst. 59 (10) (2011) 740–757.
[63] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mosenlechner, D. Pangercic, T. Ruhr, M. Tenorth, Robotic roommates making pancakes, in: Proc. of Humanoids, 2011, pp. 529–536.
[64] C. Matuszek, E. Herbst, L. Zettlemoyer, D. Fox, Learning to parse natural language commands to a robot control system, in: Experimental Robotics, Springer, 2013, pp. 403–415.
[65] M. Bollini, S. Tellex, T. Thompson, N. Roy, D. Rus, Interpreting and executing recipes with a cooking robot, in: Experimental Robotics, 2013, pp. 481–495.
[66] J. Dzifcak, M. Scheutz, C. Baral, P. Schermerhorn, What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution, in: Proc. of ICRA, 2009, pp. 4163–4168.
[67] F. Duvallet, M.R. Walter, T. Howard, S. Hemachandra, J. Oh, S. Teller, N. Roy, A. Stentz, Inferring maps and behaviors from natural language instructions, in: Proc. of ISER, 2014.
[68] K. Ramirez-Amaro, E. Dean-Leon, I. Dianov, F. Bergner, G. Cheng, General recognition models capable of integrating multiple sensors for different domains, in: Proc. of Humanoids, 2016, pp. 306–311.
[69] S. Konecnỳ, S. Stock, F. Pecora, A. Saffiotti, Planning domain+ execution semantics: a way towards robust execution? in: Proc. of Qualitative Representations for Robots, AAAI Spring Symposium, 2014.

Mirko Wächter received the diploma degree in computer science from the Karlsruhe Institute of Technology (KIT), Germany in 2011. He is currently a doctoral researcher at the Karlsruhe Institute of technology (KIT) where he is a member of the High Performance Humanoid Technologies lab. His major research interests are physical robot interaction, robot cooperation with humans or other robots, motion synthesis and robotic software development.

Ekaterina earned her Ph.D. degree in Cognitive Science from the Institute of Cognitive Science, University of Osnabrück, Germany. In her dissertation project, she was focusing on inference-based natural language understanding. Ekaterina then worked as a computer scientist at the Institute of Information Sciences, University of Southern California, USA. Later, Ekaterina moved back to Germany and joined the Karlsruhe Institute of Technology to work on human–robot interaction. Currently, Ekaterina is employed at the European Molecular Biology Laboratory in Heidelberg, where she is applying her skills to such diverse fields as natural language processing, metabolomics, and coral health biology.

Valerij Wittenbeck received his master's degree in computer science from the Karlsruhe Institute of Technology (KIT) in 2016. He was a student assistant for several years at the High Performance Humanoid Technologies lab. There, he also authored his master's thesis on the topic of symbolic planning for task execution.

Peter Kaiser received his diploma degree from Karlsruhe Institute of Technology (KIT) in 2014. He was a visiting researcher at Carnegie Mellon University in 2010 and at University of Edinburgh in 2013. He is currently a Ph.D. student at the Institute for Anthropomatics and Robotics at KIT where he is a member of the High-Performance Humanoid Technologies group.

His major research interests are planning and learning of whole-body actions, in particular actions related to locomotion and manipulation, and the detection of action opportunities based on the concept of affordances.

Sandor Szedmak was born in Hungary. He obtained an M.S. degree in Mathematics from Lajos Kossuth University, Debrecen, Hungary. He received the Ph.D. in Operations Research from the Rutgers, The State University of New Jersey, in the USA. He held research positions at the Computer Science Department of Royal Holloway, University of London and the University of Helsinki. Following that he was a senior research fellow at the School of Electronics and Computer Science of University of Southampton in the United Kingdom, and in the Informatics Institute at the University of Innsbruck in Austria.

Recently he is a senior researcher at the Department of Computer Science of the Aalto University in Finland. He main research interest focuses on mathematical problems, e.g. optimization, arising in machine learning and pattern recognition.

Wail Mustafa received the B.Sc. degree in electrical engineering from University of Khartoum, Sudan, in 2006, the M.Sc. degree in electrical engineering (signal processing) from Blekinge Institute of Technology, Sweden, in 2010, and the Ph.D. degree in robotics from University of Southern Denmark in 2015. In 2016, he was a R&D engineer at Universal Robots, Denmark. He is currently a postdoctoral researcher at The Technical University of Denmark. His main research interests are robotics vision, machine learning and image processing.

Dirk Kraft is an Associate Professor at the Mærsk McKinney Møller Institute, University of Southern Denmark. He holds a diploma degree in computer science from the University of Karlsruhe, Germany and a Ph.D. degree from the University of Southern Denmark. His research interests lie within cognitive systems, computer vision, and robotics and practical applications of these domains to industrial challenges.

Norbert Krüger is a professor at the Mærsk McKinney Møller Institute, University of Southern Denmark. He holds a M.Sc. degree from the Ruhr-Universität Bochum, Germany and his Ph.D. degree from the University of Bielefeld, Germany. His research covers computer vision, cognitive systems and applied robotics.

Justus Piater is a professor of computer science at the University of Innsbruck, Austria, where he leads the Intelligent and Interactive Systems group. He holds a M.Sc. degree from the University of Magdeburg, Germany, and M.Sc. and Ph.D. degrees from the University of Massachusetts Amherst, USA, all in computer science. Before joining the University of Innsbruck in 2010, he was a visiting researcher at the Max Planck Institute for Biological Cybernetics in Tübingen, Germany, a professor of computer science at the University of Liège, Belgium, and a Marie-Curie research fellow at GRAVIR-IMAG, INRIA Rhône-Alpes, France. His research interests focus on visual perception, learning and inference in sensorimotor systems. He has published more than 160 papers in international journals and conferences, several of which have received best-paper awards, and currently serves as Associate Editor of the IEEE Transactions on Robotics.

Tamim Asfour is full professor at the Institute for Anthropomatics and Robotics at the Karlsruhe Institute of Technology (KIT), where he holds the chair of Humanoid Robotics Systems and is the head of the High Performance Humanoid Technologies Lab (H2T). His major research interest is engineering high performance humanoid robots. He is developer and leader of the development team of the ARMAR humanoid robot family.