Tell Me Dave: Context-Sensitive Grounding of Natural Language to Manipulation Instructions

Dipendra K Misra, Jaeyong Sung, Kevin Lee and Ashutosh Saxena.
Computer Science Department, Cornell University (USA).
Email contact: {dkm,jysung,kkl53,asaxena}@cs.cornell.edu

***Abstract*—We consider performing a sequence of mobile manipulation tasks with instructions given in natural language (NL). Given a new environment, even a simple task such as of 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 NL instructions. Our model is based on an energy function that encodes such properties in a form isomorphic to a conditional random field. In evaluation, we show that our model produces sequences that perform the task successfully in a simulator and also significantly outperforms the state-of-the-art. We also verify by executing our output instruction sequences on a PR2 robot.

I. Introduction

Consider the task of boiling water shown in Figure 1—it consists of a series of steps to be performed that a user describes in natural language (NL). Each of the steps 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 them. For example, for *'heating the water'*, one can either heat it in a pot over a stove or in 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 us 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 phrase 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.

Some recent notable works have explored aspects of this problem (e.g., [33, 45, 7, 4, 18]). Guadarrama et al. [18] focused on using spatial relations to ground NL objects 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 2 for an example). Beetz et al. [4, 33] considered translating web recipe into a robot making pancakes, and focused 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 sub-task. Therefore, our work focuses on considering large variations in the NL instructions for generalizing to different tasks in changing environments. Bollini et al. [7] 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.

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 models the ambiguities in grounding them to robotic instructions with 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 four different tasks, with 5 different environments per task, free-

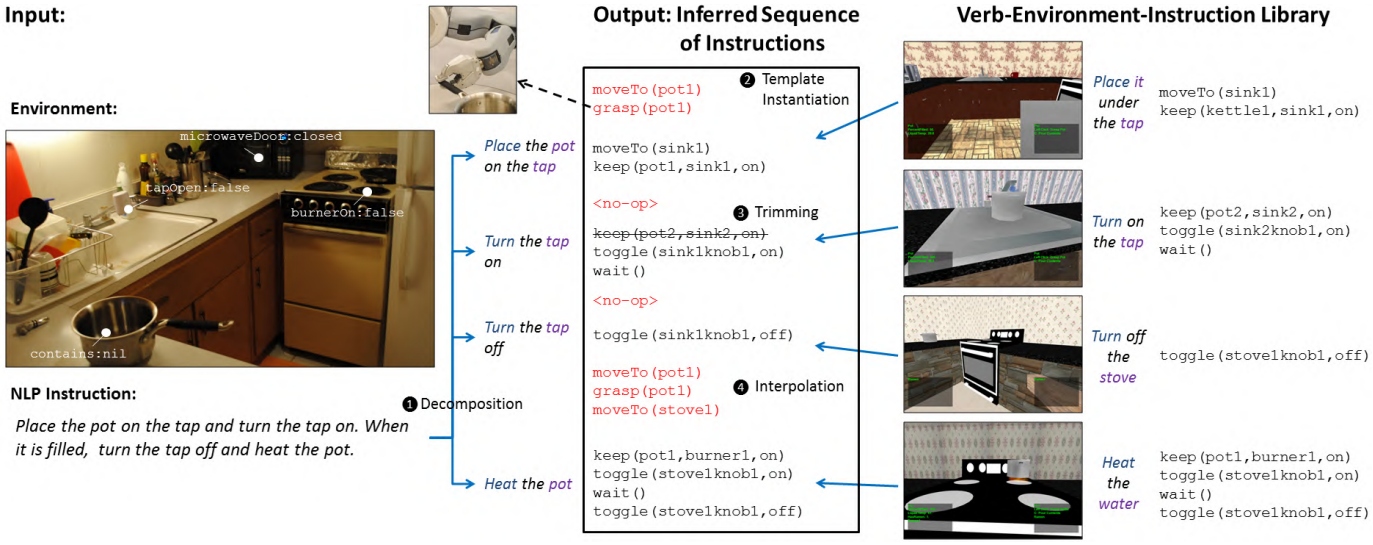

Fig. 1. **Natural Language Instructions to sequence of instructions** for a given new environment. Our approach takes description in natural language and sequences together robotic instructions that are appropriate for a given environment and task. Note that the NL instructions are often ambiguous, and are incomplete, and need to be grounded into the environment.

form natural language data 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 [18] and [7], 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 use our predicted sequences on a PR2 robot to create a dish 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 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 sequences of instructions. 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.

II. Related Work

Mobile Manipulation Tasks. In the past decade, there has been significant work on different manipulation and navigational skills such as grasping [32, 28], mixing [7], pushing [43], placing [3, 20], constructing semantic maps [48], and high degree of freedom arm planners (e.g., [40, 1]). These works form the building blocks for executing the output instructions for our model.

Traditionally, sequencing complicated controller instructions have been accomplished using symbolic planners [41]. Since real environments have uncertainty and non-determinism, Kaelbling and Lozano-Pérez [22] start with an abstract plan and recursively generate plans as needed. Or, the tasks are defined through expert designed state machines [35], which does not generalize well when the environment or the task changes. Rather than relying on symbolic representation of the environment, Sung et al. [44] 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 minimizes the score function based on the current environment and the potential candidate for the next instruction. Others use demonstrations for learning different behaviors (e.g. [36]). These approaches solve only parts of the problems that we address in this work—of creating valid plans and using a score function for data-driven retrieval of sequences of instructions. Our work addresses not only the validity of 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 several sequences of steps.

Grounding Natural Language. The use of language has gained recent attention in robotics. Other than the works discussed in the introduction [33, 7, 4, 18], the problem of navigation has been addressed by using learned models for verbs like *follow*, *meet*, *go* as well as the conditions such as *walk close to the wall* [24, 16]. In detail, Kollar et al. [24] use maximum-likelihood approach to infer the path taken by the robot. Translation of such weakly specified actions into robotic behaviors is very important; these ideas form our robotic instruction set in Table I. We go beyond navigational instructions and present a model which can ground natural language to a sequence of pre-defined set of manipulation and navigation instructions that can be executed by robots.

Several works [16, 18] have looked at the problem of grounding intricate noun-phrases in the language to the ob-

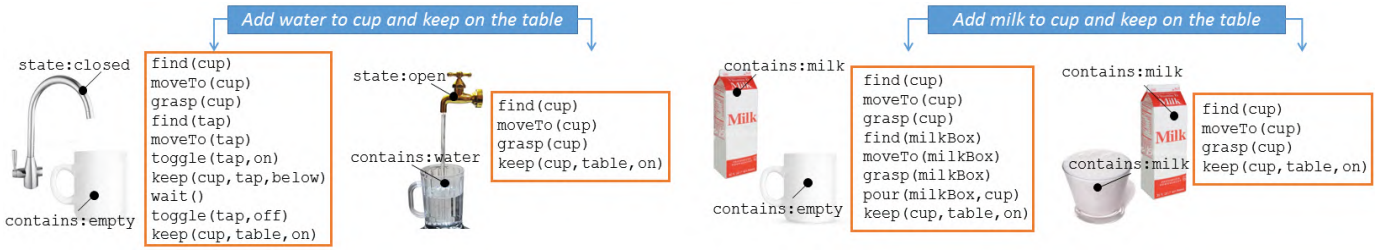

Fig. 2. **Many-many correspondence between language and controller instructions depending on the environment.** We get different instruction-sequences for `add(water, cup)` and `add(milk, cup)` depending upon the parameters `milk` and `water` and the environment the actions need to be performed in.

jects in the environment. It has been especially successful in grounding concepts [9] and objects [42, 31] through human-robot interaction. Works like Chu et al. [11] look at mapping from text to haptic signals. Kulick et al. [29] 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 [46, 10]. A recent work [12] explored the direction of using natural language as a sensor to come up with prior distribution about unseen regions of the environment.

In the area of computer vision, some works have considered relating phrases and attributes to images and videos [39, 15, 26, 25, 21, 50]. 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 grammatically parsing the sentences (e.g. [23]) and grounding text in different domains such as linking events in a news archive and mapping language to database queries [37, 51, 5, 38]. These techniques form the basis of ours in text parsing and representation. However, most of these works use only text data, and do not address grounding the physical task or the environment.

Natural language is unstructured and different intermediate representations of language have been used in previous work. Tellex et al. [45] use *tree of Spatial Description Clause (SDC)* while works such as Matuszek et al. [34] use lambda calculus representation. Others use *Linear Temporal Logic* to represent the language task which generates robot controllers which can be proven to be correct [17, 27, 49]. 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.

III. Overview

Given an environment E containing objects and the robot, a human gives the instructions for performing a task in natural language (see Figure 1). The instructions in natural language L consists 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., $\texttt{grasp}(\texttt{object}_\texttt{i})$.

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

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 such 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 are the low-level instructions $\mathcal{I}$ 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.

A. Language Representations by a Set of Verb Clauses

To handle the arbitrary structure of the natural language L, we first reduce it to a more formal intermediate structure based on clausal decomposition. 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 $\rho : obj \times obj \rightarrow Rel$ where Rel is the space of relationship (e.g. 'with', 'from'). For example, the following comprises four verb clauses:

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

In order to achieve the decomposition we begin by first parsing the sentence L using the Stanford lexical parser [23]. We traverse the parse-tree of each sentence for finding nodes with verb type and each of these is used to create a clause. We then attach objects to this clause by using minimum distance criterion in the parse tree. Care is taken to stem the verb, and noun-phrase nodes are collapsed into a single node (example `crock pot` to $\texttt{crock}-\texttt{pot}$). The relationship nodes in between two object nodes of a clause are added in the corresponding relationship matrix. Note that not all *language-objects* (e.g., water) represent an actual physical object.

B. 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.,

TABLE I

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 PR2 ROBOT (SEE SECTION VII).

Instruction	Description
`find(obj)`	Find `obj` in the environment [2].
`keep(obj1, obj2, R)`	Keeps `obj1` with respect to `obj2` such that relation R holds [20].
`grasp(obj)`	Grasp `obj` [32].
`release(obj)`	Releases `obj` by opening gripper.
`moveTo(obj)`	Move to `obj` by using motion planner [40].
`press(obj)`	Presses `obj` using end effector force controller [6].
`turn(obj, angle)`	Turns the `obj` by a certain angle.
`open(obj)`	Opens the `obj` [13].
`close(obj)`	Closes the `obj` [13].
`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.

their 3D coordinates and orientation), but also their functions. Their functions are often represented as symbolic attributes [2, 14] or in a more functional representations [47, 8, 19]. 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 the set of 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.

C. Representing Robotic Instructions

We have defined a set of low-level instructions that could be executed by the robot in Table I. 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 [8].

In order to completely execute a sub-task such as *keeping a cup on stove*, we need a sequence of instructions $\mathcal{I}$. An example of such a sequence for the sub-task might look as:

`find(cup1); moveTo(cup1); grasp(cup1); find(stove1); moveTo(stove1); keep(cup1, stoveBurner3, 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.

IV. MODEL

In this section, we describe our energy function $\Gamma(\mathcal{I}|L, E)$, which is isomorphic to a conditional random field (Fig. 3) and comprises of several nodes and factors ψ's.

It has the following observed nodes: language L, decomposed into clauses $\mathcal{C}$, 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 instructions $\mathcal{I} = \{\mathcal{I}_{i\ell}, \mathcal{I}_i\}_{i=1}^{k}$.

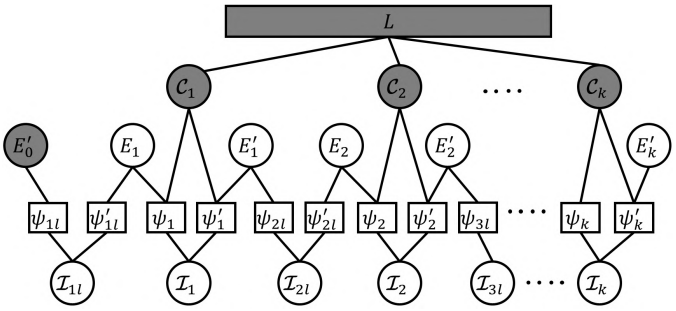

Fig. 3. **Graphical model representation of our energy function.** The clauses $\mathcal{C}$ and initial environment E'_0 is given, and we have to infer the instructions $\mathcal{I}_i, \mathcal{I}_{i\ell}$ and the environment at other time steps. The ψ's are the factors in the energy function.

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

Following the independency assumptions encoded in the graphical model in Figure 3, 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 potential functions (we describe these in detail in Section VI):

- The sequence of instructions should be *valid*. Intuitively $\psi_i(\cdot)$ represents the pre-condition 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 $\mathcal{C}_i$.
- Length of the instruction set. Shorter programs are better for doing the same task.
- Prior probabilities. A 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 obtain a few sample sequences of instructions $\mathcal{I}_i^{(s)}$ from the training data. With the samples of instructions for each clause, 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 build our verb-environment-instruction library from the training data.

V. VERB-ENVIRONMENT-INSTRUCTION LIBRARY (VEIL)

For a given verb clause $\mathcal{C}_i$, we use the training data $\mathcal{D}$ to come up with a set of samples 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 $\{z_1 \rightarrow \texttt{cup01}, z_2 \rightarrow \texttt{bottle01}\}$ 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: $\mathcal{D}^{(i)} = \{(\mathcal{C}^{(j)}, E^{(j)}, \mathcal{I}^{(j)}, \xi^{(j)}) \,|\, \nu(\mathcal{C}^{(j)}) = \nu(\mathcal{C}_i)\}$ for $j = 1 \ldots |\mathcal{D}|$ in the dataset. The information in the particular s^{th} sample is represented as $D_s = (\mathcal{C}_s, E_s, \mathcal{I}_s, \xi_s)$.

A. 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 a 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 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. We take the following approach:

- We first match the objects that have a relation. For example, if we have cup on table and template is z_1 on z_2, then we map z_1 to cup and z_2 to table. 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 variables $z_1 \mapsto a$ and $z_2 \mapsto b$.
- For a single object a used in clause $\mathcal{C}_j$ for the new environment E_j, we map it in the same way 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 $\xi_s(z_1) = a$ we map $z_1 \mapsto a$.
- For every remaining uninitialized variable $Z(s)$, we find the object in the new environment that shares the most common states with the one in the sample s.

Mapping $z \mapsto a$ of a variable z to an object a is only done if a represents an actual object in the given environment E_j (e.g., non-physical language-objects like water or objects which are mentioned in the language but are not actually present would not be used for mapping). The new mapping is stored as ξ and the instructions returned after replacing the variables using ξ is 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 $\xi(z)$.

VI. Energy Function and Inference

Now for each clause $\mathcal{C}_i$, we have obtained a sample instruction set $\mathcal{I}_i^{(s)}$ (together with 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 initial 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 potential function that encode the desired properties aforementioned.

A. 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:

$\psi_i(\mathcal{I}_i, \mathcal{C}_i, E_i | D_s, \xi; w) = w^T$

$$\begin{bmatrix} \Delta_{env}(D_s, \xi); & \Delta_{nl}(\mathcal{C}_s, \mathcal{C}_i); & \Delta_{sim}(Z(s), \xi_s, \xi); \\ \Delta_{pcc}(\mathcal{C}_i, \mathcal{C}_s); & \Delta_{jmp}(\mathcal{I}_i, E_i); & \Delta_{dscpl}(\mathcal{I}_i); \\ \Delta_{inpr}(\mathcal{I}_i); & \Delta_{para}(\mathcal{I}_i); & \Delta_{trim}(\mathcal{I}_i, \mathcal{I}_s) \end{bmatrix}$$

We describe each term in the following:

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 same values for states (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 $\xi_v(z)$ and $\xi(z)$ by $T(z)$.

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

2) Natural Language Similarity $\Delta_{nl}(\mathcal{C}_s, \mathcal{C}_i)$: We prefer to use those instruction templates whose natural language clause was similar to the new test one. Therefore, we measure the unordered similarity between objects of $\mathcal{C}_s$ and $\mathcal{C}_i$ by computing their Jaccard index:

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

3) Parameter Similarity Cost $\Delta_{sim}(Z(s), \xi_s, \xi)$: For a given verb clause, we want the objects in the instantiated sequence to be similar to the one in the training set. Therefore, we define:

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

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

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

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) 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 norm of an instruction I as the number of parameters that I takes plus 1.

7) Instruction Prior $\Delta_{inpr}(\mathcal{I}_i)$: We add the prior probability of having an instruction in the sequence. We compute it by counting the number of times it appears in the training set:

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.

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. We define a trimming cost: $\Delta_{trim} = (|\mathcal{I}_s| - |\mathcal{I}_i|)^2$.

B. 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 intent carried by the clause $\mathcal{C}_i$. Thus if the clause intends to *ignite the stove* then the stove should be in *on* state in the E'_i environment. For this we compute the probability table $P(obj, s, v)$ from training data, which gives the probability that obj in clause $\mathcal{C}_i$ could 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.

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

This term is for the instruction that does not correspond to a NL instruction—i.e., its purpose is to handle missing NL instructions. Therefore, it consists of subset of features for ψ_i:

$$\psi_{i\ell}(\mathcal{I}_{i\ell}, E'_{i-1}; w) = w^T\big[\Delta_{jmp}(\mathcal{I}_{i\ell}, E'_i); \quad \Delta_{dscpl}(\mathcal{I}_{i\ell}); \quad \Delta_{inpr}(\mathcal{I}_{i\ell}); \quad \Delta_{para}(\mathcal{I}_{i\ell})\big]$$

D. Term $\psi'_{i\ell}(\mathcal{I}_{i\ell}, E_i; w)$

This consists of a single consistency term:

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

This consistency term is define similarly to Δ_{cons} however, since we do not have a given clause we instead build the table by taking prior over the entire data-set.

E. Inference Procedure

Given the training dataset $\mathcal{D}$, sequence of clauses $\{C\}$ and starting environment E'_0, 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 nodes $\{\mathcal{I}_{i\ell}, \mathcal{I}_i\}_{i\leq j}$ such that the environment at the 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 after executing the sequence as $\Phi(E, \mathcal{I})$.

Our algorithm calls the function **Forward-Inference**(j) (Algo. 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 were created as described in Sec. V. For the edge 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 Sec. V-A which gives us the instruction sequence $\mathcal{I}$. Instantiation is followed by all possible trimming of $\mathcal{I}$ giving us $\mathcal{I}_j = \mathcal{I}[t \cdots]$ for all t. We note here that the *no-op* is also considered when $\mathcal{I}$ is totally trimmed.

```
1  global 𝒟, 𝒞, α
2  function Forward-Inference(j)
3  for each E'_{j-1} such that α_{j-1}[E'_{j-1}] is defined do
4      for each D_s ∈ 𝒟^(j) do
5          ℐ ← instantiate(D_s, 𝒞_j, E'_{j-1})
6          for each t ∈ [0..|ℐ|] do
7              ℐ_j = ℐ[t ···]  // trim the sequence
8              cstr = getConstraints(ℐ_j, E'_{j-1})
9              ℐ_jℓ = callSymbolicPlanner(E'_{j-1}, cstr)
10             E_j = Φ(E'_{j-1}, ℐ_jℓ)
11             E'_j = Φ(E_j, ℐ_j)
12             α_j[E'_j] = min{α_j[E'_j], α_{j-1}[E'_{j-1}]+
               ψ_j(ℐ_j, 𝒞_j, E_j) + ψ'_j(ℐ_j, 𝒞_j, E'_j) +
               ψ_jℓ(ℐ_jℓ, E'_{j-1}) + ψ'_jℓ(ℐ_jℓ, E_j)}
```

Algorithm 1: Forward step during the inference algorithm.

The trimmed sequence $\mathcal{I}_j$ may not be executable due to some missing instructions. To handle this, the function *getConstraints* looks at the pre-conditions 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 [41] which outputs the missing instructions $\mathcal{I}_{j\ell}$ (line 9). 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 12.

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

F. Learning Method

Note the objective function of our model is linear in the weights w, i.e., a local search method converges to the global optimum. For training in our experiments, we divide the dataset into K-folds for cross-validation, and use stochastic gradient descent to train the weights. In our experiments, we manually set the weight $w_{jmp} = \infty$, since we do not want the inferred sequence to be unexecutable.

VII. Experiments and Results

Data. For evaluating our approach, we have collected a dataset *VEIL-200*. Each data-point consists of a natural language description, state of the environment, ground-truth instruction sequence, and the mapping between phrases of each description to the instruction sub-sequence. The natural language describes the high level task but does not necessarily have exact knowledge of the environment (e.g., it may refer to the 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. Please see the project website `http://tellmedave.cs.cornell.edu` for few examples from our dataset. The dataset is then processed to form the VEIL library (see Section V).

In order to collect the ground-truth instruction sequence, we developed an online simulator (motivated by [30]). Our online simulator consists of a virtual 3D environment where the user controls the robot in first-person perspective. When the natural language task is shown to user, user interacts with the environment to accomplish the given task using 12

TABLE II

QUANTITATIVE RESULTS ON FOUR DIFFERENT TASKS ON OUR VEIL DATASET. RESULTS OF BASELINES, DIFFERENT VARIANTS OF OUR METHOD ARE REPORTED ON TWO DIFFERENT METRICS. NOTE THAT WE NORMALIZE THE *IED* AND *EED* METRIC (TO 100) SUCH THAT LARGER NUMBERS ARE BETTER.

	making coffee		*boiling water*		*making ramen*		*making affogato*		Average	
	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
Predefined Templates [18]	14.3	22.8	21.5	38.8	32.8	28.7	14.0	17.0	20.7	26.8
Instruction-Tree [7]	14.4	25.7	11.2	38.8	26.3	25.6	12.5	15.9	16.1	26.5
No NL Instructions	39.2	43.9	21.8	43.3	1.2	15.7	12.8	11.5	18.8	28.6
Nearest Environment Instantiation	63.4	47.3	43.5	50.1	52.1	51.4	45.4	49.2	51.1	49.5
Our model, No Domain Knowledge	78.6	71.7	55.7	55.2	66.1	49.5	51.2	51.3	62.9	56.9
Our model, No latent nodes.	**78.9**	73.8	**57.0**	58.4	66.3	**53.9**	51.5	51.2	**63.4**	59.3
Our full model.	76.5	**78.3**	54.5	**61.3**	**67.3**	49.4	**53.5**	**56.2**	63.0	**61.3**

different instructions. User can point on objects to see their states (e.g., temperature of water) and accordingly control the robot. Please see a screencast video at the project website.

We considered four tasks: *Boiling Water*, *Making Coffee*, *Making Ramen* and *Making Affogato*. Each task was performed in 5 different environments, and each had 50 example runs each. For the first two, we only gave 10 "general" natural instructions ($5 \times 10 = 50$). This allowed us to evaluate whether our algorithm can ground natural language instructions in different environments. For the last two, there was different natural language for each of the 50 datapoints. During training, we use 10-fold cross validation, with first two tasks trained together and then next two tasks trained together. 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 the string edit distance, *Levenshtein distance*, for measuring the syntactical metric distance between two sequences of instructions. 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 well represent the cases where one wrong instruction in the middle could completely change the resulting state. For instance, it is not acceptable to forget filling the pot with water while making ramen, even if rest of the sequence is correct.

2) Environment Edit Distance (EED). This metric relies on that *a successful execution of a task given the language is the attainment of a sequence of states for concerned objects in a possibly time-ordered way*. We compute the edit distance between the ground-truth sequence of environments $(E^g_k)^m_{k=0}$ and the predicted sequence of environments $(\hat{E}_k)^n_{k=0}$. There are two subtle issues: finding the set of concerned objects ∂, and finding the correct representation of difference, $\text{dist}_{env}(E^g_i, E^a_j)$, in two states of environments (e.g., closing the microwave after the use is irrelevant to the task).

We use the list of 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 two environments completely agree on the state of ∂. The recursive equation for computing EED is given below, where $EED(.,.,i,j)$ represents the distance between $(E^g_k)^m_{k=i}$,$(\hat{E}_k)^n_{k=j}$

$$\text{EED}((E^g_k)^m_{k=0}, (\hat{E}_k)^n_{k=0}, i=0, j=0) =$$
$$\min\{\text{EED}(\cdot,\cdot,i,j+1), \text{EED}(\cdot,\cdot,i+1,j) + \text{dist}_{env}(E^g_i, \hat{E}_j, \partial)\}$$
$$\text{where EED}(\cdot,\cdot,i,j) = m-i \text{ if } i=m \text{ or } j=n$$

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

Baselines. We compare our model against the following:

- *Chance*: Randomly choose the instructions of a predefined length. This shows how large the output space is.
- *Predefined Templates, based on Guadarrama et al. [18]* We developed a method similar to [18] in which we manually created a library of proposal-templates for every verb. For a given situation we disambiguate the objects and substitute these parameters in the templates. We extend [18], by also considering many-many mappings.
- *Instruction-Tree Search, based on Bollini et al. [7]*: 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 of word in sentence and an action). Syntactical constraints on instruction are used for pruning and gradient descent is used for training.
- *Nearest Environment Instantiation*: This method looks up the nearest template by minimizing the distance between given environment and that of the template, from the VEIL library, instantiates it according to the new environment. following Section V-A.
- *Our model - No NLP.* Our model in which we do not give the model the natural language, i.e., the given clauses $\mathcal{C}$ are missing. However, all the other terms are used.
- *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 symbolic planner based approaches.) For this we disable the latent, jump features and post-condition features.
- *Our model - No latent modes.* 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 II shows our results. We note that the chance performance is very low because the output space (of the sequence of instructions) is very big, therefore 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 II shows that the method

Fig. 4. **Robot Experiment.** Given the language instruction for making the dessert 'Affogato': *'Take some coffee in a cup. Add icecream of your choice. Finally, add raspberry syrup to the mixture.'*, our algorithm outputs a sequence that the PR2 executes to make the dessert. (Please see the video.)

Predefined Templates [18] focused on disambiguating spatial relations but was extremely brittle to ambiguity in grounding, therefore giving low performance.

Method *Instruction-Tree [7]* was able to give reasonable results on some sequences. However this approach has problem working with large search tree. Furthermore, the bag-of-word feature do not take into account the environment context, the language might say that 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 sequences for such natural language instructions. Also on an additional created data-set for different tasks in a *living room*, we received similar results with our full model outperforming the others.

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. We see that *Our Model - No NLP* gives 18.8% on average as compared to 63.0% 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 II shows the results improve by about 2% on EED metric. We found that it was especially helpful in scenarios where instructions were partially missing. For example, for the instruction in Fig. 1 -

'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 ($\texttt{place}, [\texttt{pot}, \texttt{tap}], \texttt{on} : \texttt{pot} \rightarrow \texttt{tap}$). One such template after initialization has the form -

$$\texttt{moveTo}(\texttt{sink});\ \texttt{keep}(\texttt{pot}, \texttt{sink}, \texttt{on})$$

However this will make the sequence unexecutable as robot cannot execute this sequence since it is not already grasping the pot. In such cases, interpolation models these constraints and we get the output using latent nodes as -

$$\texttt{moveTo}(\texttt{pot});\ \texttt{grasp}(\texttt{pot});\ \texttt{moveTo}(\texttt{sink});\ \texttt{keep}(\texttt{pot}, \texttt{sink}, \texttt{on})$$

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 environment and tasks. Its not obvious whether the templates learned for one task, such as *making affogato* will be useful for another task such as *making ramen*. For studying the effect of a different task, we performed another experiment in which we trained and tested the model on *making ramen* task only (instead of training together for {making ramen,making affogato}). We found that because the VEIL library from the *making affogato* task was not available for training, the performance dropped to 64.9 on the *IED* metric as compared to 67.3 in Table II. This indicates data examples from other tasks are helpful.

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, doing the interpolation and in inference) In order to test how crucial it is, we ran the *Our Model - No Domain Knowledge* and as the results in Table II, show the accuracy falls only by only 2-3 %.

Robot Experiment. We show that our grounded manipulation instructions can be executed on PR2 robot given the natural language instruction, *'Take some coffee in a cup. Add ice cream of your choice. Finally, add raspberry syrup to the mixture.'* Figure 4 shows few snapshots of PR2 making Affogato and the video is available at: `http://tellmedave.cs.cornell.edu`

VIII. Conclusion

In this work, we presented a model that grounds the free-form natural language instructions into a given environment for a given task, in order to output a sequence of instructions that the robot can execute to perform the task. We presented a learning model that encodes certain desired properties into an energy function—expressed as a model isomorphic to conditional random field with edges representing the relations between verb clauses, environment state and instructions. We showed that our model handles missing or incomplete language instructions, variations in language, as well as ambiguity in grounding well. We also show that we outperform related work in this area.

Acknowledgement

We thank Claire Cardie for useful discussions and Kejia Tao for her help with the simulator. This work was supported in part by ONR Grant N00014-14-1-0156, and Microsoft Faculty Fellowship and NSF Career award to Saxena.

References

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

[2] A. Anand, H. Koppula, T. Joachims, and A. Saxena. Contextually guided semantic labeling and search for 3d point clouds. *IJRR*, 2012.

[3] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez. Manipulation with multiple action types. In *Exp Robo.*, pages 531–545, 2013.

[4] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mosenlechner, D. Pangercic, T. Ruhr, and M. Tenorth. Robotic roommates making pancakes. In *Humanoids*, 2011.

[5] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, 2013.

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

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

[8] M. Cakmak, M.R. Dogar, E. Ugur, and E. Sahin. Affordances as a framework for robot control. In *EpiRob*, 2007.

[9] C. Chao, M. Cakmak, and A. Thomaz. Towards grounding concepts for transfer in goal learning from demonstration. In *ICDL*, 2011.

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

[11] V. Chu, I. McMahon, L. Riano, C. McDonald, Q. He, J. Perez-Tejada, M. Arrigo, N. Fitter, J. Nappo, T. Darrell, et al. Using robotic exploratory procedures to learn the meaning of haptic adjectives. In *IROS*, 2013.

[12] F. Duvallet, M. R. Walter, T. Howard, S. Hemachandra, J. Oh, S. Teller, N. Roy, and A. Stentz. Inferring maps and behaviors from natural language instructions. In *ISER*, 2014.

[13] F. Endres, J. Trinkle, and W. Burgard. Learning the dynamics of doors for robotic manipulation. In *IROS*, 2013.

[14] A. Farhadi, I. Endres, and D. Hoiem. Attribute-centric recognition for cross-category generalization. In *CVPR*, 2010.

[15] A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth. Every picture tells a story: Generating sentences from images. In *ECCV*, 2010.

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

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

[18] S. Guadarrama, L. Riano, D. Golland, D. Gouhring, Y. Jia, D. Klein, P. Abbeel, and T. Darrell. Grounding spatial relations for human-robot interaction. In *IROS*, 2013.

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

[20] Y. Jiang, M. Lim, C. Zheng, and A. Saxena. Learning to place new objects in a scene. *IJRR*, 31(9), 2012.

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

[22] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *ICRA*, 2011.

[23] D. Klein and C. Manning. Accurate unlexicalized parsing. In *ACL*, 2003.

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

[25] H. Koppula and A. Saxena. Anticipating human activities using object affordances for reactive robotic response. In *RSS*, 2013.

[26] H.S. Koppula, A. Anand, T. Joachims, and A. Saxena. Semantic labeling of 3d point clouds for indoor scenes. In *NIPS*, 2011.

[27] H. Kress-Gazit, G. Fainekos, and G. Pappas. From structured english to robot motion. In *IROS*, 2007.

[28] OB Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. *RAS*, 58 (9):1105–1116, 2010.

[29] J. Kulick, M. Toussaint, T. Lang, and M. Lopes. Active learning for teaching a robot grounded relational symbols. In *IJCAI*, 2013.

[30] L. Kunze, A. Haidu, and M. Beetz. Acquiring task models for imitation learning through games with a purpose. In *IROS*, 2013.

[31] S. Lemaignan, R. Ros, E. A. Sisbot, R. Alami, and M. Beetz. Grounding the interaction: Anchoring situated discourse in everyday human-robot interaction. *IJSR*, 4(2):181–199, 2012.

[32] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. In *RSS*, 2013.

[33] D Marco, M Tenorth, K Häussermann, O Zweigle, and P Levi. Roboearth action recipe execution. In *IAS*, 2012.

[34] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox. Learning to parse natural language commands to a robot control system. In *ISER*, 2012.

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

[36] S. Niekum, S. Chitta, A. Barto, B. Marthi, and S. Osentoski. Incremental semantically grounded learning from demonstration. In *RSS*, 2013.

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

[38] H. Poon. Grounded unsupervised semantic parsing. In *ACL*, 2013.

[39] V. Ramanathan, P. Liang, and L. Fei-Fei. Video event understanding using natural language descriptions. In *ICCV*, 2013.

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

[41] J. Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 2012.

[42] R. Ros, S. Lemaignan, E. A. Sisbot, R. Alami, J. Steinwender, K. Hamann, and F. Warneken. Which one? grounding the referent based on efficient human-robot interaction. In *RO-MAN*, pages 570–575, 2010.

[43] S. Srinivasa, D. Ferguson, C. Helfrich, D. Berenson, A. Collet, R. Diankov, G. Gallagher, G. Hollinger, J. Kuffner, and M. Weghe. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5–20, 2010.

[44] J. Sung, B. Selman, and A. Saxena. Learning sequences of controllers for complex manipulation tasks. In *IROS*, 2014.

[45] S. Tellex, T. Kollar, S. Dickerson, M. R Walter, A. G.l Banerjee, S. J. Teller, and N. Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

[46] S. Tellex, R. A. Knepper, A. Li, T. M. Howard, D. Rus, and N. Roy. Assembling furniture by asking for help from a human partner. In *HRI*, 2013.

[47] M. Tenorth, L. Kunze, D. Jain, and M. Beetz. Knowrob-map-knowledge-linked semantic object maps. In *Humanoids*, 2010.

[48] M. Walter, S. Hemachandra, B. Homberg, S. Tellex, and S. Teller. Learning semantic maps from natural language descriptions. In *RSS*, 2013.

[49] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon control for temporal logic specifications. In *HSCC*, 2010.

[50] C. Wu, I. Lenz, and A. Saxena. Hierarchical semantic labeling for task-relevant rgb-d perception. In *RSS*, 2014.

[51] H. Yu and J. Siskind. Grounded language learning from videos described with sentences. In *ACL*, 2013.