Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web

Technical Report IAS-2009-01

Intelligent Autonomous Systems Group, Technische Universität München
Boltzmannstr. 3, 85748 Garching bei München, Germany
{tenorth, nyga, beetz}@cs.tum.edu

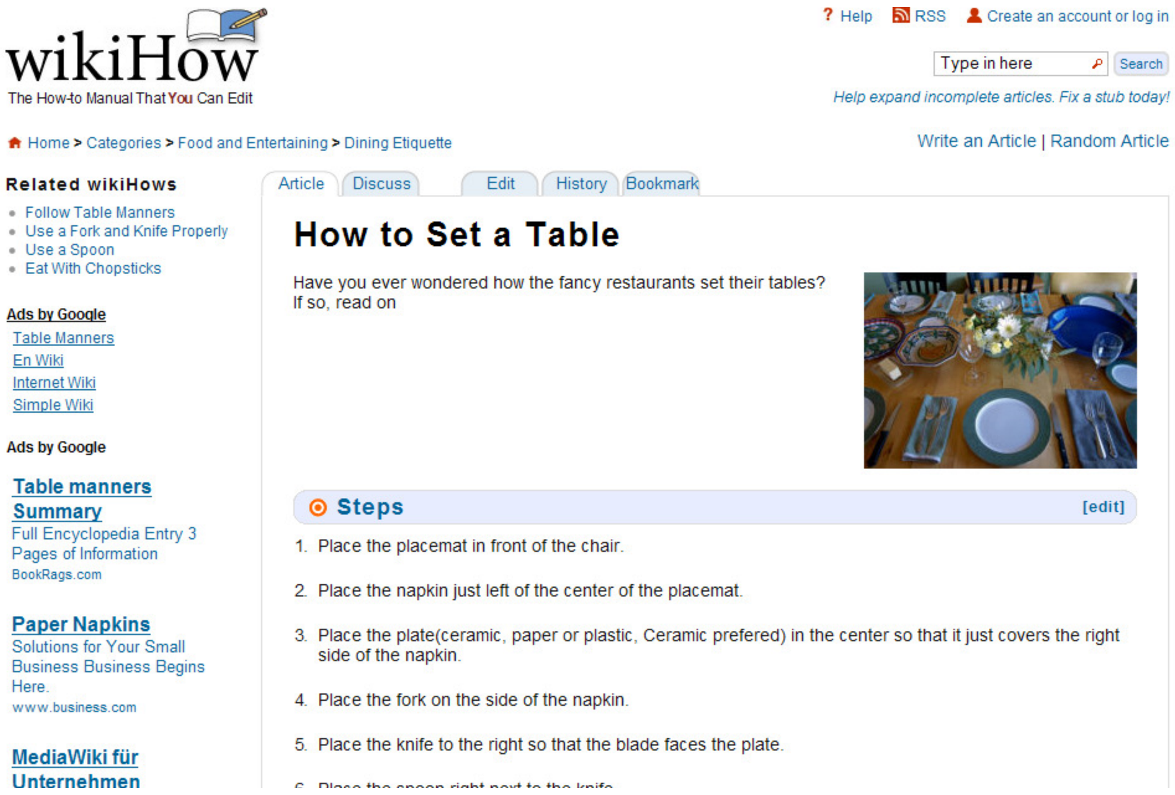
Abstract

Service robots will have to accomplish more and more complex, open-ended tasks and regularly acquire new skills. In this work, we propose a new approach to generating plans for such household robots. Instead composing them from atomic actions, we propose to transform task descriptions on web sites like ehow.com into executable robot plans. We present methods for automatically converting the instructions given in natural language into a formal, logic-based representation, for resolving the word senses using the WordNet database and the Cyc ontology, and for exporting the generated plans into the mobile robot's plan language RPL. We discuss the problems of inferring information missing in these descriptions, of grounding the abstract task descriptions in the perception and action system, and we propose techniques for solving them. The whole system works autonomously without human interaction. It has successfully been tested with a set of about 150 natural language directives, of which up to 80% could be correctly transformed.

1 Introduction

Consider autonomous personal robots that are to perform housework ([1, 2, 3]). Such a robot has to set the table. It has to cook spaghetti. In cases where children are visiting it should cook the spaghetti on the plates in the back to make the kitchen childsafe.

One of the key challenges for autonomous personal robots that are to perform everyday manipulation tasks in households is the openendedness of the task domain. It is an open challenge to generate the range of plans that contain such rich specifications of how actions are to be executed, what events to wait for before executing the next actions, which additional behavior constraints to satisfy, and which problems to watch for. The expressiveness of the necessary task instructions goes well beyond the expressiveness


wikiHow
The How-to Manual That You Can Edit

? Help RSS Create an account or log in

Type in here Search

Help expand incomplete articles. Fix a stub today!

Home > Categories > Food and Entertaining > Dining Etiquette

Write an Article | Random Article

Related wikiHows

- Follow Table Manners
- Use a Fork and Knife Properly
- Use a Spoon
- Eat With Chopsticks

Ads by Google
Table Manners
En Wiki
Internet Wiki
Simple Wiki

Ads by Google

Table manners Summary
Full Encyclopedia Entry 3 Pages of Information
BookRags.com

Paper Napkins
Solutions for Your Small Business Business Begins Here.
www.business.com

MediaWiki für Unternehmen

Article Discuss Edit History Bookmark

How to Set a Table

Have you ever wondered how the fancy restaurants set their tables? If so, read on

Steps [edit]

1. Place the placemat in front of the chair.
2. Place the napkin just left of the center of the placemat.
3. Place the plate(ceramic, paper or plastic, Ceramic prefered) in the center so that it just covers the right side of the napkin.
4. Place the fork on the side of the napkin.
5. Place the knife to the right so that the blade faces the plate.

Figure 1: Example task description from wikihow.com.

of planning problem descriptions used in the AI action planning area [4]. Even if the planning problems could be expressed, the number of objects and actions including their possible parameterizations would result in search spaces that are not tractable by the search algorithms of these planning systems.

Thus, a promising alternative is to look up a new task on webpages such as ehow.com and wikihow.com, which provide step-by-step instructions for setting the table (Figure 1), cooking spaghetti or making a kitchen childsafe, and to convert these natural language instructions into executable robot plans. About 45,000 howto-like task descriptions on wikihow.com and even more than 250,000 on ehow.com, including thousands of household tasks, cover the whole range of everyday activity. We currently use only these two websites to make sure that all instructions have a fairly similar structure and are actually meant to be executed. If the robot would simply use a search engine, it could also find fictional texts describing actions that it better should not execute.

The translation of these web instructions can be performed as a three stage process:

1. *Translation of the natural language instructions into an almost working but buggy robot plan.* Due to the fact that web instructions are written to be executed by people with commonsense knowledge, the instructions may contain ambiguities, missing parameter information and even missing plan steps. An important task is the grounding of natural-language words into physical objects and locations in the robot's environment.

2. *Debugging of the plan.* In a second step, the above plan flaws are to be detected, diagnosed, and forestalled using transformational planning based on mental simulations of the plans in a simulated

environment [5]

3. *Plan optimization.* Web instructions also fail to specify how tasks can be carried out efficiently. Thus, transformational planning is applied to find out that the table setting task can be carried out more efficiently if the robot stacks the plates before carrying them, if it carries cups in each hand, and if it leaves the cupboard doors open while setting the table [6].

In this paper we design, implement, and empirically evaluate a system that performs the first computational task: the translation of the natural language instructions into an almost working but buggy robot plan. We limit ourselves to tasks that can be characterized as "mobile manipulation" and involve picking up, putting down and handling objects at different places. Examples of such tasks are setting a table, cleaning up, making toast or cooking tea.

To the best of our knowledge this work is the first to mine complex household chores from the web and translate them into executable robot plans. Various approaches exist for building speech interfaces to robots, but normally, they are quite limited in terms of vocabulary or allowed grammatical structures ([7], [8], [9]). Kate et al. [10] use similar methods as we do for the semantic parsing, but do not apply them to web instructions and do not provide details of the knowledge processing and symbol grounding. Perkowitz et al. [11] also used task descriptions from `ehow.com`, but only extracted sequences of object interactions for activity recognition, while we generate executable action descriptions.

The main contributions of the paper are the following:

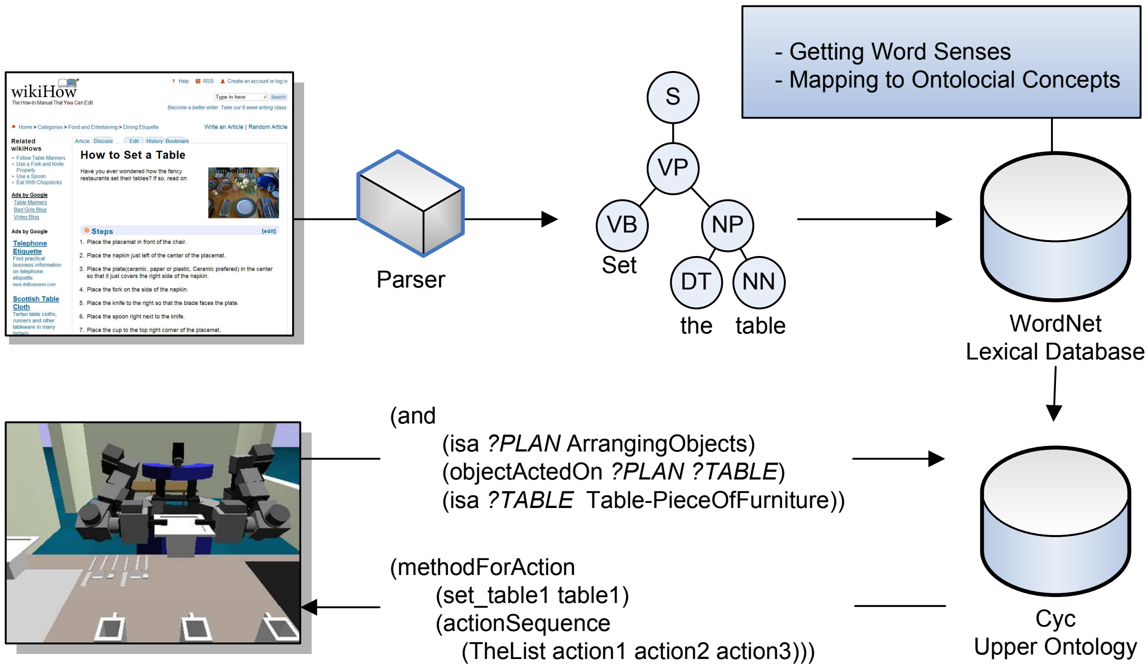
- We demonstrate that it is feasible to automatically generate executable robot plans from naturallange instructions taken from websites.
- We present techniques for semantically parsing instructions, for automatically resolving the ontological concepts belonging to the words involved, and for translating them into grounded symbolic representations that are linked to the perception and action system.
- We propose methods which exploit common sense knowledge, a rich environment model and observations of previous actions for inferring information that is missing in the howtos.

We do not see the main contributions of this paper in the area of natural language processing, which is only the first step before the resolution of the words' meanings, the mapping to grounded symbols, the generation of executable plans and the inference of missing information.

The remainder of the paper is organized as follows: We start with the semantic parsing of the instructions (2.1), continue with the resolution of word senses (2.2), the internal plan representation (2.3), and finally the export into the RPL language (2.4). We briefly sketch the plan debugging (2.5) and explain how the system infers missing information (2.6). We finish with the evaluation results (3), a discussion of the performance of the system (4) and our conclusions.

2 Translating Instructions

In this section, we will present the different steps from the instruction in natural language to an executable plan with the example sentence “Place the cup on the table”. Figure 2 shows the overall structure of our system.



```
(and
    (isa ?PLAN ArrangingObjects)
    (objectActedOn ?PLAN ?TABLE)
    (isa ?TABLE  Table-PieceOfFurniture))

(methodForAction
    (set_table1 table1)
    (actionSequence
      (TheList action1 action2 action3)))
```

Figure 2: Overview of the import procedure. After determining the syntactic structure, the system resolves the meaning of the words and builds up a formal plan representation which can afterwards be transformed into an executable robot plan.

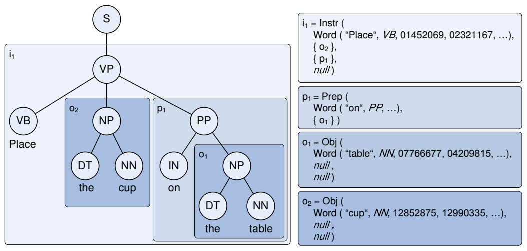
2.1 Semantic Parsing

Starting from the syntax tree generated by a PCFG parser [12], increasingly complex semantic concepts are generated in a bottom-up fashion using transformation rules similar to those in [10].

The leaves of the parse tree are words *Word(label, pos, synsets)*, consisting of a label, a part-of-speech (POS) tag and the synsets they belong to (see Section 2.2). Examples of POS tags are *NN* for a noun, *JJ* for an adjective or *CD* for a cardinal number. In the following, an underscore means that the respective slot can be filled with an arbitrary value.

Words can be accumulated to a quantifier *Quant(Word(_,CD,_),Word(_,NN,_))* consisting of a cardinal number and a unit, or an object *Obj(Word(_,NN,_),Word(_,JJ,_), Prep, Quant)* that is described by a noun, an adjective, prepositional statements and quantifiers. A prepositional phrase contains a preposition word and an object instance *Prep(Word(_,IN,_),Obj)*, and an instruction is described as *Instr(Word(_,VB,_),Obj,Prep,Word(_,CD,_))* with a verb, objects, prepositional postconditions and time constraints. Since some of the fields are optional and since the descriptions can be nested due to the recursive definitions, this method allows for representing complex relations like "to the left of the top left corner of the place mat".

Figure 3 exemplarily shows how the parse tree is translated into two *Obj* instances, one *Prep* and one *Instr*.



```
i1 = Instr (
    Word ( "Place", VB, 01452069, 02321167, ...),
    { o2 },
    { p1 },
    null )
p1 = Prep (
    Word ( "on", PP, ...),
    { o1 } )
o1 = Obj (
    Word ( "table", NN, 07766677, 04209815, ...),
    null ,
    null )
o2 = Obj (
    Word ( "cup", NN, 12852875, 12990335, ...),
    null ,
    null )
```

Figure 3: Parse tree for the sentence "Place the cup on the table" (left) and the resulting data structures representing the instruction that are created as an intermediate representation by our algorithm (right).

Some automatic post-processing of the generated data structures resolves object names consisting of multiple words (like "stove top"), phrasal verbs (like "turn on"), and pronomial anaphora (references using pronouns like "it"). Currently, we assume that "it" always refers to the last mentioned object, which proved to be a sensible heuristic in most cases. The system also handles conjunctions and alternative instructions ("and", "or"), negations, and sentences starting with modal verbs like "You should..." as long as the rest of the sentence is an imperative statement. The slight difference in meaning presented by the modal verbs is difficult to translate into a robot plan and therefore currently ignored.

2.2 Word Sense Retrieval and Disambiguation

Once the structure of instructions is identified, the system resolves the meaning of the words using the WordNet lexical database [13] and the Cyc ontology [14]. In WordNet, each word can have multiple senses, i.e. it is contained in multiple "synsets". There exist mappings from the synsets in WordNet to ontological concepts in Cyc via the *synonymousExternalConcept* predicate. "Cup" as a noun, for instance, is part of the synsets *N03033513* and *N12852875* which are mapped to the concepts *DrinkingMug* and *Cup-UnitOfVolume* respectively.

Most queries return several synsets for each word, so a word sense disambiguation method has to select one of them. The algorithm we chose is based on the observation that the word sense of the action verb is strongly related to the prepositions (e.g. "taking something from" as *TakingSomething* in Cyc up vs. "taking something to" as *PuttingSomethingSomewhere*).

Let $concepts(w)$ be the set of ontological concepts to which the word w could be mapped. For a single instruction (a_i, o_i, p_i) consisting of an action verb a_i, an object o_i and a set of prepositions $p_i \subseteq \{on, in, to, from, of, next\_to, with, without\}$, we are interested in the most probable pair of concepts

$(A_i, O_i) \in concepts(a_i) \times concepts(o_i)$. Because the most appropriate concept for the action is, as mentioned above, largely dependent on the prepositions it co-occurs with, whereas it is reasonable to assume that the object sense is independent of the prepositions given the action sense, we compute the pair by maximizing

$$\begin{aligned} P(O_i, A_i \mid p_i) &= P(O_i|A_i) \cdot P(A_i|p_i) \\ &\propto \frac{P(O_i, A_i)}{P(A_i)} \cdot P(A_i, p_i) \end{aligned}$$

The required probability values appearing in the above formulas are determined by supervised learning on a training set. If there is no statistical evidence about any sense of a word, the algorithm chooses the meaning with the highest frequency rank in WordNet.

2.3 Formal Instruction Representation

With the ontological concepts resolved, the howto can be formally represented as a sequence of actions in the knowledge base:

```
(methodForAction
  (COMPLEX_TASK ARG1 ARG2 ...)
  (actionSequence
     (TheList action1 action2 ...)))
```

Each step *action1*, *action2* etc. is an instance of an action concept like *PuttingSomethingSomewhere*. Since the knowledge base contains information about required parameters for each concept, the system can detect if the specification is complete. For instance, the action *PuttingSomethingSomewhere* needs to have information about the object to be manipulated and the location where this object is to be placed.

Action parameters are created as instances of objects or spatial concepts and are linked to the action with special predicates. In the example below, the *objectActedOn* relation specifies which object the action *put1* of type *PuttingSomethingSomewhere* is to be executed on. *purposeOf-Generic* is used to describe post-conditions; in this case, the outcome of the action *put1* shall be that the object *cup1* is related to *table1* by the *on-UnderspecifiedSurface* relation.

```
(isa put1 PuttingSomethingSomewhere)
(isa table1 Table-PieceOfFurniture)
(isa cup1 DrinkingMug)
(objectActedOn put1 cup1)
(purposeOf-Generic
      put1
      (on-UnderpecifiedSurface
            cup1
            table1))
```

Time constraints are translated into *timeSpan* relations, quantifiers are modelled with the *amountOfObject* property, for example

```
(amountOfObject tablesalt1 (Teaspoon-UnitOfVolume 1 2))
(timeSpan boilingFood1 (MinutesDuration 10 12))
```

2.4 Robot Plan Generation

The instructions are to be executed by our B21 robot acting in a kitchen environment. This scenario exists both in reality and in a realistic physical simulation (Figure 4). In this paper, we assume that the robot already has plans for a set of low-level actions like picking up objects or navigating to a position inside the environment. Building such a library including all the issues like object recognition and skillful manipulation is the topic of parallel research projects as described in [1].

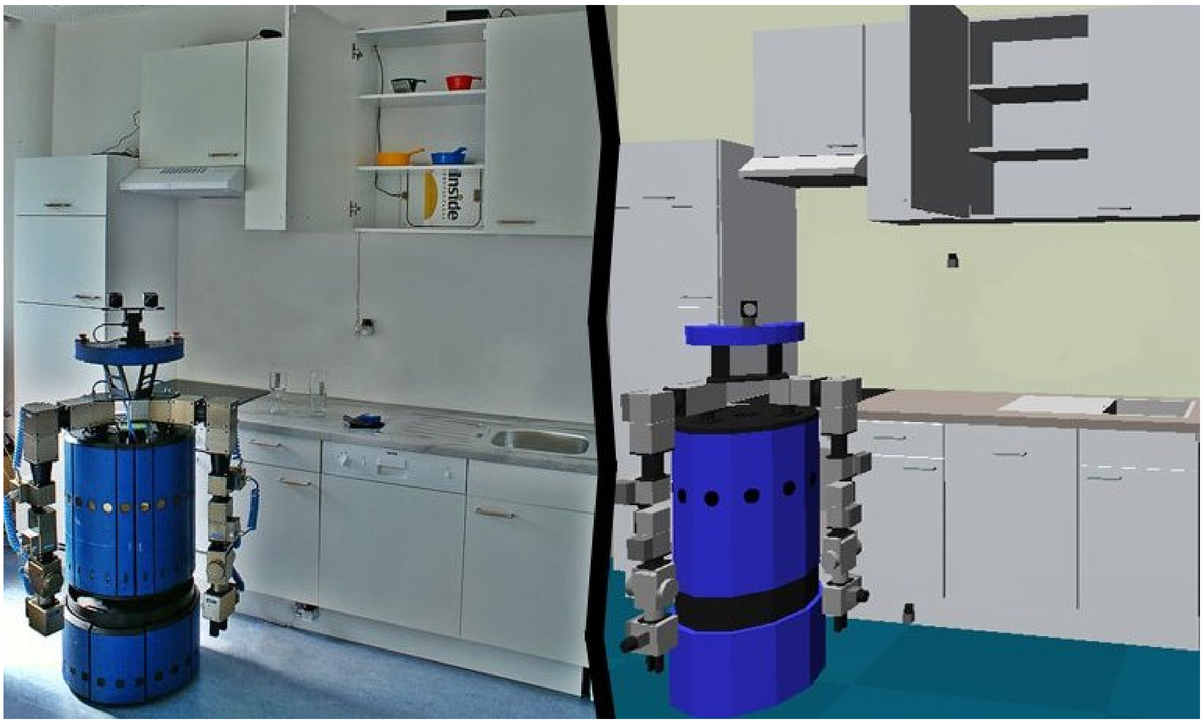


Figure 4: B21 robot in the real kitchen and in simulation.

For execution, the formal instruction representation has to be transformed into a valid robot plan. The plans for our robot are implemented in extended RPL (Reactive Plan Language) [15] which provides a clear, expressive and extensible language for writing robot plans. RPL is an interpreted language written in Lisp. Objects and locations are described by designators, qualitative specifications which are resolved during the plan execution. Instead of directly referencing a specific object in the environment, for instance by an ID *cup-23*, designators describe necessary properties a suitable object needs to have. This makes the plan more flexible because the robot can select any object that matches the specification, even if e.g. *cup-23* is not available.

The first step in resolving a designator is to match a conjunction of the required properties against the objects in the knowledge base. All resulting objects can possibly be used for the task and at least one of them has to be found in the environment. For this reason, each of the objects in the knowledge base is linked to a model from the vision system [16] that can be used to detect it in a camera image. So all models that have been returned by the query are given to the vision system in order to find a suitable object. A query for such a vision model looks like

```
(and
    (isa ?obj Cup)
    (stateOfObj ?obj Clean)
    (color ?obj Green)
    (visionModel ?obj ?model))
```

Object designators are not only grounded in the perception, but also linked to the action system. Object instances in the knowledge base are annotated with information how to manipulate them. Currently, these are links to specialized grasping routines for cups, plates, or pieces of silverware. More details about the concept of designators can be found in [6].

Each single instruction is translated into an achieve statement whose parameter list has the goal to be achieved as the first entry. Depending on the type of action, additional parameters can be specified. For each goal, there exists a plan to achieve it. Several low-level plans for goals like *entity-at-place* have already been implemented manually and are available to the system.

```
(define-high-level-plan (achieve (put1))
  (with-designators ((drinkingmug1 '(an entity
                                        (type cup)))
                     (table1 '(an entity
                                  (type table)))
                     (location1 '(a location
                                     (on ,table1))))
    (achieve (loc drinkingmug1 location1))))
```

2.5 Plan Debugging and Optimization

The plan debugging and optimization are not the main topic of this paper, but since these steps are usually necessary for obtaining working plans, we will briefly sketch the procedure and refer to the respective literature for details.

In a first step, the system executes the plan in a realistic physical simulation and records data, e.g. about the object interactions, collisions, and the times needed for each action. The debugging process then matches flaw specifications against the recorded data and, if problems are detected, infers the most probable reason [17]. An example of such problems could be that the robot collides with a chair that is standing in front of the table while trying to put items onto the table top. When such flaws are detected, the system applies transformations [6] to the plan which add parameters to object specifications, change the order of actions, or insert additional goals in order to eliminate the source of the error. In this example, a suitable fix would be to first remove the chair and put it back to its original location after having performed the desired actions.

Low performance can also be seen as a flaw which can be fixed by suitable transformations as described in [6], for example by using a container for transporting objects or by using both hands for carrying objects and thereby making better use of the robot's resources.

2.6 Inference of Missing Information

Many plan flaws are caused by incomplete action specifications: Details are often omitted since humans can easily infer them using their common sense knowledge. Some pieces of information also depend of the actual environment, like the position where an object should be put, and cannot be specified in general.

Which common sense knowledge is required heavily depends on the task at hand. We include knowledge extracted from the OpenMind Indoor Common Sense database [18] into our knowlede base, but providing sufficient common sense knowledge for a robot to correctly interpret a large set of different instructions remains an open topic.

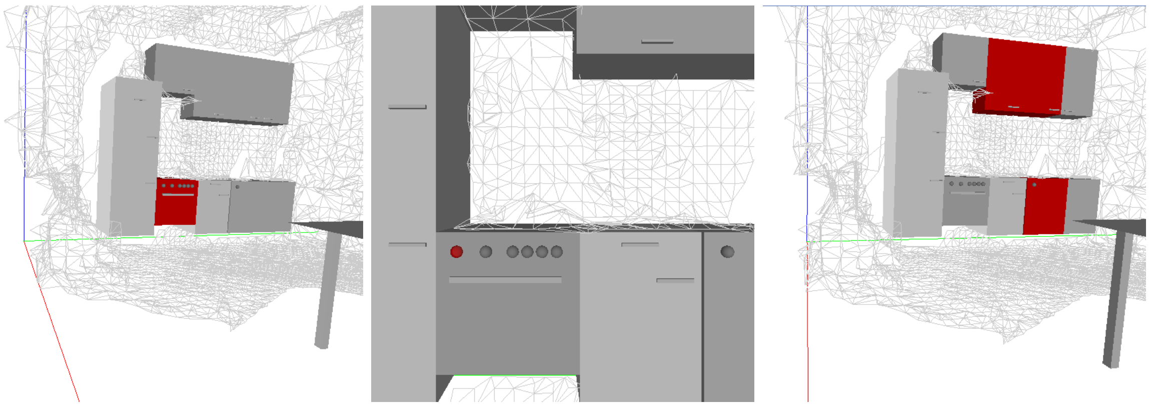


Figure 5: Visualized results of queries to the environment model represented in the knowledge base including the function or current state of objects. The images show objects that serve for cooking food (left), parts of the oven that cause a *Boiling* event (center) and objects that contain drinking vessels (right).

Environment-specific information is acquired from the environment model and from observations of previous actions. Our environment model [19] is created from 3D laser scans in which objects are detected and classified. These objects are then represented as instances of concepts in the knowledge base, e.g. *table1* as an instance of the general concept *EatingTable*, and therefore inherit all the properties of the respective concept:

```
(isa table1 EatingTable)
(heightOfObject table1 0.74)
...
(xCoordinate table1 2.31)
...
```

This complete integration of the environment model into the knowledge base allows for reasoning on general object properties (e.g. that a table can be a supporting entity for other objects) as well as environment-specific information like the position or dimensions of objects (Figure 5). Using such information, the system translates relative position specifications from the instructions into global environment coordinates.

Log data of actions performed by the robot [20] or observed from humans [21] can also be accessed from within the knowledge base. Consider the problem of inferring that a cup is to be put on the table when the instruction only states "in front of the chair". From previous observations of different tasks, the robot has log data of which objects it has put on top of which supporting objects at which position. From this information, it learns a classifier that generates rules like "if x between 0.6 and 1.8, and y is between 2.32 and 2.98, and if the object is of type tableware, the supporting entity is table1". These classifiers are learned on demand and embedded into the knowledge representation as described in [22].

These pieces of information are used for determining the most probable plan flaws and suitable bug fixes. For learning the concepts, it is sufficient to have log data of similar actions (like objects being put on the table), but the robot does not need to have seen the whole task like setting the table before.

3 Evaluation

We tested the implemented system on 88 instructions from a training set and another 64 from a test set of howtos which are taken from `ehow.com` and `wikihow.com`. Since many of the errors are caused by the syntax parser, we evaluate the system both with automatically parsed syntax trees and manually created ones in order to better show the performance of the other components. For the training set, we manually added 72 missing mappings from WordNet synsets to Cyc concepts; the test set was transformed without such manual intervention.

First, we trained the disambiguator on the training set using manually created parse trees. Afterwards, we ran the system including the syntax parser on the same set of howtos, the results are shown in the upper part of Table 1. With correct parse trees, the system achieves a recognition rate of 82% on the training set and even 91% on the test set before the ontology mapping and the transformation of the instructions into the formal representation.

	aut. parsed		**man. parsed**	
Training Set:				
Actual Instructions	**88**	**100%**	**88**	**100%**
Correctly Recognized	59	67%	72	82%
False Negative	29	33%	16	18%
False Positive	4	5%	2	2%
Test Set:				
Actual Instructions	**64**	**100%**	**64**	**100%**
Correctly Recognized	44	69%	58	91%
False Negative	20	31%	6	9%
False Positive	3	5%	6	9%

Table 1: Summary of the evaluation on instruction level

The remaining 18% resp. 9% have either been recognized incorrectly (missing object or preposition in the instruction) or not at all. The latter group also comprises instructions that are not expressed as imperative statements and are therefore not supported by the current implementation. In both test runs, errors caused by the syntax parser result in a significant decrease in the recognition rate (15 percentage points in the training set, 22 in the test set).

Table 2 shows the results of the translation into the formal instruction representation. In the training set, 70 of the 72 instructions which have been recognized in the previous step could successfully be transformed, the two errors were caused by a mapping of a word sense to a concept that cannot be instantiated as an object in Cyc.

	aut. parsed		**man. parsed**	
Training Set:				
Actual Instructions	**88**	**100%**	**88**	**100%**
Import Failures	**31**	**35%**	**18**	**20%**
Incorrectly/Not recognized	29	94%	16	89%
Missing WordNet entries	0		0	
caused Import Failures	0	0%	0	0%
Missing Cyc Mappings	0			
caused Import Failures	0	0%	0	0%
Misc. Import Errors	2	6%	2	11%
Disambiguation Errors	0		0	
Correctly imported into KB	**57**	**65%**	**70**	**80%**
Test Set:				
Actual Instructions	**64**	**100%**	**64**	**100%**
Import Failures	**33**	**52%**	**28**	**44%**
Incorrectly/not recognized	20	61%	6	21%
Missing WordNet entries	3		3	
caused Import Failures	2	6%	2	7%
Missing Cyc Mappings	14		23	
caused Import Failures	11	33%	20	71%
Misc. Import Errors	0	0%	0	0%
Disambiguation Errors	2		3	
Correctly imported into KB	**31**	**48%**	**36**	**56%**

Table 2: Summary of the evaluation on knowledge base level

The results of the translation of the test set show that two external components are the main sources of error: 40% of the import failures are caused by the syntax parser, since a decrease from 61% to 21%

of failures in the initial recognition step can be observed when switching to manually created syntax trees. In this case, missing Cyc mappings and WordNet entries are the main problem, causing about 78% of the remaining errors.

Test set of Howtos	**Instr. Level**	**KB Level**
How to Set a Table	100%	100%
How to Wash Dishes	92%	46%
How to Make a Pancake	93%	73%
How to Make Ice Coffee	88%	63%
How to Boil an Egg	78%	33%

Table 3: Per-Howto evaluation of the import procedure.

An evaluation per howto (Table 3) shows that a reasonably large number of the instructions can be recognized correctly, assuming that the coverage of important mappings from WordNet to Cyc will increase. The generation of a robot plan from the formally represented instruction is a rather simple translation from Cyc concepts to RPL statements which did not produce any further errors.

4 Discussion

The translation into a formal instruction representation suffers from two main sources of errors: Especially for longer sentences, the quality of the syntax trees generated by the Stanford parser decreases, which has a strong impact on the recognition rate. In the test set, 14 of 20 false negatives are caused by the parser.

Missing WordNet entries or missing mappings to Cyc concepts are another important issue. Only 11,000 of the 300,000 Cyc concepts are already mapped to a WordNet synset. However, this source of error will have less impact when the most relevant household items have correct mappings. Once adding a set of mappings for the most common items can significantly improve the recognition of a large number of howtos, as can be seen by the results of the training set.

While the system can resolve spatial relations and qualitative specifications based on collected experiences and the environment model, some aspects of common sense knowledge are hard to obtain, e.g. if and how tasks scale with the number of people: When setting a table, there is one plate per person, but only one soup tureen for all of them. The number of breakfast eggs is linearly related to the number of people, the amount of butter is not, but a second piece of butter may be needed if many people attend.

5 Conclusions

In this paper we present a novel approach for generating plans for mobile robots: Instead of composing plans from a set of atomic actions, we propose to generate plans by transforming natural-language task instructions from websites like ehow.com into formal, executable robot plans. These plans are much better suited to the domain of complex mobile manipulation, like for instance common household tasks, since they inherently handle issues like incomplete task specifications, unknown start and goal states or constraints to the order of actions.

We propose techniques for semantically parsing the natural-language instructions, transforming them into grounded symbolic plan representations, and generating robot plans out of this information. For inferring information missing in the instructions, we propose techniques for debugging the plan and adding required information based on a rich environment model and collected experiences.

The evaluation of our implementation shows that it is feasible to correctly transform about 80% of the instructions taken from websites. A better syntax parser and more mappings between WordNet and Cyc will help increase this number. We believe that this system is an important module for scaling mobile household robots towards task complexity by giving them the ability to autonomously extend their task repertoire.

REFERENCES

[1] M. Beetz, F. Stulp, B. Radig, J. Bandouch, N. Blodow, M. Dolha, A. Fedrizzi, D. Jain, U. Klank, I. Kresse, A. Maldonado, Z. Marton, L. Mösenlechner, F. Ruiz, R. B. Rusu, and M. Tenorth, "The assistive kitchen — a demonstration scenario for cognitive technical systems," in *IEEE 17th International Symposium on Robot and Human Interactive Communication (RO-MAN), Muenchen, Germany*, 2008, invited paper.

[2] F. Gravot, A. Haneda, K. Okada, and M. Inaba, "Cooking for humanoid robot, a task that needs symbolic and geometric reasonings," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 462–467, 2006.

[3] C. Burghart, R. Mikut, R. Stiefelhagen, T. Asfour, H. Holzapfel, P. Steinhaus, and R. Dillmann, "A cognitive architecture for a humanoid robot: A first approach," *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, pp. 357–362, 2005.

[4] M. Fox and D. Long, "Modelling mixed discrete-continuous domains for planning," *Journal of Artificial Intelligence Research*, vol. 27, pp. 235–297, 2006.

[5] M. Beetz, *Plan-based Control of Robotic Agents*, ser. Lecture Notes in Artificial Intelligence. Springer Publishers, 2002, vol. LNAI 2554.

[6] A. Müller, "Transformational planning for autonomous household robots using libraries of robust and flexible plans," Ph.D. dissertation, Technische Universität München, 2008. [Online]. Available: http://mediatum2.ub.tum.de/node?id=645588

[7] J. Zelek, "Human-robot interaction with minimal spanning natural language template for autonomous and tele-operated control," in *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1997. IROS '97.*, vol. 1, Sep 1997.

[8] S. Tellex and D. Roy, "Spatial routines for a simulated speech-controlled vehicle," in *HRI '06: Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction.* New York, NY, USA: ACM, 2006.

[9] N. Mavridis and D. Roy, "Grounded situation models for robots: Where words and percepts meet," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 4690–4697.

[10] R.J.Kate, Y. W. Wong, and R. Mooney, "Learning to Transform Natural to Formal Languages," in *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 2005, pp. 1062–1068.

[11] M. Perkowitz, M. Philipose, K. Fishkin, and D. J. Patterson, "Mining models of human activities from the web," in *WWW '04: Proceedings of the 13th international conference on World Wide Web.* ACM, 2004, pp. 573–582.

[12] D. Klein and C. D. Manning, "Accurate unlexicalized parsing," in *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics.* Morristown, NJ, USA: Association for Computational Linguistics, 2003, pp. 423–430.

[13] C. Fellbaum, *WordNet: an electronic lexical database.* MIT Press USA, 1998.

[14] C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira, "An introduction to the syntax and content of Cyc," *Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering*, pp. 44–49, 2006.

[15] D. McDermott, "A Reactive Plan Language," Yale University," Research Report YALEU/DCS/RR-864, 1991.

[16] U. Klank, M. Z. Zia, and M. Beetz, "3D Model Selection from an Internet Database for Robotic Vision," in *International Conference on Robotics and Automation (ICRA)*, 2009.

[17] M. Beetz, *Concurrent Reactive Plans: Anticipating and Forestalling Execution Failures*, ser. Lecture Notes in Artificial Intelligence. Springer Publishers, 2000, vol. LNAI 1772.

[18] R. Gupta and M. J. Kochenderfer, "Common sense data acquisition for indoor mobile robots," in *In Nineteenth National Conference on Artificial Intelligence (AAAI-04*, 2004, pp. 605–610.

[19] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, “Towards 3D Point Cloud Based Object Maps for Household Environments,” *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge)*, 2008.

[20] A. Kirsch, “Integration of programming and learning in a control language for autonomous robots performing everyday activities,” Ph.D. dissertation, Technische Universität München, 2008. [Online]. Available: http://mediatum2.ub.tum.de/node?id=625553

[21] J. Bandouch, F. Engstler, and M. Beetz, “Accurate human motion capture using an ergonomics-based anthropometric human model,” in *Proceedings of the Fifth International Conference on Articulated Motion and Deformable Objects (AMDO)*, 2008.

[22] N. v. Hoyningen-Huene, B. Kirchlechner, and M. Beetz, “GrAM: Reasoning with grounded action models by combining knowledge representation and data mining,” in *Towards Affordance-based Robot Control*, 2007.