The International Journal of
Robotics Research
2016, Vol. 35(1–3) 281–300
© The Author(s) 2015
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0278364915602060
ijr.sagepub.com

Tell me Dave: Context-sensitive grounding of natural language to manipulation instructions

Dipendra K. Misra, Jaeyong Sung, Kevin Lee and Ashutosh Saxena

Abstract
It is important for a robot to be able to interpret natural language commands given by a human. In this paper, we consider performing a sequence of mobile manipulation tasks with instructions described in natural language. Given a new environment, even a simple task such as boiling water would be performed quite differently depending on the presence, location and state of the objects. We start by collecting a dataset of task descriptions in free-form natural language and the corresponding grounded task-logs of the tasks performed in an online robot simulator. We then build a library of verb–environment instructions that represents the possible instructions for each verb in that environment, these may or may not be valid for a different environment and task context. We present a model that takes into account the variations in natural language and ambiguities in grounding them to robotic instructions with appropriate environment context and task constraints. Our model also handles incomplete or noisy natural language instructions. It is based on an energy function that encodes such properties in a form isomorphic to a conditional random field. We evaluate our model on tasks given in a robotic simulator and show that it successfully outperforms the state of the art with 61.8% accuracy. We also demonstrate a grounded robotic instruction sequence on a PR2 robot using the Learning from Demonstration approach.

Keywords
Natural language understanding, machine learning, human–robot interaction and crowd-sourcing

1. Introduction

For robots working in the presence of humans, it is important for them to be able to ground a task described in natural language for the given environment. In this paper, we present a learning algorithm that maps natural language instruction to a mobile manipulation instruction sequence depending upon the environment context.

We consider a personal household robot setting as shown in Figure 1 where the robot interacts with non-expert users only using natural language and is asked to do different sets of tasks. For a given task described in a natural language instruction, the robot must come up with a valid sequence of instructions that accomplishes the task. This is challenging for several reasons. Consider the task of boiling water shown in Figure 2, it consists of a series of steps to be performed that a user describes in natural language (NL). Each step is challenging because there are variations in the natural language, and because the environment context (i.e. the objects and their state) determines how to perform it. For example, to accomplish the instruction *"heating the water"*, one can either use a stove for heating or a microwave (if it is available). For each option, several steps of grasping, placing, turning the stove, etc. would need to be performed. In another example, the NL instruction *"fill the cup with water"* may require the robot to pick up a cup and fill it with water either from the tap (if there is one) or fill it from a fridge water-dispenser or whatever other means are available in the environment. We also note that if the cup already has water then we may need to do nothing. This mapping (or grounding) of the NL instruction into a sequence of mobile manipulation instructions thus varies significantly with the *task constraints* and the *environment context*. In this paper, our goal is to develop an algorithm for learning this grounding.

Different aspects of the language grounding problem have been explored by recent works (Beetz et al., 2011; Bollini et al., 2012; Guadarrama et al., 2013; Marco et al., 2012; Tellex et al., 2011). In particular, Guadarrama et al.

Computer Science Department, Cornell University, USA

Corresponding author:
Dipendra Misra, Computer Science Department, Cornell University, Cornell Tech, 111 8th Ave #302, New York, NY 10011, USA.
Email: dkm@cs.cornell.edu

Fig. 1. Personal household robot setting where the user describes a task in natural language and the robot infers an action plan based on the environment that accomplishes the task.

(2013) focused on using spatial relations to ground noun-phrases to objects in the environment. They used an injective mapping from verbs to controller instructions based on pre-defined templates. As we show in our experiments, pre-defined templates do not work well with the variations in NL and with changing environment and task context (see Figure 3 for an example). Beetz et al. (2011) and Marco et al. (2012) consider translating web recipe into a robot making pancakes and focus on translating the knowledge into a knowledge reasoning system. However, our problem requires data-driven retrieval of relevant pieces of instructions that are contextually-relevant for that subtask. Therefore, our work focuses on considering large variations in the NL instructions for generalizing to different tasks in changing environments. Bollini et al. (2012) showed that mapping from natural language to recipe is possible by designing a probabilistic model for mapping NL instructions to robotic instructions, and by designing an appropriate state-action space. They then perform a tree search in the action space to come up with a feasible plan. Since in our problem the search space is very large, their tree search becomes infeasible.

Natural language instructions given in a task and environment context are often incomplete since the remaining information can be inferred from the context. Malmaud et al. (2014) noticed many ambiguous and incomplete instructions in the context of interpreting cooking recipes. This pattern was found in our user study as well. Table 1 shows some natural language instructions from our dataset along with the different challenges that exist in grounding them.

One key property of our model is that we can handle missing or *incomplete NL instructions*, for which the robotic instructions have to be inferred from context. For example, the NL instruction *"heat the pot"* does not explicitly say that the pot must be placed on the stove first, and it has to be inferred from the task constraints and the environment context. Furthermore, sometimes one should not follow the NL instructions precisely and instead come up with alternatives that are suitable for the robot to perform in that particular situation.

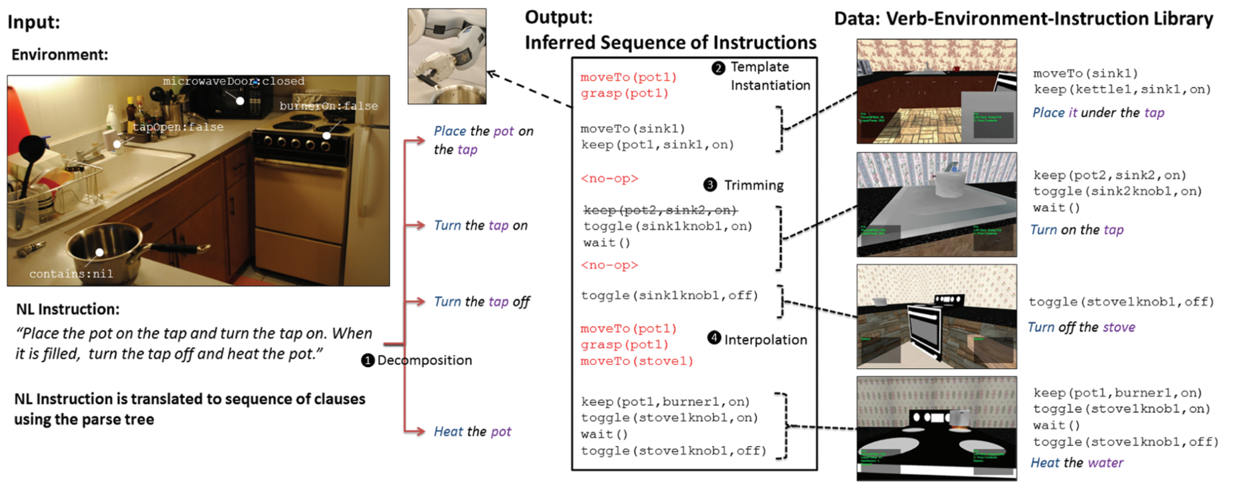
In this work, we focus on developing a method that models the variations in NL and the ambiguities in grounding them to robotic instructions given an environment context and task constraints. Our model considers the trade-off in following NL instructions as closely as possible while relying on previously-seen contextually-relevant instructions in the training dataset. In detail, we take a data-driven approach where we first collect a database of NL instructions and robotic instructions sequences performed for different tasks in an online simulated game. Using this data, we build a verb-environment-instruction library (VEIL). We then present a machine learning approach that models the relations between the language, environment states and robotic instructions. Our model is isomorphic to a conditional random field (CRF), which encodes various desired properties in the form of potential functions on the edges. With a sampling based inference algorithm, we show that our approach produces valid and meaningful instructions, even when the environment is new or the NL instructions are incomplete and not precisely valid for that environment.

We evaluate our approach on our VEIL dataset for six different tasks. Each task has five different environments with free-form natural language instructions and robotic instruction logs, collected from several users. The tasks comprise performing several steps in sequence and there are often different ways of performing the task in different environments. We compare our method against our implementation of Guadarrama et al. (2013) and Bollini et al. (2012), and show significant improvements. More importantly, we find that our method handles generalization to new environments and variations in language well, and is also able to handle incomplete NL instructions in many cases.

Finally, we train our PR2 robot using the Learning from Demonstration approach for several controller instructions and several objects. While mapping each controller instruction to an executable trajectory is a challenging task in itself and is not the main focus of this paper, our approach does provide a proof of concept, that in a constrained environment the controller instruction can be reliably mapped to a trajectory. We demonstrate this by testing a full predicted controller instruction sequence on the PR2 robot for making a dessert, following NL instructions given by a user.

In summary, the key contributions of this paper are;

- We encode the environment and task context into an energy function over a CRF which allows grounding of the NL instructions into an environment for tasks.
- Our model is able to handle missing NL instructions and free-form variations in the language.
- Our method can handle mobile manipulation tasks with long instruction sequence. Our setting has a large state space of the objects, and a large robotic action space.
- We contribute an online data collecting method, and the resulting VEIL dataset comprising free-form natural language instructions and corresponding robot instruction logs. Our experiments show good results on the dataset and our model outperforms the related work.



```
moveTo(pot1)
grasp(pot1)

moveTo(sink1)
keep(pot1,sink1,on)

<no-op>

~~keep(pot2,sink2,on)~~
toggle(sink1knob1,on)
wait()
<no-op>

toggle(sink1knob1,off)

moveTo(pot1)
grasp(pot1)
moveTo(stove1)

keep(pot1,burner1,on)
toggle(stove1knob1,on)
wait()
toggle(stove1knob1,off)
```

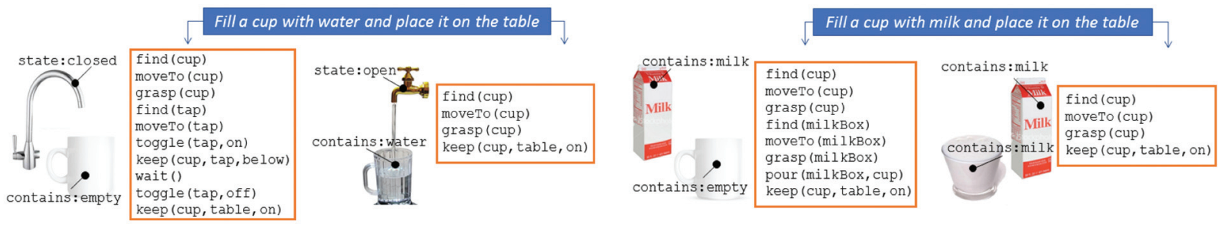
```
moveTo(sink1)
keep(kettle1,sink1,on)

keep(pot2,sink2,on)
toggle(sink2knob1,on)
wait()

toggle(stove1knob1,off)

keep(pot1,burner1,on)
toggle(stove1knob1,on)
wait()
toggle(stove1knob1,off)
```

Fig. 2. Natural language instructions to a sequence of controller instructions for a given new environment. Our approach takes a description in natural language and sequences together robotic instructions that are appropriate for a given environment and task. Note that the NL instructions are often ambiguous or incomplete.



```
find(cup)
moveTo(cup)
grasp(cup)
find(tap)
moveTo(tap)
toggle(tap,on)
keep(cup,tap,below)
wait()
toggle(tap,off)
keep(cup,table,on)
```

```
find(cup)
moveTo(cup)
grasp(cup)
keep(cup,table,on)
```

```
find(cup)
moveTo(cup)
grasp(cup)
find(milkBox)
moveTo(milkBox)
grasp(milkBox)
pour(milkBox,cup)
keep(cup,table,on)
```

```
find(cup)
moveTo(cup)
grasp(cup)
keep(cup,table,on)
```

Fig. 3. Many–many correspondence between language and controller instructions depending on the environment. The two sentences differ only in one object (water/milk) and they ground to two different or same instruction-sequences depending upon the environment. This shows that the mapping is dependent upon both the objects in the given sentence and the environment. This means that we cannot assume a fixed mapping for sentence template such as *"fill x with y"*.

The paper is organized as follows. We give a summary of related works and their limitations in the context of this problem in Section 2. We give an overview of our algorithm and the representation of input - environment and language and the output - controller instruction sequence in Section 3. We describe our model in Section 4 and the VEIL dataset format that we use for solving the model in Section 5. We provide details of the features used by our model and the inference and learning procedure in Section 6. Our crowd-sourcing system that we developed for collecting the VEIL dataset is described in Section 7. We describe our experiments and results and give details of our robot experiment in Sections 8 and 9 respectively. We describe future work and discuss the limitations of our approach in Section 10. We give the concluding remarks in Section 11.

2. Related work

In this section, we first describe related works in the field of semantic parsing, environment representation and mobile manipulation instructions. In this paper, we use several ideas and results from these fields. We then describe related work for the problem of grounding natural language.

Semantic Parsing. Semantic parsing is the problem of representing natural language by a formal representation that preserves the meaning. In spite of success in the area of syntactic parsing, semantic parsing remains an open challenging task. For application in robotics, this problem is further complicated by the presence of an environment context. This often results in incomplete and ambiguous natural language instructions which implicitly use context (environment and task objective), for resolution. As an example, consider the following sentence from our dataset: *"Microwave the coffee, scoop some ice cream, drizzle syrup on top"*. This sentence is incomplete since it does not specify that ice-cream should be added to the coffee after scooping it and that syrup needs to be drizzled on top of the coffee cup. However, these missing instructions are obvious to humans since drizzling syrup on the floor will make little sense.

In order to map NL instructions into their meaning representations, Guadarrama et al. (2013) parse the NL instructions into one of the many manually created

Table 1. Dataset examples of natural language descriptions from VEIL-300 along with the difficulty in grounding them. Our dataset shows that users may often provide minimal information in their instructions if the context of the problem is clear.

Task	Natural Language Description	Challenges
Cooking ramen	*Heat up the water, then cook the ramen.*	Ambiguous. (How to cook?)
Serving affogato	*Drop a scoop of ice cream, drizzle syrup on top, then add coffee to the bowl.*	Missing object. (Add drizzle on top of what?)
Making coffee	*Add milk, sugar and coffee to a cup. Next ignite the stove and place the cup on the stove and wait. After 5 minutes close the stove and take out the cup.*	Verb with many arguments.
Boiling water	*Microwave for 20 minutes and place the cup on the desk.*	Missing object with resolution in future. High-level verb (microwave).
Serving affogato	*Microwave the coffee, scoop some ice cream, and drizzle syrup on top.*	Incomplete (adding scoops to coffee not specified). Ambiguous. (How to fill?)
Cooking ramen	*Fill the pot with ramen and water. Microwave it.*	Anaphoric reference. (Microwave what?)

templates.This approach becomes unscalable with the increasing complexity of the task.

Recently, learning methods based on *combinatory categorial grammar (CCG)* (Steedman, 1996, 2000) have been used with success for different applications (Zettlemoyer and Collins, 2007). Matuszek et al. (2012b) use probabilistic CCG parsers to convert natural language commands to a Robot Control Language (subset of typed-lambda calculi) such as given below.

"*exit the room and go left*"
(do-sequentially (take-unique-exit) (turn-left))

Their approach does not handle the ambiguities of natural language that can only be resolved by incorporating environment in the parsing step, and it also cannot handle missing actions, incorrect or missing object references. The probabilistic CCG parsers such as UBL (Kwiatkowski et al., 2010) take a manually defined seed lexicon as input. This seed lexicon contains lexical entries which represent mapping between words and formal expressions, which is instruction sequence in our case. The accuracy of these semantic parsers is sensitive to this seed lexicon and providing it requires domain expertise. Yet another challenge is that annotating the NL instructions with their formal expressions in typed-lambda calculi is tedious and requires expert knowledge. In contrast, we show that collecting instruction sequence can be easily crowd-sourced.

In another work, Matuszek et al. (2012a) jointly model language and perception for the task of selecting a subset of objects that are described by natural language. The joint modeling gives significant improvement over separate models for language and perception. However, they do not handle the complex sequence of manipulation tasks that we address in this paper.

Some researchers have taken the direction to parse natural language into intermediate representations instead of directly parsing into formal semantic expressions. This is also the direction we take in this paper. Tellex et al. (2011) use *tree of Spatial Description Clause (SDC)* for this purpose. Others use *Linear Temporal Logic* to represent the language task which generates robot controllers which can be proven to be correct (Finucane et al., 2010; Kress-Gazit et al., 2007; Wongpiromsarn et al., 2010). However, these works focus on creating formal descriptions and creating controllers, and not on handling ambiguous NL variations or data-driven grounding into the environment and task context.

Environment representation. There has been a lot of work done in the computer vision community on representing environment. Previous works represent environment as a graph whose nodes represent objects such as cup, microwave, book, television etc. and whose edges represent object-object relationships which can be spatial relations (e.g. *near, far, left*), *is-part of*, or *is-type of* relationship etc.

Wu et al. (2014) produce a hierarchical labeling of a RGB-D scene using *is-part of* and *is-type of* relationships between objects. Aydemir et al. (2011) use spatial relations between objects to search for a specified object. These works form the ideas behind our environment representation. We use spatial relations between solid objects such as microwave, fridge and is-part of relationship between components of a given object such as fridge-door, fridge-platform, main-fridge-body etc.

Mobile manipulation tasks. The previous decade has seen significant work on different manipulation and navigational skills such as grasping (Kroemer et al., 2010; Lenz et al., 2013), mixing (Bollini et al., 2012), pushing (Srinivasa et al., 2010), placing (Barry et al., 2013; Jiang et al., 2012), constructing semantic maps (Walter et al., 2013), and high degree of freedom arm planners (e.g. Alterovitz et al. (2011); Ratliff et al. (2009)). These works form the building blocks for executing the output instructions for our model. On the other hand, rather than building specific manipulation or navigation primitives, Learning from Demonstration (LfD) approach allows even non-experts to train the robot by using the arms of the robot (Argall et al., 2009). For actual robotic experiment on a PR2 robot, since our goal is to verify the applicability of grounded robotic instruction sequences on a real robot, we

Table 2. List of low-level instructions that could be executed by the robot. Each instruction is parametrized by the required objects. We implement a subset of these instructions on a PR2 robot (see Section IX).

Instruction	Description
`find(obj)`	Find obj in the environment (Anand et al., 2012).
`keep(obj1,obj2,R)`	Keeps obj1 with respect to obj2 such that relation R holds (Jiang et al., 2012).
`grasp(obj)`	Grasp obj (Lenz et al., 2013).
`release(obj)`	Releases obj by opening gripper.
`moveTo(obj)`	Move to obj by using motion planner (Ratliff et al., 2009).
`press(obj)`	Presses obj using end effector force controller (Bollini et al., 2011).
`turn(obj,angle)`	Turns the obj by a certain angle.
`open(obj)`	Opens the obj (Endres et al., 2013).
`close(obj)`	Closes the obj (Endres et al., 2013).
`scoopFrom(obj1,obj2)`	Takes scoop from obj2 into obj1.
`scoopTo(obj1,obj2)`	Puts the scoop from obj1 into obj2.
`squeeze(obj1,obj2,rel)`	Squeezes obj1 satisfying relation rel with respect to obj2.
`wait()`	Wait for some particular time.

take LfD-based approach where we train the robot via our teleoperation method, among many other LfD alternatives.

Traditionally, symbolic planners (Rintanen, 2012) have been used to accomplish sequencing complicated controller instructions. Since real environments have uncertainty and non-determinism, Kaelbling and Lozano-Pérez (2011) start with an abstract plan and recursively generate plans as needed. Or, the tasks are defined through expert designed state machines (Nguyen et al., 2013), which do not generalize well when the environment or the task changes. Rather than relying on symbolic representation of the environment, Sung et al. (2014) rely on a set of visual attributes to represent each object in the environment and dynamically choose the controller sequence from a list of possible sequences that minimize the score function based on the current environment and the potential candidate for the next instruction. Others use demonstrations for learning different behaviors (Niekum et al., 2013). These approaches solve only parts of the problem that we address in this work, of creating valid plans and using a score function for data-driven retrieval of instruction sequence. Our work addresses not only the validity of instruction sequences and data-driven retrieval of low-level instructions, but it also models the ambiguity and grounding of natural language instructions in the environment context. Furthermore, the tasks considered by our work are complex manipulation tasks requiring long instruction sequences.

Grounding Natural Language. Several recent works in robotics have looked at the problem of grounding natural language. This has been driven by advances in HRI, better vision algorithms and more reliable manipulation instructions. Other than the works discussed in the introduction (Beetz et al., 2011; Bollini et al., 2012; Guadarrama et al., 2013; Marco et al., 2012), the problem of navigation has been addressed by using learned models for verbs such as *follow, meet, go*, and dynamic spatial relations such as *walk close to the wall* (Fasola and Matarić, 2013; Kollar et al., 2010). To detail, Kollar et al. (2010) use a maximum-likelihood approach to infer the path taken by the robot. Translation of such weakly specified actions into robotic behaviors is very important and forms the ideas for our robotic instruction set in Table 2. Artzi and Zettlemoyer (2013) focus on mapping natural language instructions representing navigational tasks, to a sequence of instructions for a 2D world scenario. In contrast to these works, the chief linguistic objects of our investigation are high-level verbs such as *serve, cook, throw* which have more complex representation than simple navigational verbs. In contrast to Artzi and Zettlemoyer (2013), we also consider more complex 3D scenarios with a larger set of manipulation instructions.

Several works (Fasola and Matarić, 2013; Guadarrama et al., 2013) have looked at the problem of grounding intricate noun-phrases in the language to the objects in the environment. Guadarrama et al. (2014) ground open vocabulary descriptions of objects to the described objects in the environment. There has also been success in grounding concepts (Chao et al., 2011) and objects (Lemaignan et al., 2012; Ros et al., 2010) through human-robot interaction. Works like Chu et al. (2013) look at mapping from text to haptic signals. Kulick et al. (2013) consider active learning for teaching robot to ground relational symbols. The inverse problem of generating natural language queries/commentaries has also seen increasing interest in robotics (Chen et al., 2010; Tellex et al., 2013). In a recent work, Duvallet et al. (2014) explored the direction of using natural language as a sensor to come up with prior distribution over unseen regions in the environment.

In the area of computer vision, some works have considered relating phrases and attributes to images and videos (Farhadi et al., 2010b; Jiang et al., 2013; Koppula et al., 2011; Koppula and Saxena, 2013; Ramanathan et al., 2013; Wu et al., 2014). These works focus primarily on labeling the image/video by modeling the rich perceptual data rather than modeling the relations in the language and the entities in the environment. Thus, our work complements these works.

In NLP community a lot of literature exists on parsing natural language sentences (e.g. Klein and Manning (2003)) and grounding text in different domains such as linking events in a news archive and mapping language to database queries (Berant et al., 2013; Nothman et al., 2012; Poon, 2013; Yu and Siskind, 2013). These techniques form the basis of ours in syntactic parsing and representation. However most of these works use only text data, and do not address grounding the physical task or the environment. In another work, Branavan et al. (2010) consider mapping natural language queries to a sequence of GUI action commands for the windows operating system. They also consider incomplete instructions and learn environment model but they do not consider the real world 3D environments that robots encounter nor the complex controller instructions that a robot can perform.

3. Overview

In this section, we give an overview of our approach and the representation we use for natural language, environment, mobile manipulation instructions and the domain knowledge.

Given an environment E containing objects and the robot, a human gives the instructions for performing a task in natural language (see Figure 2). The instructions in natural language L consist of a set of sentences, and our goal is to output a sequence of controller instructions $\mathcal{I}$ that the robot can execute. Each of these low-level robot instructions often have arguments, e.g. `grasp(object_i)`

$$E, L \rightarrow \mathcal{I}$$

This mapping is hard to learn for two reasons: (a) the output space $\mathcal{I}$ is extremely large; and (b) the mapping changes significantly depending on the task context and the environment.

For a given E and L, certain instructions $\mathcal{I}$ are more likely than others, and we capture this likelihood by using an energy function $\Gamma(\mathcal{I}|E, L)$. We will use this energy function to encode desired properties and constraints in our problem. Once having learned this energy function, for a given new language and environment, we can simply minimize this energy function to obtain the optimal sequence of instructions

$$\mathcal{I}^* = \text{argmin}_{\mathcal{I}} \Gamma(\mathcal{I}|E, L)$$

There are several steps that we need to take in order to define this energy function. We need to convert the language L into a set of verb clauses $\mathcal{C}$, we need to represent the environment E with a usable representation that contains information about the objects, we need to describe what the low-level instructions $\mathcal{I}$ are and how do they connect to the actual execution on the robot, and we need to figure out how to represent and store the training data for their use in inference.

3.1. Language representations by a set of verb clauses

We follow previous work (Tellex et al., 2011) in parsing natural language commands into a structured intermediate representation. These structured representations only contain information relevant to the manipulation tasks and are easier to work with. We use a sequence of *verb clauses* as the representation of natural language which are ordered temporally. Each verb clause informally represents an atomic natural language command dictated by the main verb; more formally, a verb clause $\mathcal{C}$ is a tuple

$$\mathcal{C} = (\nu, [obj], \rho)$$

containing the verb ν, the set of *language-objects* [*obj*] on which it acts and a relationship matrix ρ: $obj \times obj \rightarrow Rel$ where *Rel* is the space of relationship (e.g. “with”, “from”). For example, the following natural language command comprises of four verb clauses

$$\underbrace{\textit{Take the cup with water and}}_{(\texttt{take}, [\texttt{cup}, \texttt{water}], \texttt{with}:\texttt{cup}\rightarrow\texttt{water})} \; \underbrace{\textit{then ignite the stove.}}_{(\texttt{ignite}, [\texttt{stove}], \emptyset)}$$

$$\underbrace{\textit{Now place the cup on the stove}}_{(\texttt{place}, [\texttt{cup}, \texttt{stove}], \texttt{on}:\texttt{cup}\rightarrow\texttt{stove})} \; \underbrace{\textit{and wait.}}_{(\texttt{wait}, \emptyset, \emptyset)}$$

In order to convert unconstrained natural language L to a sequence of verb-clauses $\{\mathcal{C}\}$ we use the following steps

- First, the constituency parse tree of L is found using the Stanford Parser (Klein and Manning, 2003).
- A verb clause $\mathcal{C}$ is created for each corresponding node of verb type in the parse tree. The clauses are ordered by the order of their respective verb in L. The verbs are also stemmed before being initialized as ν.
- Next, we compute the set of all *maximal nouns*, which are noun-type nodes in the parse tree whose parents are not noun-type nodes e.g. *“get the red cup”* has the node [NP: *the red cup*] as the only maximal noun.
- These *maximal nouns* are attached to the clause whose verb node is closest in the parse tree. In the ambiguous case, the nodes get attached to the nearest immediate left clause.
- Finally, all relationship-type nodes between *maximal nouns* of the same clause are inserted in the corresponding relationship matrix ρ.

Note that *language-objects* are not the same as physical objects in an environment and not all *language-objects* (e.g. water) represent an actual physical object.

3.2. Object and environment representation using graphs

For performing a task in the environment, a robot would need to know not only the physical location of the objects (e.g. their 3D coordinates and orientation), but also their functions. These functions are often represented as symbolic attributes (Anand et al., 2012; Farhadi et al., 2010a) or using a more functional representation (Cakmak et al., 2007; Höfer et al., 2014; Tenorth et al., 2010). For example, a microwave consists of four parts: main body; door; buttons; and display screen. Each part has its own states (e.g. a door could be open/closed), and sometimes its state is affected by another part, e.g. a button when pressed could unlatch the microwave door. We represent each object as a directed-graph G where the vertices are parts of the object (also storing their states), and edges represent the functional dependency between two object parts. Now for a given environment, we define E to store aforementioned representation of every object in the environment along with spatial relations between objects. Here we consider five basic spatial relations: Grasping; Inside; On; Below; and Near.

3.3. Representing robotic instructions

We have defined a set of low-level instructions that could be executed by the robot in Table 2. Each controller instruction is specified by its name and its parameters (if any). For example, `moveTo(obj)` is an instruction which tells the robot to move close to the specified object `obj`. We ground the parameters in objects instead of operation-space constants such as "2m North" because it is objects that are relevant to the task (Cakmak et al., 2007).

In order to completely execute a subtask such as *"keeping a cup on stove"*, we need a sequence of instructions $\mathcal{I}$. An example of such a sequence for the subtask might be as follows

```
find(cup1); moveTo(cup1); grap(cup1);
find(stove1);
moveTo(stove1); keep(cup1, stoveBurner2, on)
```

Note that the space of possible sequence of instructions $\mathcal{I}$ is extremely large, because of the number of possible permutations of the instructions and the arguments.

3.4. Representing domain knowledge

In this paper, the domain knowledge for a given set of environment states and a set of controller instruction primitives refer to the set of preconditions under which an instruction can be executed in an environment and its effect on the environment. In a more general setting, it can also include post-conditions and transition probabilities between two environments. The domain knowledge is provided as input to our algorithm.

Table 3. List of types of STRIPS predicate used to represent precondition and effect of controller instruction primitives which are listed in Table 2.

STRIPS predicate	Truth condition
`state obj st`	Boolean state *st* of object *obj* has value *True*
`affordance_ type obj`	object *obj* has the given affordance
`relationship_ type obj1 obj2`	Given relationship exists between object *obj*1 and *obj*2

We use the STRIPS formal language (Fikes and Nilsson, 1972) to write the preconditions and effect of each instruction listed in Table 2. To make this representation scalable for a large set of objects, we consider three types of predicates as listed in Table 3. We consider common affordances such as, *graspable, turnable, pourable etc*. and five basic spatial relationships as described earlier. Our set of object affordance is inspired by previous work (Kjellström et al., 2011; Koppula et al., 2013; Montesano et al., 2008).

This domain knowledge encodes high-level symbolic information about the world such as, a cup is filled with water if kept below an open tap. For example, this information can be encoded by the following STRIPS function

(***:action*** *keep* ***:parameters*** *(obj*1 *obj*2*R)*
:precondition *(and (grasping Robot obj*1*) (near Robot obj*2*))*
:effect *(when (and (= obj*2 *sink) (= R on) (containable obj*1*))*
*(and (when (state sink TapIsOn) (state obj*1 *Water))*
*(on obj*1 *sink)))*

This function describes an instruction primitive *keep* which accepts three arguments, object *obj*1, *obj*2 and relationship *R*. The second line defines the precondition for this instruction, which evaluates to true only when the robot is grasping the object *obj*1 and is near the object *obj*2. The third line defines the effect of this instruction, which says that when the object *obj*2 is *sink*, the relationship *R* is *on* and the object *obj*1 has the *containable* affordance, then fill the object *obj*1 with water if the sink is open and also place the object *obj*1 on the sink.

Encoding this information for an environment is very tedious and there is a large body of literature on learning them from observations (Mourao et al., 2012). While we use this domain knowledge in our main model we also show that in the absence of this knowledge, the accuracy of our algorithm does not reduce significantly.

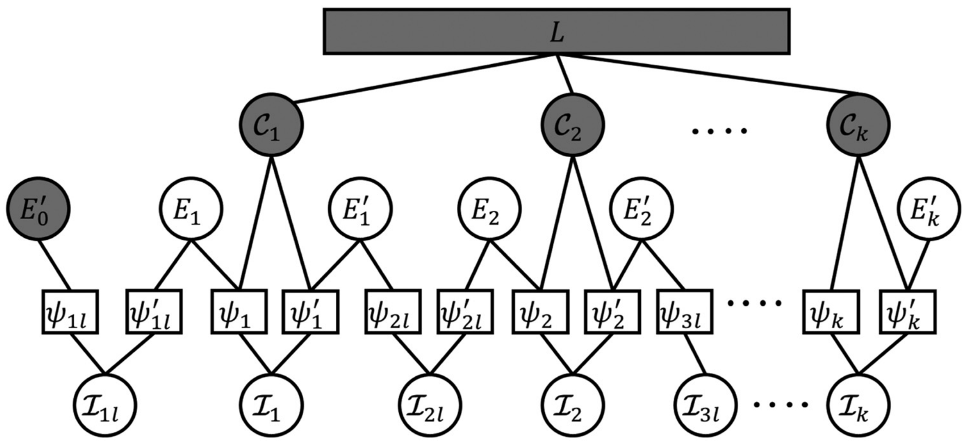


Fig. 4. Graphical model representation of our energy function. The clauses $\mathcal{C}$ and initial environment E'_0 are given, and we have to infer the instructions $\mathcal{I}_i, \mathcal{I}_{i\ell}$ and the environment at other time steps. The ψ's are the factors in the energy function.

4. Model

In this section we describe our energy function $\Gamma(\mathcal{I}|L, E)$, which is isomorphic to a conditional random field (Figure 4) and is composed of several nodes and factors (ψ).

It has the following observed nodes: natural language instruction L which is deterministically decomposed into a sequence of verb clauses $\mathcal{C}$ and the initial environment E'_0. As the instructions are executed, the state of the environment changes, and we use E_i, E'_i to represent the environment-state at step i. Our goal is to infer the sequence of controller instructions $\mathcal{I} = \{\mathcal{I}_{i\ell}, \mathcal{I}_i\}_{i=1}^{k}$.

Note that for each clause $\mathcal{C}_i$, we have two instruction sequences $\mathcal{I}_i$ and $\mathcal{I}_{i\ell}$, this is because natural language instructions are often incomplete and $\mathcal{I}_{i\ell}$ represents this missing instruction sequence. For example, if the natural language instruction is *place cup in microwave*, then one may need to open the microwave door first ($\mathcal{I}_{i\ell}$) and then place the cup within ($\mathcal{I}_i$).

Following the independence assumptions encoded in the graphical model in Figure 4, the energy function is written as a sum of the factor functions

$$\Gamma(\mathcal{I}|E,\mathcal{C}) = \sum_{i=1}^{k} \psi_i(\mathcal{I}_i, \mathcal{C}_i, E_i) + \psi'_i(\mathcal{I}_i, \mathcal{C}_i, E'_i) + \psi_{i\ell}(\mathcal{I}_{i\ell}, E'_{i-1}) + \psi'_{i\ell}(\mathcal{I}_{i\ell}, E_i)$$

We encode several desired properties into these factor functions (we describe these in detail in Section 6);

- The sequence of instructions should be *valid*. Intuitively $\psi_i(\cdot)$ represents the precondition satisfaction score and $\psi'_i(\cdot)$ represents the post-condition satisfaction score.
- The output instructions should follow the natural language instructions as closely as possible. Thus, $\psi_i(\cdot)$ and $\psi'_i(\cdot)$ depend on the clause $\mathcal{C}_i$.
- The length of the instruction set. Shorter instruction sequences are preferred for doing the same task.
- Prior probabilities. An instruction sequence with too many unlikely instructions is undesirable.
- Ability to handle missing natural language instructions. The $\psi_{i\ell}(\cdot)$ and $\psi'_{i\ell}(\cdot)$ represent the potential function for the missing instruction set, and they do not depend on any clause.

For each verb clause $\mathcal{C}_i$, the first step is to sample a few sequences of instructions $\mathcal{I}_i^{(s)}$ from the training data. With the sample of instructions for each clause obtained, we will then run our inference procedure to minimize the energy function to give a complete set of instructions satisfying the aforementioned properties. We first describe how we obtain these samples by building a verb-environment-instruction library (VEIL) from the training data. This library plays the role of a lexicon in our grounding approach.

5. Verb-environment-instruction library (VEIL)

Space of all possible instruction sequences is countably infinite, and even when the length of sequence is limited and invalid instruction sequences are pruned, the sample space still remains very large. Moreover the instruction sequence structure is often shared between different tasks such as for example between the NL commands *"fill a cup with coke"* and *"fill a mug with beer"*. This motivates the sampling based approach that we take.

For a given verb clause $\mathcal{C}_i$, we use the training data $\mathcal{D}$ to come up with a sample of the instruction-templates $\{\mathcal{I}_i^{(s)}\}$. During training we collect a large set of verb clauses $\mathcal{C}^{(j)}$, the corresponding instruction-sequence $\mathcal{I}^{(j)}$ and the environment $E^{(j)}$. The parameters of the instructions in the training dataset are specific values, such as `cup01`, and that particular object may not be available in the new test environment. Therefore, we replace the parameters in the training instructions with generalized variables *Z*. For example, `{ moveTo(cup01); grasp(bottle01)}` would be stored as *generalized* instruction sequence `{ moveTo(`z_1`); grasp(`z_2`)}`. The original mapping `{ `$z_1 \rightarrow$` cup01, `$z_2 \rightarrow$` bottle01}` is stored in $\xi^{(j)}$.

In order to come up with proposal templates for a given clause $\mathcal{C}_i$, we return all entries containing the corresponding verb clause, environment, instruction and the grounding: $\mathcal{D}^{(i)} = \{(\mathcal{C}^{(j)}, E^{(j)}, \mathcal{I}^{(j)}, \xi^{(j)}) \mid \nu(\mathcal{C}^{(j)}) = \nu(\mathcal{C}_i)\}$ for $j = 1, \ldots, |\mathcal{D}|$ in the dataset. The information in the particular s^{th} sample is represented as $D_s = (\mathcal{C}_s, E_s, \mathcal{I}_s, \xi_s)$. Note that we capture both the language context *C* and the environment context *E* in the structure of these samples.

5.1. Instantiation algorithm

Since we only store generalized instructions (with actual objects replaced by general variables *Z*), we need a way to instantiate the generalized instructions for a new environment and language context.

Given the s^{th} instruction-template $D_s = (\mathcal{C}_s, E_s, I_s, \xi_s)$, a clause $\mathcal{C}_j$ and a new environment E_j, the aim of the instantiation algorithm is to return the instantiated instruction template $\mathcal{I}_j$. In order to accomplish this task, we should find a grounding of the generalized variables *Z*(*s*) to specific objects in the environment E_j. In order to do so, we first define the function *Ground*(*a, E*) which takes a language-object *a* such as "*mug*", "*stove*", etc. and an environment *E* and returns the object in *E* which is being described by the language-object. We use a simple syntax based similarity between language-object *a* and categories of the environment objects to perform this grounding, e.g. the language object *"milk"* has high syntactical similarity with the object *milkBox* but not with *mug*. If no suitable match is found, such as for non-physical language-objects like water or objects which are mentioned in the language but not actually present, the function returns *Null*.

We then take the following rule based approach;

1) We first map to the objects that have a relation. For example, if we have `cup on table` in the clause and the template contains z_1`on` z_2, then we map z_1, z_2 to the grounding of cup and table respectively. More formally, $\forall z_1, z_2 \in obj(\mathcal{C}_s)$, $\forall a, b \in obj(\mathcal{C}_j)$ such that $\rho(\mathcal{C}_s)[z_1, z_2] = \rho(\mathcal{C}_j)[a, b]$ we map variable $z_1 \mapsto$ *Ground*(*a*, E_j) and $z_2 \mapsto$ *Ground*(*b*, E_j).
2) For an unmapped variable *z*, we map it in the same way as it was mapped in the sample *s*. More formally, $\forall z_1 \in obj(\mathcal{C}_s), a \in obj(\mathcal{C}_j)$ such that the object $\xi_s(z_1)$ has the same category as *Ground*(*a*, E_j), we map $z_1 \mapsto$ *Ground*(*a*, E_j).
3) For every remaining unmapped variable *z* in *Z*(*s*), we map it to the object in E_j that has the most common state-value pairs with the object $\xi_s(z)$.

Each rule only maps the variables which are not already mapped to an object. Also the mapping $z \mapsto$ *Ground*(*a, E*) is performed only when *Ground*(*a, E*) is not *Null*. Note that the last rule will always map every remaining unmapped variable.

The new mapping is stored as ξ and the instructions returned after replacing the variables using ξ are given by $\mathcal{I}_j$. We further define the predicate *replace* such that *replace*$(\mathcal{I}_s, Z(s), \xi)$ will return the instantiated instruction sequence $\mathcal{I}_j$ after replacing all variables $z \in Z(s)$ with the object $\xi(z)$.

6. Energy function and inference

Now for each clause $\mathcal{C}_i$, we have obtained a sample instruction set $\mathcal{I}_i^{(s)}$ (together with the clause and environment data in $\mathcal{D}_s$). We now need to come up with a full sequence $\{\mathcal{I}_{i\ell}^*, \mathcal{I}_i^*\}$ based on these initial samples. Note that we only have samples for $\mathcal{I}_i$ and not for $\mathcal{I}_{i\ell}$, which are for handling the missing natural language instructions. Furthermore, the sample instruction set is not consistent and valid, and also does not apply directly to the current environment. Based on these samples, we need to optimize and infer a sequence that minimizes the energy function.

In the following subsections, we now describe the different terms in the factor functions that encode the desired properties aforementioned.

6.1. Term $\psi_i(\mathcal{I}_i, \mathcal{C}_i, E_i; w)$

This term consists of features that encapsulate properties such as pre-condition score, instruction length, prior probabilities, etc. For a given sample $D_s = (\mathcal{C}_s, E_s, \mathcal{I}_s, \xi_s)$, we define the cost of setting $\mathcal{I}_i := replace(\mathcal{I}_s, Z, \xi)$ as

$$\begin{aligned}\psi_i(\mathcal{I}_i, \mathcal{C}_i, E_i | D_s, \xi; w) = w^T \\ [\Delta_{env}(D_s, \xi); \quad & \Delta_{nl}(\mathcal{C}_s, \mathcal{C}_i); \quad \Delta_{sim}(Z(s), \xi_s, \xi); \\ \Delta_{pcc}(\mathcal{C}_i, \mathcal{C}_s); \quad & \Delta_{jmp}(\mathcal{I}_i, E_i); \quad \Delta_{dscpl}(\mathcal{I}_i); \\ \Delta_{inpr}(\mathcal{I}_i); \quad & \Delta_{para}(\mathcal{I}_i); \quad \Delta_{trim}(\mathcal{I}_i, \mathcal{I}_s)]\end{aligned}$$

We describe each term in the following.

6.1.1. Environmental distance $\Delta_{env}(D_s, \xi)$. It is more likely to have instructions that were made in similar environments in the training data as compared to the test environment. Hence, if a variable $z \in Z(s)$ is mapped to a cup in the new environment and was mapped to a pot in the original environment, then we prefer the template D_s if cup and pot have similar state-value pairs (e.g. both are empty). We encode it as the average difference between states of objects $\xi_s(z)$ and $\xi(z)$ and for all $z \in Z(s)$. We represent the union of the states of $\xi_s(z)$ and $\xi(z)$ by $T(z)$

$$\Delta_{env}(D_s, \xi) = \frac{1}{|Z(s)|} \sum_{z \in Z(s)} \frac{1}{|T(z)|} \sum_{t \in T(z)} \mathbf{1}(\xi(z)[t] \neq \xi_s(z)[t])$$

Algorithm 1. We use dynamic programming to minimize the energy function.

```
1 global $\mathcal{D}, \mathcal{C}, \alpha$
2 function Forward-Inference($j$)
3 for each $E'_{j-1}$ such that $\alpha_{j-1}[E'_{j-1}]$ is defined do
4   for $D_s \in \mathcal{D}^{(j)}$ do                    // iterate over all VEIL templates with the same verb as $\nu(\mathcal{C})$
5     $\mathcal{I} \leftarrow instantiate(\mathcal{I}_s, \mathcal{C}_s, \mathcal{C}_j, E'_{j-1})$   // instantiate the VEIL template using the test clause and the environment
6     for $t \in [0..|\mathcal{I}|]$ do
7       $\mathcal{I}_j = \mathcal{I}[t \cdots]$                    // trim the instantiated sequence as a check for noise
8       $cstr = getContraints(\mathcal{I}_j, E'_{j-1})$   // find constraints needed to execute $\mathcal{I}_t$ by looking at STRIPS preconditions
9       $\mathcal{I}_{j\ell} = callSymbolicPlanner(E'_{j-1}, cstr)$          /* Call the planner to find an instruction sequence such
10                                                         that the resultant environment $E_j$ satisfies the constraints */
11      $E_j = \Phi(E'_{j-1}, \mathcal{I}_{j\ell})$
12      $E_{j'} = \Phi(E_j, \mathcal{I}_j)$
13      $\alpha_j[E'_j] = min\{\alpha_j[E'_j], \alpha_{j-1}[E'_{j-1}] + \psi_j(\mathcal{I}_j, \mathcal{C}_j, E_j) + \psi'_j(\mathcal{I}_j, \mathcal{C}_j, E'_j) + \psi_{j\ell}(\mathcal{I}_{j\ell}, E'_{j-1}) + \psi'_{j\ell}(\mathcal{I}_{j\ell}, E_j)\}$
```

6.1.2. Natural language similarity $\Delta_{nl}(\mathcal{C}_s, \mathcal{C}_i)$. We prefer to use those instruction templates whose verb clause was similar to the test case. Therefore, we measure the unordered similarity between the language-objects of $\mathcal{C}_s$ and $\mathcal{C}_i$ by computing their Jaccard index

$$\Delta_{nl}(\mathcal{C}_s, \mathcal{C}_i) = \frac{|obj(\mathcal{C}_s) \cap obj(\mathcal{C}_i)|}{|obj(\mathcal{C}_s) \cup obj(\mathcal{C}_i)|}$$

6.1.3. Parameter similarity cost $\Delta_{sim}(Z(s), \xi_s, \xi)$. We want the objects in the instantiated sequence to be similar (i.e. have the same category) to the one in the training set. Therefore, we define

$$\Delta_{sim}(Z(s), \xi_s, \xi) = \frac{1}{|Z(s)|} \sum_{z \in Z(s)} \mathbf{1}\{\xi_s(z) \neq \xi(z)\}$$

6.1.4. Parameter cardinality cost $\Delta_{cvr}(\mathcal{C}_i, \mathcal{C}_s)$. Different clauses with the same verb can have a different number of *language-objects*. For example, the sentences *"add ramen to the crockpot"* and *"add milk and sugar to the mug"* have a different number of language objects (2 and 3 respectively). We thus, prefer to use the template which has the same number of language objects as the given clause

$$\Delta_{pcc}(\mathcal{C}_i, \mathcal{C}_s) = \mathbf{1}(|obj(C_i)| \neq |obj(C_s)|)$$

6.1.5. Jump distance $\Delta_{jmp}(\mathcal{I}_i, E_i)$. The jump distance is a boolean feature which is 0 if program $\mathcal{I}_i$ can be executed by the robot given the starting environment E_i, and 1 otherwise.

6.1.6. Description length $\Delta_{dscpl}(\mathcal{I}_i)$. Other things remaining constant, we believe a smaller sequence is preferred. Therefore, we compute the sum of norms of each instruction in the sequence $\mathcal{I}_i$, where we define the norm of an instruction I as the number of parameters that I takes plus 1.

6.1.7. Instruction prior $\Delta_{inpr}(\mathcal{I}_i)$. We want our algorithm to give preference to instruction sequences which are more likely to occur. This is particularly useful while dealing with ambiguous or incomplete sentences. For example, the instruction `keep(book, shelf, on)` has higher prior probability than `keep(book, fridge, inside)`, although the later is not restricted. Thus, if we have an incomplete sentence *"keep the book"* then the grounded instruction sequence will be preferred if it's keeping the book on a shelf rather than inside a fridge. We therefore add the average of prior probability *prior*(I) of every instruction I in the sequence. We compute it by counting the number of times it appears in the training set

$$\Delta_{inpr}(\mathcal{I}_i) = \frac{1}{||\mathcal{I}_i||} \sum_{I \in \mathcal{I}_i} prior(I)$$

6.1.8. Parameter instruction prior $\Delta_{para}(\mathcal{I}_i)$. Here we add the prior probability of how often a parameter (e.g. `fridge`) is used for a particular instruction (e.g. `grasp`). We compute it from the training data. The verb-correlation score is the average taken over all instructions in $\mathcal{I}_i$ of the probabilities that the grounded parameter appears in the instruction at the specified position

$$\Delta_{para}(\mathcal{I}_i) = \frac{1}{|\mathcal{I}_i|} \sum_{I \in \mathcal{I}_i} \frac{1}{|d_I|} \sum_{j \in [1, |d_I|]} P(n_I, d_I[j], j)$$

Here $P(n_I, d_I[j], j)$ is the prior probability that the parameter $d_I[j]$ appears at position j for the instruction primitive n_I.

6.1.9. Trimming cost $\Delta_{trim}(\mathcal{I}_i, \mathcal{I}_s)$. Often we do not use the full sequence of instructions from the set D_s but trim them a little bit. This removes irrelevant instruction belonging to neighboring clauses in the training corpus. We define this trimming cost: $\Delta_{trim} = (|\mathcal{I}_s| - |\mathcal{I}_i|)^2$.

6.2. Term $\psi'_i(\mathcal{I}_i, \mathcal{C}_i, E'_i; w)$

This term consists of a single consistency term, given as

$$\psi'_i(\mathcal{I}_i, \mathcal{C}_i, E'_i; w) = w_{cons}\Delta_{cons}(E'_i, \mathcal{C}_i)$$

The purpose of this consistency score $\Delta_{cons}(E'_i, \mathcal{C}_i)$ is to capture the fact that at the end of execution of $\mathcal{I}_i$, the resultant environment E'_i should have fulfilled the semantics of the clause $\mathcal{C}_i$. Thus if the clause intends to *ignite the stove* then the stove should be in *on* state in the environment E'_i. For this we compute the probability table *P*(*obj, s, v*) from training data using only those data points that have the same verb as $\mathcal{C}_i$, which gives the probability that *obj* in clause $\mathcal{C}_i$ can have state *s* with value *v*. We use this table to find the average probability that objects of clause C'_i have the given end state values in E'_i.

6.3. Term $\psi_{i\ell}(\mathcal{I}_{i\ell}, E'_{i-1}; w)$

This term is for the instruction sequence $\mathcal{I}_\ell$ that does not correspond to a NL instruction, i.e. its purpose is to handle missing NL instructions. Therefore, it consists of a subset of features used in ψ_i

$$\psi_{i\ell}(\mathcal{I}_{i\ell}, E'_{i-1}; w) = w^T \begin{bmatrix} \Delta_{jmp}(\mathcal{I}_{i\ell}, E'_i); & \Delta_{dscpl}(\mathcal{I}_{i\ell}); \\ \Delta_{inpr}(\mathcal{I}_{i\ell}); & \Delta_{para}(\mathcal{I}_{i\ell}) \end{bmatrix}$$

6.4. Term $\psi'_{il}(\mathcal{I}_{il}, E_i; w)$

This consists of a single consistency term

$$\psi'_i(\mathcal{I}_{il}, E_i; w) = w_{cons,l}\Delta_{cons,l}(E'_i)$$

This consistency term is defined similarly to Δ_{cons} however since we do not have a given clause, we therefore build the table *P*(*obj, s, v*) using all the data points in the training dataset. This term prevents the robot from performing an action which takes it into an unlikely environment.

6.5. Inference procedure

Given the training dataset $\mathcal{D}$, a sequence of clauses $\{C\}$ and the starting environment E'_0, the goal is to find the instruction sequence that minimizes the energy function. Since the structure of our model is a chain, we use an approach similar to the forward-backward inference to obtain the $\{\mathcal{I}^*_{i\ell}, \mathcal{I}^*_i\}_{i=1}^k$.

The inference procedure computes the forward variable $\alpha_j[E'_j]$ which stores the cost of the minimum-cost assignment to the node $\{\mathcal{I}_{il}, \mathcal{I}_i\}_{i\leq j}$ such that the environment at chain depth j is E'_j. As a base case we have $\alpha_0[E'_0] = 0$.

We also define the environment simulator Φ as taking an environment E and an instruction sequence $\mathcal{I}$ and outputting the final environment $\Phi(E, \mathcal{I})$ after simulating the execution of the sequence in the given environment.

Our algorithm calls the function **Forward-Inference** (*j*) (Algorithm 1) to compute α_j given $\alpha_i\ \forall i<j$. To do so, the algorithm iterates over all samples $\mathcal{D}^{(j)}$ for clause C_j which are created as described in Section V. For the case when the verb $\nu(C_j)$ was unseen in the training data, we define $\alpha_j = \alpha_{j-1}$.

Each sample D_s is instantiated as described in Section 5.1 which gives us the instruction sequence $\mathcal{I}$. Instantiation is followed by all possible trimming of $\mathcal{I}$ giving us $\mathcal{I}_j = \mathcal{I}[t\ldots]$ for all *t*. We note here that the *no-op* is also considered when $\mathcal{I}$ is totally trimmed.

The trimmed sequence $\mathcal{I}_j$ may not be executable due to some missing instructions. To handle this, the sub-routine *getConstraints* looks at the pre-condition of every instruction in $\mathcal{I}_j$ to find the hard-constraints required for executing $\mathcal{I}_j$. These constraints along with the environment are passed onto a symbolic-planner (Rintanen, 2012) which outputs the missing instructions $\mathcal{I}_{j\ell}$. The cost of the new assignment $\mathcal{I}_{j\ell}, \mathcal{I}_j$ is used to update the value of α_j as shown in line 13.

Once the $\alpha_j[E]$ has been computed $\forall j$ and all reachable *E*, the optimum assignment is computed by backward-traversal.

6.6. Learning method

For training in our experiments, we divide the dataset into k-folds for cross-validation and use stochastic gradient descent to learn the weights. While doing experiments, we manually fixed the weight $w_{jmp} = \infty$ since we do not want the inferred sequence to be unexecutable. The remaining weights were initialized to 0.

7. Robot simulator for data collection

To train our algorithm, we need a dataset in the VEIL format as described in Section V. This is challenging for two reasons. Firstly, it is difficult to find natural language commands paired with environment descriptions using approaches such as web-mining, and secondly, it is difficult to elicit controller instruction sequences from non-expert users.

We therefore designed a crowd-sourcing system to solve these challenges. Our system consists of a virtual simulator which presents a 3D environment to users. Figure 5 shows sample environments that are presented to the users. Each environment has around 30 objects of which around 50 % are interactive. The interactive objects have 2-5 states in addition to location and orientation. The value of these states can be boolean, real number or strings from a finite set, and can be modified by the interaction of a user with the environment. For example, the object of category *mug* has the following states: *percentage-filled; is-grasped; has-coffee; has-water; has-ice _ cream; has-syrup*. Objects also have affordances such as *graspable, pourable, placeable*, etc. There could be more than one object of the same category and appearance.

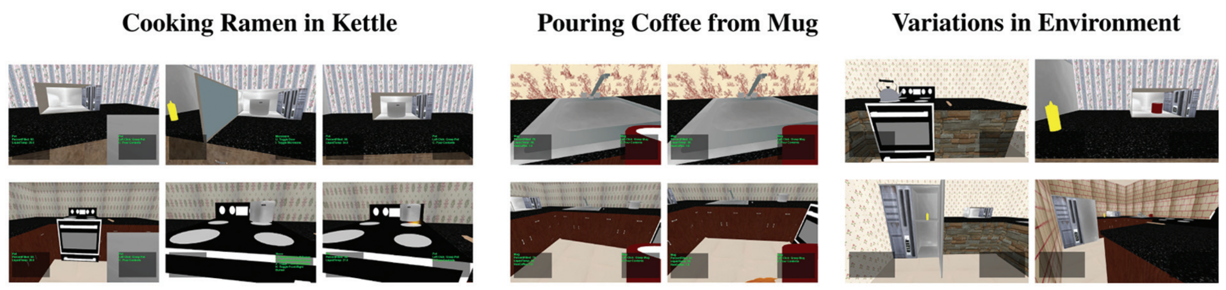


Fig. 5. Left: Different ways of performing the same task. Users have multiple options to do a task, example ramen can be cooked using a microwave or a stove. Thus, the algorithm should closely follow the sentence specification and not simply work with the high-level objective of making ramen. Center: Unrestrictive actions. Actions are unrestricted and can be done even when they have no semantic meaning. Example, coffee can be poured in a sink or on the floor. Right: Environment variation. Shows variations in an environment created by modifying the object set, state values of objects and object-object relationship.

Users of our system are presented with two type of tasks, *language task* and *game task*. In the language task, they write natural language commands for accomplishing a given objective after exploring the environment. In the game task, users are presented with natural language commands written by other users and have to control the virtual robot to accomplish the given command. As shown in Figure 6, users can point on objects to view their states and can modify them by taking different actions. Whenever a user takes an action, it is stored in the database in a symbolic form. For instance, if a user drops a cup on a table then the action is recorded as `keep(cup, table, on)`. Since that may not have been the intention of the user, therefore we also store the discretized trajectory of the action.

In the following section, we describe how we use this system to collect a VEIL dataset for testing our model.

8. Experiments and results

Dataset Details. For evaluating our approach, we have collected a dataset called *VEIL-300*. Each data point consists of a natural language command, the starting environment, ground-truth instruction sequence and the mapping between segments of the command and the instruction subsequence.

The natural language command describes the high-level task but does not necessarily have the exact knowledge of the environment (e.g. it may refer to a cup while there is no actual cup in the environment). Furthermore, it may miss several steps in the middle, and use ambiguous words for describing a task. As we can see from the examples in Table 1, the context plays a very important part in grounding these sentences. For example, in sentence 5 from Table 1, scooping ice-cream using a spoon and keeping it on a table would make little sense compared to adding it to the coffee that has been just prepared.

The dataset is then processed to be in the VEIL format (see Section 5).

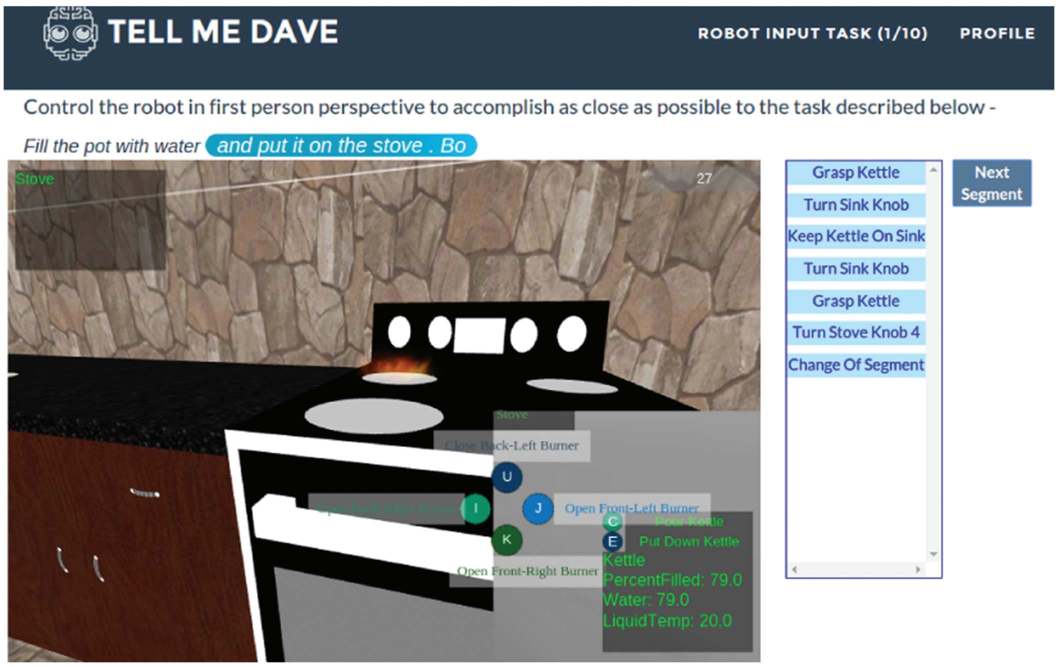
We considered six tasks: *boiling water; making coffee; making ramen*; *making affogato; prepare the room for party*; and *clean the room*. Each task was performed in five different environments and each environment had ten natural language commands (thus giving us a dataset of size $6 \times 5 \times 10 = 300$).

For the first two tasks, we only gave ten “general” natural language instructions ($5 \times 10 = 50$). This allowed us to evaluate whether our algorithm can ground natural language instructions in different environments.

For the last four tasks, there was a different natural language instruction for each of the 50 data points. We evaluate the six tasks in three pairs of two: {(task1, task2), (task3, task4), (task5, task6)}. For each of these three experiments, we use 10-fold cross validation for evaluation. While testing, the algorithm always tests on a new environment.

Evaluation metric. We consider two different type of evaluation metrics;

1) *Instruction Edit Distance (IED)*. We use a string edit distance namely the *Levenshtein distance*, for measuring the edit distance between the two instruction sequences. This gives some idea on how similar our model's output $\hat{\mathcal{I}}$ is to the ground-truth sequence $\mathcal{I}^g$. However, it is limited in that it does not handle the case where one wrong instruction in the middle can completely change the resulting state. For instance, it is not acceptable to forget filling a pot with water while making ramen, even if the rest of the sequence is correct.
2) *Environment Edit Distance (EED)*. This metric relies on the fact that *a successful execution of a task given the NL instruction is the attainment of a sequence of states for concerned objects in a possibly time-ordered way*. For this, the metric computes the edit distance between the ground-truth sequence of environments $(E^g_k)^m_{k=0}$ and the predicted sequence of environments $(\hat{E}_k)^n_{k=0}$. However, there are two subtle issues: finding



Control the robot in first person perspective to accomplish as close as possible to the task described below -

Fill the pot with water *and put it on the stove . Bo*

Fig. 6. Robotic simulator. User points on the stove to see the possible actions. User can also see different states of the stove and the grasped kettle thus taking appropriate actions. The natural language segment on which the user is working, is shown above the simulator while the recorded instructions are shown in the right panel.

the set of concerned objects ∂; and finding the correct representation of difference, $\text{dist}_{env}(E_i^g, E_j^a)$, between two environments. This is because not all the object states are important in an environment, for a given task (e.g. closing the microwave door after its use is irrelevant to the task of heating water).

We use the objects in the ground-truth sequence $\mathcal{I}^g$ as the set of concerned objects ∂. Further we define dist_{env} as a 0-1 loss function on whether the two environments completely agree on the states of ∂. The recursive equation for computing *EED* is given below, where *EED*(.,.,*i, j*) represents the distance between $(E_k^g)_{k=i}^m$, $(\hat{E}_k)_{k=j}^n$

$$\begin{aligned}&\text{EED}((E_k^g)_{k=0}^m, (\hat{E}_k)_{k=0}^n, i=0, j=0) = \\ &\quad \min\{\text{EED}(\cdot,\cdot,i,j+1), \\ &\text{EED}(\cdot,\cdot,i+1,j) + \text{dist}_{env}(E_i^g, \hat{E}_j, \partial)\} \\ &\quad \text{where } \text{EED}(\cdot,\cdot,i,j) = m - i \text{ if } i = m \text{ or } j = n\end{aligned}$$

We normalize these two metrics and report them as percentages in Table 4, where higher numbers are better.

Baselines. We compare our model against the following;

- *Chance*. Randomly chooses an instruction sequence of fixed length. This shows how large the output space is.
- *Predefined templates, based on Guadarrama et al. (2013)*. We developed a method similar to Guadarrama et al. (2013), in which we manually created a library of proposal-templates for every verb. For a given situation, we disambiguate the language-objects and accordingly instantiate the templates. We further extend their approach by also considering many-many mappings.
- *Instruction-tree search, based on Bollini et al. (2012)*. We define a log-linear reward function that searches over the sequence of instructions for every clause and chooses the one which maximizes the reward. The reward function contains bag-of-word features (correlation between words in the clause and objects in the instruction sequence). Invalid instruction sequences are pruned and gradient descent is used for training the parameters.
- *Nearest environment instantiation*. This method looks up for the nearest template by minimizing the distance between the given environment and that of the template, from the VEIL library. Template is instantiated according to the new environment, following Section 5.1.
- *Our model - no NLP.* Our model in which we do not give the natural language command, i.e. the given clauses $\mathcal{C}$ are missing in the model. However, all the other terms are present.
- *Our model - no domain knowledge*. Our model in which the robot does not have the knowledge of the results of its action on the environment, and instead relies only on the training data. (This is to compare against the symbolic planner based approaches.) For this we disable the latent, jump features and post-condition features. We also simplify the inference

Table 4. Quantitative results on six different tasks from our dataset. Results of baselines, different variants of our method are reported on two different metrics (*IED* and *EED*), which are normalized to 100 with larger numbers being better.

	making coffee		*boiling water*		*making ramen*		*making affogato*		*party night*		*clean room*		Average	
	IED	EED	IED	EED	IED	EED	IED	EED	IED	EED	IED	EED	IED	EED
Chance	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Predefined Templates (Guadarrama et al., 2013).	14.3	22.8	21.5	38.8	32.8	28.7	14.0	17.0	7.1	10.0	16.7	13.7	17.8	21.8
Instruction-Tree (Bollini et al., 2012).	14.4	25.7	11.2	38.8	26.3	25.6	12.5	15.9	22.7	49.7	20.5	14.3	17.9	28.3
No NL Instructions.	39.2	43.9	21.8	43.3	1.2	15.7	12.8	11.5	15.0	17.9	10.8	9.4	16.8	23.6
Nearest Environment Instantiation.	63.4	47.3	43.5	50.1	52.1	51.4	45.4	49.2	47.5	58.3	61.7	41.7	52.3	49.7
Our model, no domain knowledge.	78.6	71.7	55.7	55.2	66.1	49.5	51.2	51.3	51.8	56.4	**69.2**	47.1	62.1	55.2
Our model, no latent nodes.	**78.9**	73.8	**57.0**	58.4	66.3	**53.9**	51.5	51.2	**53.5**	**62.6**	66.2	**51.5**	**62.2**	58.6
Our full model.	76.5	**78.3**	54.5	**61.3**	**67.3**	49.4	**53.5**	**56.2**	53.2	**62.6**	66.0	**51.5**	61.8	**59.9**

procedure alg1, which uses the simulator Φ and hence the domain knowledge, by doing a greedy assignment based on the initial environment.

- *Our model - no latent nodes*. Our model in which the latent instruction and environment nodes are missing. This model cannot handle the missing natural language instructions well.
- *Our full model*. This is our full model.

Results. Table 4 shows our results. We note that the chance performance is very low because the output space (of instruction sequence) is very big, therefore the chances of still getting a correct sequence are very low.

We compare our results to our implementation of two recent notable works in the area. Table 4 shows that the method *Predefined Templates (Guadarrama et al., 2013)* focused on disambiguating spatial relations but was extremely brittle to ambiguity in grounding, therefore giving low performance.

Method *Instruction-Tree (Bollini et al., 2012)* was able to give reasonable results for some data points. However this approach has problems working with large search trees. Furthermore, the bag-of-word features do not take into account the environment context. For instance, the natural language instruction might say *"keep the cup in microwave"* but the cup might already be inside the microwave (unless such constraints are hard-coded). This approach thus fails when the language is vague, for example, for the following sentence, *"heat the water and add ramen"*. However, our approach takes this vague sentence and grounds it in the environment using our model. Our energy function incorporates several features and thus is able to often give reasonable output for such natural language instructions.

Following are some natural language instructions that our algorithm was able to ground successfully:

- *"serve the chips and then bring the pillows to the couches and turn on the tv";*
- *"clean up the trash and replace the pillows and turn off the tv";*
- *"fill the pot with ramen and water and then cook it".*

and following are some natural language instructions that our algorithm was not able to ground:

- *"throw away the chips and throw away the remaining bag";*
- *"Microwave for 12 minutes and place it on the table";*
- *"heat up water in the pot and place ramen inside pot".*

Common reasons for errors include the required VEIL template not being present, incorrect object grounding and other linguistic complications that we are not handling in this paper such as conditionals, quantifiers etc.

We analyze the results in light of the following questions.

Is language important? If we enforce all the constraints of the task and provide the end-state of the environment, one may argue that just using a symbolic planner may give reasonable programs. However, the success of a task depends on the way things are done. Natural language gives an approximate guide that our model tries to follow. For example, we may provide that the cup has water in the goal environment. But if the natural language command is *"fill a cup with water from the sink"* then an instruction sequence that uses the fridge to fill the cup with water will result in failure.

We see that *"Our model - no NLP"* gives 16.8% on average on the IED metric as compared to 61.8% for our full model.

In fact, we see evidence of such behavior in our results also. While our model can handle ambiguous and incomplete NL instructions, e.g. *"heat up the water and then cook the ramen"* that resulted in success, in some of the test cases the NL instructions were quite ambiguous, e.g. *"Microwave for 12 minutes and place it on the table"* on which our model failed.

How important is the latent node? Overall, Table 4 shows that the results improve by about 1.3% on the EED metric after adding the latent node. We found that it was especially helpful in scenarios where instructions were partially missing. For example, for the instruction in Figure 2 *"place the pot on the tap and turn the tap on…Turn the tap off and heat the pot."* there is no template that can fit in for the first clause (`place`, [`pot, tap`], `on`: `pot`→`tap`). One such template after initialization has the form

```
moveTo(sink); keep(pot, sink, on)
```

However the robot cannot execute this sequence since it is not already grasping the p*ot*. In such cases, the latent nodes help us with modeling these missing instructions and give us the following executable output

```
moveTo(pot); grasp(pot); moveTo(sink); keep
(pot, sink, on)
```

How well does our model generalize to new environments and tasks? In this test, we wanted to examine how well our model can make use of examples from different environments and tasks. It is not obvious whether the templates learned for one task, such as *making ramen* will be useful for another task such as *making affogato*. For studying this, we performed another experiment in which we trained and tested the model on *making affogato* task only and then we added samples from *making ramen* to the training. We found that because the VEIL library from the *making ramen* task was not available for training, the performance dropped by 1.5-2% on the *IED* metric. The results are shown in Figure 7. This suggests that examples from other tasks are helpful.

How does the algorithm handle noisy and incomplete sentences? The algorithm handles noisy object references as part of the instantiation procedure. For example, consider that the robot is asked to get a cup of water and there is no object of category cup in the given environment. In this case, the function *Ground* (*"cup", E*) will return *Null* and the instantiation procedure 5.1 will then find an object in the given environment which shares the most state-value pairs with the object that it saw in the training environment. This object is then taken as the intended object.

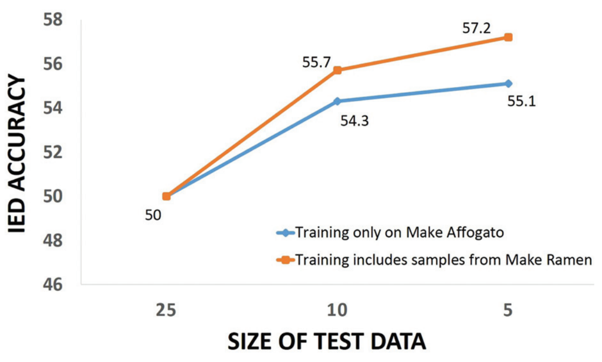


Fig. 7. IED accuracy on the "Making Affogato" dataset versus size of test data. Cross-validation is used with test-data of the given size. In another experiment, we add "Making Ramen" samples to the training and we see an improvement of 1-2% for test data of size 5 and 10. This shows that samples from other tasks were useful.

Our algorithm also handles several incomplete sentences such as *"microwave a cup of water"*, which does not specify filling the cup with water in case it is empty, or opening the door of the microwave if it is closed etc. The algorithm handles this in two ways, either by using a similar sample, that was present in the VEIL template which contains the missing steps, or if the missing steps constitute a hard constraint (such as opening the door of a microwave before trying to put a cup in) then they are handled by the latent node $\mathcal{I}_\ell$ using a symbolic planner.

What if the robot does not know the result of its action? The algorithm implicitly assumes that the robot knows the result of its interaction with the environment (it is being used to compute certain features, the planning and in inference). In order to test how crucial it is, we ran the baseline *Our model - no domain knowledge* and as the results in Table 4 show, the accuracy falls by only 4.7 % on the EED metric. However, without the domain knowledge the guarantee that the output sequence is executable is lost. The robot will then have to recover at run-time if the lower level controllers fail.

9. Robot experiment

We use our grounding algorithm in an end-to-end robotic system, which can take NL instructions from users and manipulate real world objects to achieve the task. In this section we describe our robotic system for the task of making affogato dessert. We take the following NL instruction, given by a user, as our working example: *"Take some*

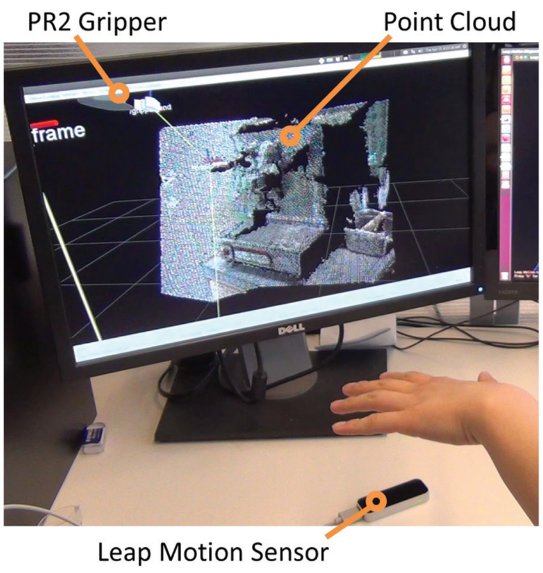


Fig. 8. The demonstration system using Leap motion sensor. The expert demonstrates the instruction for object shown on the screen by hovering over the sensor and moving the gripper of the PR2 on the screen.

coffee in a cup. Add ice cream of your choice. Finally, add raspberry syrup to the mixture."

Grounded instruction sequence given by our grounding algorithm consists of instruction listed in Table 2, with appropriate parameters which are chiefly objects. Although there are many related works for each instruction (Anand et al., 2012; Bollini et al., 2011; Endres et al., 2013; Jiang et al., 2012; Lenz et al., 2013; Ratliff et al., 2009), it is still a very active area of research and it is quite challenging to execute these instructions reliably in a sequence. Thus, we take a Learning from Demonstration approach (Argall et al., 2009) for the set of instructions relevant to the task, in order to test the validity of the grounded sequence given by our model.

First, the robot observes the scene using RGB-D sensor (Microsoft Kinect). Given the point cloud from the sensor, the scene is segmented into smaller patches representing object parts (Anand et al., 2012). In order to reuse demonstrations regardless of orientation and location of the object, the object frame using the segmented object part is found for the use of demonstration and execution. Reliable frame of the object can be established by aligning axis with the principal axis of the point cloud computed using PCA (Hsiao et al., 2010). The segmentation and alignment of the axis with the segmented point-cloud allows the manipulation to be location and orientation invariant when the demonstration is trained with respect to this frame.

Among many approaches of Learning from Demonstration such as kinesthetic teaching, teleoperation, and so forth (Argall et al., 2009), we use a teleoperation-based approach for demonstrating the task to the robot. Figure 8, shows our system built using the Leap motion sensor where a user can teach the robot how to execute the instruction with certain parameters (objects). By observing the object shown on-screen, the user controls the gripper of the robot by hovering over the Leap motion sensor. The center of the object frame is virtually placed a few centimeters above the Leap Motion sensor, which allows an expert to easily demonstrate the sequence by controlling the gripper with his palm. Rather than recording the full movement of the demonstrator which could be not smooth, the recorded sequence for the robot to execute is based on a sequence of keyframe similar to Akgun et al. (2012). Each keyframe is recorded by pressing the key as the user demonstrates the instruction. Also, rather than recording the full joint configuration, each keyframe records only the location and orientation of the end-effector so that it can used regardless of the location of the object relative to the robot.

For the robotic experiment, demonstration given by the user can be executed in a new setting by segmenting the point cloud, finding the object frame and then executing the demonstration using the trajectory controller. We utilize the impedance controller (`ee_cart_imped`) (Bollini et al., 2011) for our PR2 robot to follow the end-effectory trajectory of learned demonstration. Once the object frame is centered correctly, the robot can successfully execute the instruction, as long as there is no collision and the object is in the workspace of the robot. This is because each control sequence was trained with respect to the object frame.

Figure 9 and 10 show several snapshots of PR2 making Affogato, making coffee, and preparing ramen. See also Extension 1 which shows PR2 making affogato dessert.

10. Future work

There are many directions in which we can extend the approach presented in this paper, as described briefly in the following.

Improving the language model. In this paper, our main focus has been on grounding high-level verbs which possess the most complex semantic structure. For grounding noun-phrases such as *"the white cup", "red mug by the door"*; we used a simple syntactical matching based on object categories. While this worked relatively well for our dataset, this needs to be improved to handle complex object descriptions. These complex object descriptions can contain determiners (e.g. *"bring me the cup"*), anaphoric reference (e.g. *"keep it on the table"*), recursive object descriptions (e.g. *"microwave the cup which is on the table, close to fridge"*) etc.

As mentioned before, there are several works (Guadarrama et al., 2013, 2014) which aim to solve this problem. The algorithms can be easily integrated into our model by replacing the *Ground* function in Section 5.1 with them.

Besides handling complex object descriptions, the language model needs to handle conditionals (e.g. *"if there is*

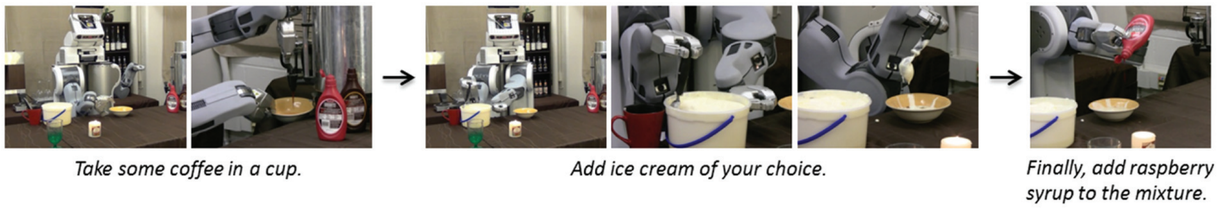

Take some coffee in a cup.
Add ice cream of your choice.
Finally, add raspberry syrup to the mixture.

Fig. 9. Robot experiment. Given the natural language instruction for making the “Affogato” dessert: *“Take some coffee in a cup. Add ice-cream of your choice. Finally, add raspberry syrup to the mixture.”*, our algorithm outputs a controller instruction sequence, which is mapped to previously learned trajectories in order to make the dessert. (See Extension 1)

Fig. 10. Snapshots of PR2 using the microwave for making coffee when only the microwave is available for heating the water and preparing ramen.

a cup”), quantifiers as well as domain specific expressions such as time (e.g. *“microwave the cup for 12 minutes”*) and quantity (e.g. *“take 2 ounces of potatos”*). Each of these challenges is an interesting problem in the field of natural language understanding. We refer the interested readers to Artzi and Zettlemoyer (2013) for parsing quantifiers and spatial relations and Lee et al. (2014) for parsing time expressions.

Richer Representations. In this paper, we ground natural language commands to a sequence of instructions, each having a single predicate, `controller-name(arguments)`. These instructions are then mapped onto trajectories (see Section IX). One advantage of this is that instructions with one predicate can be easily recorded in our crowd-sourcing system from user interaction. However, single predicate instructions cannot handle variable numbers of arguments which may be present in the natural language command. For example, the verb *move* can accept varied numbers of arguments and modifiers as shown in Table 5. A single predicate cannot handle all of these cases. Whereas having a predicate for every situation leads to combinatorial explosion. In the field of semantics, a solution is to use Neo-Davidsonian semantics which defines a common event shared between multiple atoms (instantiated predicate). Table 5 shows examples of Neo-Davidsonian semantics.

In future, we want to ground natural language to a sequence of instructions where each instruction is a conjunction of atoms coupled by a Neo-Davidsonian event.

Learning environment model: As mentioned before, in this paper we assume access to domain knowledge of the environment. We also assumed that the world is deterministic. In future, we want to learn the environment model along with learning to ground natural language. One direction could be to use the reinforcement learning setting of Branavan et al. (2010).

Real-time inference: The inference algorithm presented in this paper uses all the samples corresponding to a verb, which makes the algorithm impractical for a very large dataset. In future, we plan to use better approximate inference techniques to make the inference real-time while maintaining accuracy.

In future, we also want to test our algorithm on much larger datasets as well as improving our crowd-sourcing system by integrating it with the RoboBrain platform (Saxena et al., 2014).

11. Conclusion

In this paper, we presented a novel approach for grounding free-form natural language instructions into a controller instruction sequence for a given task and environment, that can be executed by a robot to perform the task. This is

Table 5. Neo-Davidsonian semantics for representing verbs with varying number of arguments, modifiers and relative clauses. Variable e is the Neo-Davidsonian event.

Sentence	Neo-Davidsonian semantics
1. *"move to the cup"*	$\exists e.\mathtt{moveto}(e) \wedge \mathtt{to}(e, \mathtt{cup}_{02})$
2. *"slowly move to the cup"*	$\exists e.\mathtt{moveto}(e) \wedge \mathtt{to}(e, \mathtt{cup}_{02}) \wedge \mathtt{speed}(e, \mathtt{slow})$
3. *"slowly move to the cup by staying close to the wall"*	$\exists e.\mathtt{moveto}(e) \wedge \mathtt{to}(e, \mathtt{cup}_{02}) \wedge \mathtt{speed}(e, \mathtt{slow}) \wedge \mathtt{close}(e, \mathtt{wall})$

challenging since the grounding is highly dependent on the environment. Another challenge is that the natural language instructions can be incomplete and ambiguous as shown in Table 1. To solve these issues, we represented this context in a VEIL dataset format which was collected using crowd-sourcing. We presented a new learning model that encodes certain desired properties into an energy function, expressed as a model isomorphic to a conditional random field with edges representing relations between natural language, environment and controller instructions. We showed that our model handles incomplete natural language instructions, variations in natural language, as well as ambiguity in grounding. We also showed that we outperform related work in this area on both syntax-based and semantic-based metrics.

Acknowledgements

We thank Claire Cardie for useful discussions and Kejia Tao and Aditya Jami for help with the crowd-sourcing system. The first version of this work was presented at Robotics Science and Systems (RSS) conference in July 2014 (Berkeley) (Misra et al., 2014).

Funding

This work was supported in part by ONR Grant N00014-14-1-0156, Microsoft Faculty Fellowship and NSF Career award to Saxena.

References

Akgun B, Cakmak M, Jiang K and Thomaz A (2012) Keyframe-based learning from demonstration. *International Journal of Social Robotics* 4(4): 343–355.

Alterovitz R, Patil S and Derbakova A (2011) Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning. In: *ICRA*.

Anand A, Koppula H, Joachims T and Saxena A (2012) Contextually guided semantic labeling and search for 3D point clouds. *The International Journal of Robotics Research* 32(1): 19–34.

Argall B, Chernova S, Veloso M and Browning B (2009) A survey of robot learning from demonstration. In: *Robotics and Autonomous Systems*.

Artzi Y and Zettlemoyer L (2013) Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics* 1: 49–62.

Aydemir A, Sjoo K, Folkesson J, Pronobis A and Jensfelt P (2011) Search in the real world: Active visual object search based on spatial relations. In: *ICRA*. IEEE, pp. 2818–2824.

Barry J, Hsiao K, Kaelbling LP and Lozano-Pérez T (2013) Manipulation with multiple action types. In: *Experimental Robotics* (*Springer Tracts in Advanced Robotics*, Vol. 88). New York: Springer, pp. 531–545.

Beetz M, Klank U, Kresse I, et al. (2011) Robotic roommates making pancakes. In: *Humanoids*.

Berant J, Chou A, Frostig R and Liang P (2013) Semantic parsing on freebase from question-answer pairs. In: *Proceedings of EMNLP*.

Bollini M, Barry J and Rus D (2011) Bakebot: Baking cookies with the pr2. In: *The PR2 Workshop, IROS*.

Bollini M, Tellex S, Thompson T, Roy N and Rus D (2012) Interpreting and executing recipes with a cooking robot. In: *ISER*.

Branavan S, Zettlemoyer L and Barzilay R (2010) Reading between the lines: Learning to map high-level instructions to commands. In: *ACL'10 proceedings of the 48th annual meeting of the Association for Computational Linguistics*. Stroudsburg, PA: ACL, pp. 1268–1277.

Cakmak M, Dogar M, Ugur E and Sahin E (2007) Affordances as a framework for robot control. In: *EpiRob*.

Chao C, Cakmak M and Thomaz A (2011) Towards grounding concepts for transfer in goal learning from demonstration. In: *ICDL*.

Chen DL, Kim J and Mooney RJ (2010) Training a multilingual sportscaster: Using perceptual context to learn language. *Journal of Artificial Intelligence Research* 37(1): 397–436.

Chu V, McMahon I, Riano L, et al. (2013) Using robotic exploratory procedures to learn the meaning of haptic adjectives. In: *IROS*.

Duvallet F, Walter MR, Howard T, et al. (2014) Inferring maps and behaviors from natural language instructions. In: *Proceedings of the international symposium on experimental robotics (ISER)*.

Endres F, Trinkle J and Burgard W (2013) Learning the dynamics of doors for robotic manipulation. In: *IROS*.

Farhadi A, Endres I and Hoiem D (2010a) Attribute-centric recognition for cross-category generalization. In: *CVPR*.

Farhadi A, Hejrati M, Sadeghi MA, Young P, Rashtchian C, Hockenmaier J and Forsyth D (2010b) Every picture tells a story: Generating sentences from images. In: *Computer vision–ECCV* 2010. New York: Springer, pp. 15–29.

Fasola J and Matarić M (2013) Using semantic fields to model dynamic spatial relations in a robot architecture for natural language instruction of service robots. In: *IROS*.

Fikes RE and Nilsson NJ (1972) Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3): 189–208.

Finucane C, Jing G and Kress-Gazit H (2010) Ltlmop: Experimenting with language, temporal logic and robot control. In: *IROS*.

Guadarrama S, Riano L, Golland D, et al. (2013) Grounding spatial relations for human-robot interaction. In: *IROS*.

Guadarrama S, Rodner E, Saenko K, et al. (2014) Open-vocabulary object retrieval. In: *Robotics: science and systems*.

Höfer S, Lang T and Brock O (2014) Extracting kinematic background knowledge from interactions using task-sensitive relational learning. In: *ICRA*.

Hsiao K, Chitta S, Ciocarlie M and Jones E (2010) Contact-reactive grasping of objects with partial shape information. In: *IROS*.

Jiang Y, Koppula H and Saxena A (2013) Hallucinated humans as the hidden context for labeling 3d scenes. In: *CVPR*.

Jiang Y, Lim M, Zheng C and Saxena A (2012) Learning to place new objects in a scene. *The International Journal of Robotics Research* 31(9): 1021–1043.

Kaelbling LP and Lozano-Pérez T (2011) Hierarchical task and motion planning in the now. In: *ICRA*.

Kjellström H, Romero J and Kragić D (2011) Visual object-action recognition: Inferring object affordances from human demonstration. *Computer Vision and Image Understanding* 115(1): 81–90.

Klein D and Manning C (2003) Accurate unlexicalized parsing. In: *ACL*.

Kollar T, Tellex S, Roy D and Roy N (2010) Grounding verbs of motion in natural language commands to robots. In: *ISER*.

Koppula H, Anand A, Joachims T and Saxena A (2011) Semantic labeling of 3d point clouds for indoor scenes. In: *NIPS*.

Koppula H and Saxena A (2013) Anticipating human activities using object affordances for reactive robotic response. In: *RSS*.

Koppula HS, Gupta R and Saxena A (2013) Learning human activities and object affordances from RGB-D videos. *The International Journal of Robotics Research* 32(8): 951–970.

Kress-Gazit H, Fainekos G and Pappas G (2007) From structured English to robot motion. In: *IROS*.

Kroemer O, Detry R, Piater J and Peters J (2010) Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems* 58(9): 1105–1116.

Kulick J, Toussaint M, Lang T and Lopes M (2013) Active learning for teaching a robot grounded relational symbols. In: *IJCAI'13 Proceedings of the twenty-third international joint conference on artificial intelligence*. AAAI Press, pp. 1451–1457.

Kwiatkowski T, Zettlemoyer L, Goldwater S and Steedman M (2010) Inducing probabilistic ccg grammars from logical form with higher-order unification. In: *EMNLP*. Stroudsburg, PA: ACL, pp. 1223–1233.

Lee K, Artzi Y, Dodge J and Zettlemoyer L (2014) Context-dependent semantic parsing for time expressions. In: *ACL 2014 workshop on semantic parsing*. Stroudsburg, PA: ACL.

Lemaignan S, Ros R, Sisbot EA, Alami R and Beetz M (2012) Grounding the interaction: Anchoring situated discourse in everyday human–robot interaction. *International Journal of Science and Research* 4(2): 181–199.

Lenz I, Lee H and Saxena A (2013) Deep learning for detecting robotic grasps. In: *Robotics: science and systems*.

Malmaud J, Wagner E, Chang N and Murphy K (2014) Cooking with semantics. In: *ACL 2014 workshop on semantic parsing*. Stroudsburg, PA: ACL, pp. 33–38.

Marco D, Tenorth M, Häussermann K, Zweigle O and Levi P (2012) Roboearth action recipe execution. In: *IAS*.

Matuszek C, Fitzgerald N, Zettlemoyer L, Bo L and Fox D (2012a) A joint model of language and perception for grounded attribute learning. In: *ICML*, pp. 1671–1678.

Matuszek C, Herbst E, Zettlemoyer L and Fox D (2012b) Learning to parse natural language commands to a robot control system. In: *ISER*.

Misra D, Sung J, Lee K and Saxena A (2014) Tell me Dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In: *RSS*.

Montesano L, Lopes M, Bernardino A and Santos-Victor J (2008) Learning object affordances: From sensory–motor coordination to imitation. *IEEE Transactions on Robotics* 24(1): 15–26.

Mourao K, Zettlemoyer L, Petrick R and Steedman M (2012) Learning strips operators from noisy and incomplete observations. In: *Proceedings of the twenty eighth conference on uncertainty in artificial intelligence*.

Nguyen H, Ciocarlie M, Hsiao J and Kemp CC (2013) Ros commander (rosco): Behavior creation for home robots. In: *ICRA*.

Niekum S, Chitta S, Barto A, Marthi B and Osentoski S (2013) Incremental semantically grounded learning from demonstration. In: *RSS*.

Nothman J, Honnibal M, Hachey B and Curran J (2012) Event linking: Grounding event reference in a news archive. In: *ACL*.

Poon H (2013) Grounded unsupervised semantic parsing. In: *ACL*.

Ramanathan V, Liang P and Fei-Fei L (2013) Video event understanding using natural language descriptions. In: *ICCV*.

Ratliff N, Zucker M, Bagnell D and Srinivasa S (2009) Chomp: Gradient optimization techniques for efficient motion planning. In: *ICRA*.

Rintanen J (2012) Planning as satisfiability: Heuristics. *Artificial Intelligence* 193: 45–86.

Ros R, Lemaignan S, Sisbot EA, et al. (2010) Which one? Grounding the referent based on efficient human-robot interaction. In: *RO-MAN*, pp. 570–575.

Saxena A, Jain A, Sener O, et al. (2014) Robobrain: Large-scale knowledge engine for robots. *Preprint arXiv:1412.0691*.

Srinivasa S, Ferguson D, Helfrich C, et al. (2010) Herb: A home exploring robotic butler. *Autonomous Robots* 28(1): 5–20.

Steedman M (1996) *Surface structure and interpretation*. Cambridge, MA: MIT Press.

Steedman M (2000) *The syntactic process,* volume 35. Cambridge, MA: MIT Press.

Sung J, Selman B and Saxena A (2014) Synthesizing manipulation sequences for under-specified tasks using unrolled Markov random fields. In: *IROS*.

Tellex S, Knepper RA, Li A, et al. (2013) Assembling furniture by asking for help from a human partner. In: *Collaborative manipulation workshop at human–Robot interaction*.

Tellex S, Kollar T, Dickerson S, et al. (2011) Understanding natural language commands for robotic navigation and mobile manipulation. In: *AAAI*.

Tenorth M, Kunze L, Jain D and Beetz M (2010) Knowrob-map-knowledge-linked semantic object maps. In: *Humanoids*.

Walter M, Hemachandra S, Homberg B, Tellex S and Teller S (2013) Learning semantic maps from natural language descriptions. In: *Robotics: science and systems*.

Wongpiromsarn T, Topcu U and Murray RM (2010) Receding horizon control for temporal logic specifications. In: *HSCC*.

Wu C, Lenz I and Saxena A (2014) Hierarchical semantic labeling for task-relevant RGB-D perception. In: *Robotics: science and systems*.

Yu H and Siskind J (2013) Grounded language learning from videos described with sentences. In: *ACL*.

Zettlemoyer L and Collins M (2007) Online learning of relaxed ccg grammars for parsing to logical form. In: *EMNLP-CoNLL-2007*.

Appendix: Index to Multimedia Extension

Archives of IJRR multimedia extensions published prior to 2014 can be found at http://www.ijrr.org, after 2014 all videos are available on the IJRR YouTube channel at http://www.youtube.com/user/ijrrmultimedia

Table of Multimedia Extensions

Extension	Media Type	Description
1	Video	The PR2 robot takes a command from a user and executes our algorithm to find an instruction sequence for accomplishing the task. PR2 then executes these instructions sequentially as described in the paper.