Grounding English Commands to Reward Functions

***Abstract*—As intelligent robots become more prolific, methods to make interaction with the robots more accessible will become increasingly important. Communicating the tasks that a person wants the robot to carry out via natural language and training the robot to ground the natural language through demonstration are especially appealing approaches for interaction since they do not require a technical background. In this work, we present a system that can learn how to ground natural language commands into reward functions from demonstrations of different natural language commands being carried out in the environment. Because language is grounded to reward functions, rather than explicit actions that the robot can perform, commands can be high-level and carried out in novel environments autonomously and even transfered to other robots with different actions spaces. We demonstrate that our learned model can both be generalized to novel environments and be transfered to a robot with a different action space than the action space used during training.**

I. Introduction

For robots and other intelligent agents to be useful to the general public, they need to be able to autonomously carry out complex tasks. However, it is equally important for humans to be able to communicate a desired complex task to an agent, ideally using natural language commands instead of a formal machine-oriented task representation. In this work, we present novel methods for training an agent to learn the groundings of natural language commands to high-level task descriptions using training data that consists of pairs of natural language commands and demonstrations of the command being executed. While data that paired the actual task representation with the natural language commands would be a more direct and easier learning problem, allowing data to be provided by demonstration is more user friendly since providing demonstrations requires far less technical knowledge of the internal task representations. To learn groundings to task descriptions from example demonstrations, we introduce a generative model of tasks, behavior, and commands that enables *Inverse Reinforcement Learning* (IRL) [14] techniques to identify the set of likely tasks intended by each demonstration and use them train the language model with weakly supervised learning.

Our task model represents task descriptions as abstract *Markov decision process reward functions* that are defined using propositional features of the world. Defining tasks as abstract reward functions is useful because reward functions can induce complex multi-step behavior without having to specify the steps of that behavior. As a result, the natural language commands can be quite general. For instance, the command "bring me a cup of coffee" would give rise to a reward function that motivates the agent to choose steps for getting the coffee and bringing it to the user, without the user specifying any of those details in the command. A significant advantage of this generality compared to grounding language to actions the agent can perform is that the same command can be given in a variety of different environments. Furthermore, training can be performed with one robot and then transferred to different robot that has different primitive actions as long as the high-level representation of the task is consistent.

JM: Stefanie, in addition to the related work section, I imagine here (or perhaps worked into the above paragraph) might be a good place to contrast our approach with existing work; particularly with respect to grounding to reward functions versus actions. Can you write something up about that?

We investigate two language models: a Bag-of-Words mixture model and the IBM Model 2 machine-translation model [4, 5]. While we suspect that other language models that incorporate grammar knowledge might yield better performance and robustness, an advantage of these grammar-free models is that they do not require any additional corpus training and can be used with our task and behavior model as is. Furthermore, an advantage of our generative model and approach is that the language model can be replaced with a different language model without much work, if a specific language model is needed for a domain. In future work, we plan to explore the benefits of using more complex language models with our task and behavior model.

We empirically validate our approach with a simulated version of a "Cleanup World" domain where an agent may be tasked with going to different rooms or taking certain objects to different rooms. Next we show the learned model can be transfered to an actual robot with a different action space successfully. Two main datasets were created for our domain; one with clean natural language commands generated from an author of the paper to demonstrate a range of tasks and environments that we can support and another that gathered language from untrained Amazon Mechanical Turk (AMT) users designed to elicit varying language. A typical AMT task is to take a star block into an orange painted room; one data instance for this task is the command "go into tan room and push star into orange room" with a sequence of states and actions demonstrating the completion of the task. Note that this demonstration consists of many primitive movement actions, but the provided command does not describe each action that the agent needs to take, only the high-level goal. Performance is evaluated using leave-one-out cross validation for each command and we compare results to a baseline that grounds language to actions. We also separately evaluate performance when the test command is given on a set of environments never seen in the training dataset. We found that our task grounding model performs well both when the test environments were the same as the training environments and when the environments

were novel. In contrast, the action grounding model is only able to perform well in the known environments and fails in novel environments.

II. Background

To represent tasks, we make use of the *Object-oriented Markov Decision Process* (OO-MDP) formalism [8]. OO-MDPs extend the conventional MDP formalism by providing a rich state representation. MDPs are defined by a five-tuple: ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$, $\mathcal{F}$), where $\mathcal{S}$ is a set of states of the world; $\mathcal{A}$ is a set of actions that the agent can take; $\mathcal{T}$ describes the transition dynamics, which specify the probability of the agent transitioning to each state given the current state and the action taken; $\mathcal{R}$ is a function specifying the reward received by the agent for each transition; and $\mathcal{F}$ is a set of terminal states that cause action to cease once reached. The goal of planning in an MDP is to find a *policy*—a mapping from states to actions—that maximizes the expected discounted cumulative reward.

An OO-MDP extends the classic MDP formalism by including a set of *object classes*, each defined by a set of attributes, and a set of *propositional functions* whose parameters are typed to object classes. A state in an OO-MDP is a set of objects that each belong to one of the possible object classes; each object has its own state, which is represented as a value assignment to the attributes of its associated object class. The propositional functions defined in the OO-MDP are functions of the object states being evaluated. For example, consider a blocks world in which block objects are defined by their position in the world. The propositional function evaluation $\texttt{on}(b_1, b_2)$, where b_1 and b_2 are block objects in a state, returns true when the position of b_1 is adjacent and above the position of b_2 and false otherwise. Although an OO-MDP state is fully defined by the union of objects, these propositional functions provide additional high-level information about the state. In this work, we use propositional functions to define abstract task definitions. Note that although OO-MDPs provide a rich state representation, any standard MDP planning algorithm can be used for planning in an OO-MDP.

III. The Task And Behavior Model

Our goal is to learn how to ground language to task descriptions represented as MDP reward functions indirectly from demonstrations of the tasks being carried out. To perform this learning, we use a form of IRL to identify the set of likely tasks from each demonstration and use the identified tasks and their probability to perform weakly supervised learning of the language model. Once the grounding between language and tasks is learned, an agent can receive a command, ground it into a reward function, and then compute a policy to follow to complete the task using any "off-the-shelf" MDP planning algorithm.

To ground language to reward functions and perform IRL from demonstration to those possible reward functions, we introduce a generative model of factored reward functions, behavior, and language. More specifically, we assume that any *task* that a person may request can be represented as a reward function and corresponding terminal states (if it is a terminating task) defined with OO-MDP propositional functions. Everything about the OO-MDP, except the task, is assumed to be known by the agent. Our generative model, shown in Figure 1, consists of an input initial state (S), a set of lifted tasks (L), a set of grounded tasks (T), a set of object binding constraints (C), a set of behavior demonstrations/trajectories (B), and a language model that produces natural language commands and is dependent on the lifted task and binding constraints. A lifted task specifies a kinds of problems the agent can solve with propositional functions, but leaves the OO-MDP objects on which it operates as free variables; a grounded task binds a lifted task's free variables to specific objects in the world thereby producing a well-defined decision making problem; object binding constraints specify the set of possible properties of the grounded task's bound objects that a user might refer to in language. In the rest of this section we describe these various components and their probability distributions in more detail. As a running example, we will refer to the example Cleanup World state shown in Figure 2. Note that the specific language model is a parameter of our overall model and learning algorithm. In the following section we will discuss two grammar-free models that we explored that were successful in grounding to Clean-up World tasks.

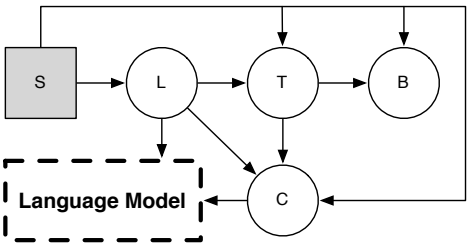

Fig. 1. The generative task model, with arrows indicating conditional probabilities. S is the current state and is an input to the model. L is the lifted task which specifies the kind of problem to solve. T is the grounded tasks which binds specific objects to the lifted task making for a well defined problem to solve. B is the behavior the agent follows to complete task T. C is the object binding constraints which specifies properties about the bound objects in T that a user might refer to in their language to specify the objects. The specific language model is a parameter of our overall language model.

A. Lifted Tasks

The set of possible lifted task definitions are provided by a designer and represent the kinds of tasks an agent could conceivably carry out in an environment. Specifically, each lifted task is a factored reward function defined as a logical expression of OO-MDP propositional functions with the parameters of the propositional functions set to free variables. In our home environment (Figure 2), a reward function for the task of taking a "block" object (*e.g.*, chair) to a room is:

$$\mathcal{R}(s, a, s') = \begin{cases} 1 & \text{if } \texttt{blockInRoom}^{s'}(?b, ?r) \\ 0 & \text{otherwise} \end{cases}, \qquad (1)$$

where s' is the outcome state for the agent after taking action a in state s, $\texttt{blockInRoom}^{s'}$ is a propositional function evaluated

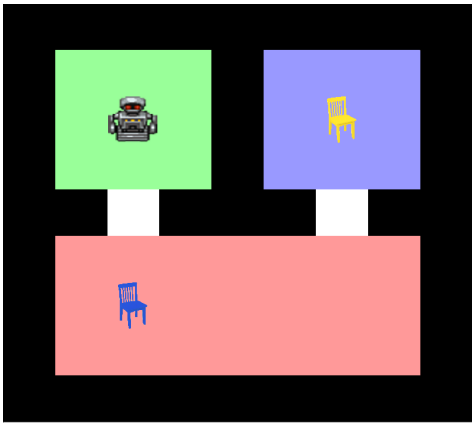

Fig. 2. An example three-room floor layout with one robot and two chairs. The object references for the red, green, and blue rooms are r_1, r_2, and r_3, respectively. The references for the yellow and blue chair are c_1 and c_2, respectively.

in state s', $?b$ is a free variable that can be ground to any movable block object in the environment, and $?r$ is a free variable that can be ground to any room in the environment. If a task is goal directed, then the reward function will also be paired with a similar set of termination conditions.

The prior probability distribution on the set of lifted tasks is assumed to be uniform over the set of lifted tasks that are *permissible* in an input state. A lifted task is permissible in an input state if there exists at least one object of each object class required by the free variables in the lifted task.

B. Grounded Tasks

A grounded task (T) is dependent on the lifted task and input state and is an assignment of objects in the input state to each of the free variables in the lifted task. For example, given the input state shown in Figure 2, an object assignment to lifted task 1 that represents the task of the robot taking the yellow chair to the red room is $?b := c_1$ and $?r := r_1$. The probability distribution of grounded tasks is uniform over the set of possible variable assignments for the lifted task. In goal-directed tasks, only tasks that are not satisfied in the initial state are considered.

C. Object Binding Constraints

If language was only dependent on the lifted task, it would be impossible for the agent to determine which of the possible grounded tasks for an input state was the intended task from a command. Object binding constraints (C) are logical expressions that extend a lifted task and on which language is dependent, enabling the agent to resolve grounding ambiguity. For example, the command "take the yellow chair to the red room," indicates that a grounding for lifted task 1 should satisfy the constraints $\texttt{isYellow}(?b) \wedge \texttt{isRed}(?r)$.

The probability distribution of object constraints given a grounded task, lifted task, and input state is uniform across the set of possible additional logical constraints that are true in the input state for the given variable assignments in the grounded task. For example, if $?b$ is assigned to c_1 in the grounded task, $\texttt{isYellow}(?b)$, is a possible constraint since it is true in the initial state for c_1, but $\texttt{isBlue}(?b)$ is not.

D. Behavior

A *behavior trajectory* is a sequence of state–action pairs that the agent can experience from an input state and is conditionally dependent on the grounded task and input state. The conditional probability of the behavior, given the task, is formulated by treating each action selection in each state of the trajectory as an independent event and by defining the action-selection probability distribution as a Boltzmann distribution over the optimal Q-values for the task—similar to the likelihood of a trajectory in maximum likelihood inverse reinforcement learning [1]. The optimal Q-values are computed using any off-the-shelf MDP planning algorithm. The probability of any behavior trajectory b (of length N) given task t is defined as

$$\Pr(b|t) = \prod_{i}^{N} \pi_t(s_i, a_i), \tag{2}$$

where (s_i, a_i) is the ith state–action pair in behavior trajectory b, and $\pi_t(s_i, a_i)$ is the Boltzmann policy distribution (the probability of taking action a_i in state s_i).

For goal-directed tasks, we add a virtual "terminate" action to the MDP that must be executed to actually terminate the task; this virtual terminate action is added to the end of observed trajectories. The inclusion of a terminating action more strongly differentiates between tasks where the optimal trajectory for one is a subsequence of the optimal trajectory for another. For example, an optimal trajectory for going to the red room is a subsequence of an optimal trajectory for going to the blue room (in Figure 2), but since it is not expected for the agent to terminate in the red room if the task is to go the blue room, terminating in the red room makes it more likely for the trajectory to be generated by a "go to the red room" task.

We next delve more deeply into the model that relates language and tasks.

IV. Language Models

We investigated two different grammar-free language models for use with our task model: a Bag-of-Words model and IBM Model 2, a word-based machine-translation model. In this section, we describe each model and how it integrates with our task model.

A. Bag-of-Words

In the bag-of-words (BoW) model, commands are treated as an unordered bag of words generated from a mixture model of semantic components, similar to previous topic modeling approaches [13]. The semantic components given a lifted task and object binding constraints include (1) the set of propositional function names included in the task and binding constraint expressions, (2) the object class names of the arguments of the propositional functions, and (3) a constant symbol

that acts as a catch all for words not explicitly related to a semantic component. For example, lifted task 1 coupled with the constraint $\texttt{isRed}(?r)$ would have the semantic components $\texttt{blockInRoom}, \texttt{isRed}, \texttt{BLOCK}, \texttt{ROOM}$, and $\#$, where $\#$ is the constant symbol. The probability that any semantic component is selected to generate a word is proportional to its frequency of appearance in the task and binding constraint expressions.

The parameters of the BoW model are the conditional probabilities that any word will be generated from a given semantic component. Therefore, the probability of a command (e) given a lifted task (l) and object binding constraints (c) is

$$\Pr(e|l,c) = \prod_{w \in e} \left[\sum_{v} \Pr(v|l,c)\theta_{vw} \right]^{K(w,e)}, \qquad (3)$$

where θ_{vw} is a parameter specifying the conditional probability that the natural language word w is generated given semantic component v, and $K(w,e)$ is the number of times that word w appears in command e.

A limitation of the BoW models is that, because it does not account for word order or structure, it cannot disambiguate which words in a command apply to which object. For example, for the command "take the blue chair to the red room," BoW would not be able to disambiguate whether "blue" refers to the chair or the room (and *vice versa* for the word "red").

B. IBM Model 2

IBM Model 2 (IBM2) [4, 5] is a word-based statistical machine-translation model. In statistical machine translation, the task is to translate a sentence from a source language f (*e.g.*, French) to a target language e (*e.g.*, English). Our task corresponds to the problem of translating from an English command e to its corresponding machine-language command m.

We integrate IBM2 into our task and behavior model by using the lifted task and object binding constraints to deterministically generate a machine-language command m and then generate the natural language command e from m in the standard IBM2 fashion. A machine-language command (m) is generated from a lifted task (l) and object binding constraints (c) by first adding each semantic component in l to m in the order that the components appear. Next, the same is done for the components of c. For example, the lifted task in Equation 1 and object binding constraint $\texttt{isRed}(?r)$ would generate the machine-language command "# blockInRoom block room isRed room," where # is the constant symbol that IBM2 always assumes present at the start of an expression. The probability of a natural language command (e) given the machine-language command (m) is defined as:

$$\Pr(\mathbf{e}|\mathbf{m}) = \eta(n_e|n_m) \sum_{\mathbf{a}} \prod_{j}^{n_e} q(a_j|j,n_m,n_e) r(e_j|m_{a_j}), \qquad (4)$$

where $\eta(n_e|n_m)$ is the parameter specifying the probability that a machine-language command of length $n_m$would generate a natural language command of length n_e; $\mathbf{a}$ is a possible alignment from natural language words to machine-language words; $q(a_j|j,n_m,n_e)$ is the alignment parameter specifying the probability that the natural language word in position j would be aligned with the machine-language word in position a_j for a machine-language command of length n_m and natural language command of length n_e; and $r(e_j|m_{a_j})$ is the translation parameter specifying the probability that natural language word e_j would be generated from machine-language word m_{a_j}. The number of alignments ($\mathbf{a}$) is typically very large, so in practice we estimate the value using sampling.

C. Learning Language Model Parameters

Given a dataset of demonstration trajectory and natural language command pairs, the language model parameters can be learned through weakly supervised learning.

When the task and behavior model is paired with the BoW language model, we use a standard Bayesian Network expectation maximization (EM) algorithm [7] to iteratively update the BoW model's word generation parameters.

In classic IBM2 parameter learning, EM is used on a training dataset consisting of pairs of natural language expressions (*e.g.*, French and English), often with a bake-in period where initially only translation parameters are updated and then translation parameters and alignment parameters. In our dataset, we have pairs of trajectory and natural language commands, so we use a modified version of the classic IBM2 EM algorithm where we convert each trajectory into a set of possible machine-language commands weighted by the probability that each machine-language command would be generated given the trajectory. Then, the standard IBM2 EM update rules are performed on the pairs of machine-language and natural language commands, except the contribution of each pair is weighted by the probability of that machine-language command having been generated by its associated trajectory. The probability of any machine-language command (m) given a trajectory (b) and initial state (s) is

$$\Pr(m|s,b) = \frac{\sum_{l,t,c} \Pr(l|s)\Pr(t|s,l)\Pr(b|s,t)\Pr(c|s,l,t)\Pr(m|l,c)}{\sum_{l,t} \Pr(l|s)\Pr(t|s,l)\Pr(b|s,t)}. \qquad (5)$$

A similar approach could be used for training the parameters of other language models.

V. Experimental Results

To empirically validate our approach, we tested our model on datasets that collected example language for the "Cleanup World" from two different sources: an Amazon Mechanical Turk (AMT) user study, and and "expert" who is an author of this paper. Different experiments were run for the AMT language source and the expert source. In the AMT study, we asked users to provide example commands for a smaller range of tasks, but used more abstract object representations and unusual color choices to elicit more variety in the command descriptions. In the expert study, we evaluated a larger range of tasks and environments but with fairly clean expressions of the language. The details of the various experiments run with each study are explained in the following subsections. We then also show how we can take the language model learned from the AMT study, recreate the Cleanup World domain on an actual mobile robot and successfully transfer the learned model so that the robot correctly interprets the commands despite the fact that the robot has a different action space than the simulated robot used for training.

Cleanup World (like Sokoban [11]) is a 2D grid world of various rooms connected by open doors. Rooms can also

contain "blocks" that can be moved around. The agent moves using north-south-east-west actions. Moving into a location containing a block causes the block to move in the direction the agent is moving, as long as a wall or other item is not in its immediate path. Cleanup World is represented as an OO-MDP consisting of four object classes: `AGENT`, `BLOCK`, `ROOM`, and `DOORWAY`. The propositional functions defined for the OO-MDP include `agentInRoom(AGENT,ROOM)`, and `blockInRoom(BLOCK,ROOM)`, as well as propositional functions to indicate the color and type of rooms and blocks (*e.g.*, `roomIsOrange(ROOM)`, `blockIsStar(BLOCK)`).

In both studies, the Boltzmann policy parameter τ was set to 0.005 and performance was measured using leave-one-out (LOO) cross validation; a prediction was considered correct if the interpretation of the reward function resulted in behavior that achieved the actual goal.

For a baseline, we compared learning results to a model that grounds language to actions rather than tasks. Because the order of the actions is important, we always paired the baseline with the IBM2 language model. When training the IBM2 model, the action sequence in the demonstration is converted into a machine language sentence in which each word is the name of the action. Note that the action grounding learning problem is in some sense easier than training the task model since the action sequence is always directly observed in the demonstration, whereas in our task grounding model, IRL is needed to perform weakly supervised learning. Inferring an action sequence from a command would generally require searching the action sequence space and finding the most likely sequence from the command or at least using it to bias a planning algorithm's search similar to **JM: Mooney paper**. However, we simplify this inference problem greatly by allowing the action grounding model to exploit our task model. Specifically, when inferring an action sequence for a command, the only valid action sequences permitted are those that solve one of the possible grounded tasks and we let the most likely of the possible sequences be the one chosen by the agent.

A. AMT Dataset

In the AMT study, we considered two different lifted tasks: the agent going to a specific room and the agent moving a block to a specific room. To collect natural language instructions for different grounded versions of these tasks, we presented Turkers on AMT animated images showing either the agent moving to a room of a specific color or moving a star block to a room of a specific color. An example image from the animation is shown in Figure 3. To prevent contamination of the commands we received, we never provided users with any example commands.

After removing labels that did not follow the instructions or which did not describe the task in any capacity, we obtained a dataset of 240 instances. The goal of our work is to be able to give autonomous agents high-level commands that leave the details of how to complete the task as a problem for the agent to solve. However, most of the natural language commands we received included some high-level details of the path the agent followed (*e.g.*, subgoals), rather than only describing the task goal. For example, one such command was "go through center opening into beige enclosure and get behind star and push it into opening of orange enclosure." Although this data is interesting because it tests our model's performance on language that it wasn't intended to model, a dataset that better reflects the problem we were trying to solve is also useful for comparison. Therefore, in addition to this source dataset, we created a simplified version that omits the text that is extraneous to the task description. For instance, the previous example became: "go through center opening and get behind star and push it into opening of orange enclosure." The average command lengths (in words) were 13.57 and 8.87 in the original and simplified datasets, respectively.

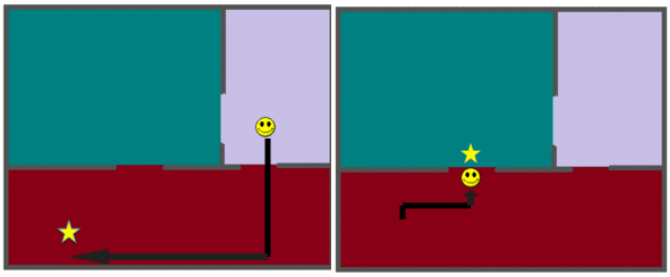

Fig. 3. An example task to take a star to a room shown to users on AMT. The left frame shows the initial state and the path to the star; the right frame shows the path from there to the end state. (In the actual study, users were shown a single animated image without arrows.)

In the training data we gathered from users, the room layout was always the same; however, one of the advantages of our approach is commands can generalize to novel environments. To demonstrate the ability for our approach to generalize to new environments, we performed a third dataset in which we used LOO cross validation for the language on the original dataset and tested the command on three different novel environments. Note that these different environments were not merely a different spatial arrangement of the rooms compared to the training dataset; the environments also had different initial conditions for the agent not seen in the training that results in a different set of propositional functions being true in the initial state. As a consequence, our task model could perform worse if during training it over fit the object binding constraints to inappropriate features. We refer to this dataset as to the *novel environment* (NE) dataset

The LOO accuracy for each language model on each variant of the dataset is shown in Table I. Note for comparison that if the agent randomly selected a random grounded task it would achieve an expected accuracy of 37.5% in the simplified and original dataset and 33.3% accuracy in the NE dataset.

On the simplified dataset, both BoW and IBM2 performed well and comparably (according to a chi-squared test, Bow and IBM2 are not statistically different; $p = 0.54$); however, while action grounding was able to perform better than randomly

Model	Simplified Dataset	Original Dataset	NE Dataset
BoW	83.75%	53%	51.8%
IBM2	81.25%	74.1%	73.3%
Action Grounding	69.5%	69.5%	21.9%

TABLE I
LOO ACCURACY FOR THE AMT DATASETS.

guessing, it did not perform as well as BoW or IBM2 and the difference was statistically significant ($p < 0.001$ on a chi-squared test with all three groups).

On the original dataset, both BoW and IBM2's performance dropped, although BoW's drop in performance was much greater than IBM2's drop in performance and of the two, only BoW's drop in performance was statistically significant ($p < 0.01$ and $p = 0.07$ for BoW and IBM2's decrease in performance, respectively). The action grounding baseline performed identically to its performance on the simplified dataset and is reasonably comparable to the task model using IBM2.

On the NE dataset, the task model was able to successfully generalize to the novel environments achieving nearly the same performance as its performance on the original dataset for the corresponding language model. In contrast, the action grounding baseline completely failed to generalize achieving a performance worse than what would be expected from randomly choosing tasks.

To demonstrate that IBM2 successfully learned reasonable groundings of English words, we extracted IBM2's translation parameters after it had finished training on the original dataset. The most likely English words generated from the semantic word "agentInRoom" (which was associated with the lifted "go to room" task), were "walk," "through," "move," "go," and "from." "From" and "through" occurred because in "go to room" tasks, users often described *from* which room to leave and often commanded the robot to go *through* a door to the goal room. For example, one of the commands provided was "walk through doorway from orange room to beige room."

Since the color of rooms was typically used to describe the goal room of both lifted tasks, the words associated with it are especially relevant. The "roomIsOrange" semantic word was mostly likely to generate the words "red" and "orange;" "roomIsTan" was most likely to generate the words "tan" and "beige;" and "roomIsTeal" was most likely to generate "green" and "blue."

Between the the expert and AMT datasets we found that our approach can be used to successfully training a language model that can be used to ground language. Overall, the BoW language model seems like it will only be successful when commands are very simple, whereas IBM2 can handle more complex commands and may continue to improve with more data.

B. Expert Dataset

The expert dataset includes three kinds of goal-directed lifted tasks: one for going to a room; one for taking a block to a room; and one for taking a block to a room and then going to another room. Environments had either one or two blocks present, which could be either a chair or a bag, and three rooms whose colors varied between environments. The dataset consisted of many different verbal descriptions of tasks, which also resulted in different necessary object binding constraints that would need to be inferred. For example, some expressions took the form of "move the bag to the blue room," which requires object binding constraints for the shape of the block (bag) and the color of the destination room. Another example is "move bag to the room with chair," which requires inferring object binding constraints for the shape of the target block, and a `blockInRoom` and a block shape propositional function to disambiguate the target room. Different commands also had variable levels of specificity. For example, some included information that was not necessary to infer the correct task, such as "push chair from blue room to red room," in which specifying the color of the room in which the chair *initially* resides (blue) is unnecessary when there is only one chair. We also included commands that could not be modeled with the propositional functions defined. For example, the command "go to left room," is not representable because there are no propositional functions defined that indicate the relative spatial position of the rooms.

A strength of our framework is that different demonstrations of a command, even suboptimal ones, can be given and still provide meaningful information. In our dataset, we have 35 additional instances of a command given in the same environment, but with a different demonstration. However, to make it so that the LOO evaluation is always inferring behavior from either a novel command or environment, we also created a version of the dataset that did not include any duplicate commands unless they were given in a different environment. We will refer to the dataset without multiple demonstrations in the same environment as *expert* and the dataset with multiple demonstrations as *expert-MD*. The expert dataset has 118 instances, with 104 unique commands (14 of the commands are the same as another, but are given in a different environment); expert-MD has 153 instances.

The LOO performance of the expert datasets is shown in Table II. Note that if the agent randomly select a possible grounded task, it would have an expected performance of 8.7% and 8.6% for the Expert and Expert-MD datasets, respectively. On both the Expert and Expert-MD dataset, our task model with IBM2 performed the best and these differences were statistically significant ($p = 0.013$ on a chi-squared test with all three groups). Furthermore, IBM2 performed better on Expert-MD than Expert (and the differences were statistically significant with $p = 0.011$), whereas BoW and Action Grounding did not improve performance by any significant amount, suggesting that even better performance may be possible with additional data, even if that data uses different demonstrations

Model	Expert	Expert-MD
BoW	48.3%	52.2%
IBM2	65.2%	79.7%
Action Grounding	49.1%	49.6%

TABLE II
LOO ACCURACY FOR THE EXPERT DATASETS.

than prior examples.

C. Transferring the Learned Model to a Different Robot

One of the strengths of grounding language to high-level tasks is that the learned model can be transferred to different robots with a different set of actions, because each robot can plan independently for each reward function. We demonstrate this transferability after training in our simulated version of the Cleanup World by taking the learned model from our simulated domain and executing it on an actual mobile robot whose action space consists of three actions: turning 90 degrees clockwise or counterclockwise and moving forward a fixed distance, rather than moving north, south, east, or west as in our simulator in which training was performed. To execute a policy, the robot needed to know its location and the location of block objects in the world, which we determined by using a motion tracking camera system. A video of the robot executing the AMT gathered command "" can be found at **JM: video URL and command used** and an image of it following a command is shown in Figure **??**.

VI. Related Work

Our work relates to the broad class of methods for grounded language learning that aim to ground words from a situated context. Instead of using annotated training data consisting of sentences and their corresponding semantic representations [12, 17, 20, 19], most of these approaches leverage non-linguistic information from a situated context as their primary source of supervision.

These approaches have been applied to various tasks, the one closest to ours being interpreting verbal commands in the context of navigational instructions [16, 6, 10], robot manipulation [9, 15], and puzzle solving and software control [3]. Reinforcement learning has been applied to train a policy to follow natural language instructions for software control and map navigation [16]. However, our goal is to move away from low-level instructions that correspond directly to actions in the environment to high-level task descriptions expressed using complex language. Unlike previous approaches that learn word groundings from pairs of natural language instructions and demonstrations of corresponding high level actions [15, 9], our method learns mappings of natural language instructions to task descriptions represented as MDP reward functions, enabling the agent to be autonomous and creative in its behavior. In addition, our generative model allows an investigation of multiple language models that can be used with the task model. Besides the generally used Bag-of-Words model [2, 16], we showed that a word-based statistical machine-translation model provides better results. The idea of using statistical machine translation approaches for semantic parsing was introduced by Wong and Mooney [17] in a supervised learning setting. In future work, we plan to move beyond word-based SMT models to grammar-based SMT models such as Synchronous Context-Free Grammars [18].

VII. Conclusions and Future Work

We presented a novel generative task model that expresses tasks as factored MDP reward functions and terminal states, and to which natural language commands can be grounded. This generative task model is flexible and can be combined with different language models. We further showed how our model enables task groundings to be learned indirectly from example demonstrations by using IRL to weakly supervise the language model learning. Grounding natural language commands to MDP tasks rather than actions is a powerful approach because it allows people to provide robots high-level commands without specifying the details of how to complete the tasks. This task grounding enables the robot to correctly follow a command in a variety of environments, including environments it's never previously encountered as shown in our results. Moreover, grounding language to tasks allows the learned model to be transfered to different robots that have different actions spaces than the robot used to train the model, which was also demonstrated by transferring the learned language model onto a physical robot with a different action space than the action space of the simulated robot used in training.

In this work, we investigated two grammar–free language models that can be combined with our task model: a Bag-of-Words (BoW) mixture model and IBM Model 2 (IBM2). We empirically validated our approach with these two language models on an expert developed dataset with a variety of tasks and a dataset in which commands were provided by users of the Amazon Mechanical Turk. We found that when commands are simple and can be fully represented by the words that are present in the command, both the BoW model and IBM2 are effective. However, when the correct interpretation of the commands depends more heavily on the structure of the sentences, IBM2 is more effective than BoW. Grammar-free models have the advantage that they do not require a corpus for training the grammar model and they were sufficient for successfully grounding the language to tasks in our test domain. However, we imagine there are other domains in which more complex grammar-based language models could provide better results. Because our task and behavior model is modular, we believe other language models could be incorporated relatively easily.

In the future, we will extend this approach to operate on a wider variety of task types. For instance, logical quantifiers such as "for all" and "there exists" can be used with the OO-MDP propositional functions to define more expressive tasks; reward functions that include penalties for certain behaviors can also be incorporated. In complex environments, planning for a task can be difficult and expensive, so another future

direction is to incorporate into our model the ability for a command to specify subtasks, which would make the overall planning problem more tractable for the robot.

REFERENCES

[1] Monica Babeş-Vroman, Vukosi Marivate, Kaushik Subramanian, and Michael Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the Twenty Eighth International Conference on Machine Learning (ICML 2011)*, 2011.

[2] S. R. K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, 2009.

[3] S.R.K. Branavan, Luke S. Zettlemoyer, and Regina Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *Association for Computational Linguistics (ACL 2010).*, 2010.

[4] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Comput. Linguist.*, 16 (2):79–85, June 1990. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=92858.92860.

[5] Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Comput. Linguist.*, 19(2):263–311, June 1993. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=972470.972474.

[6] David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-2011).*, pages 859–865, 2011.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[8] Carlos Diuk, Andre Cohen, and Michael Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, 2008.

[9] Felix Duvallet, Thomas Kollar, and Anthony (Tony) Stentz. Imitation learning for natural language direction following through unknown environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2013.

[10] Alexander Grubb, Felix Duvallet, Stephanie Tellex, Thomas Kollar, Nicholas Roy, Anthony Stentz, and J. Andrew Bagnel. Imitation learning for natural language direction following. In *Proceedings of the ICML Workshop on New Developments in Imitation Learning.*, 2011.

[11] Andreas Junghanns and Jonathan Schaeffer. Sokoban: A challenging single-agent search problem. In *In IJCAI Workshop on Using Games as an Experimental Testbed for AI Reasearch*. Citeseer, 1997.

[12] Rohit J. Kate and Raymond J. Mooney. Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, 2006.

[13] Andrew McCallum. Multi-label text classification with a mixture model trained by em. In *AAAI'99 Workshop on Text Learning*, pages 1–7, 1999.

[14] Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann, 2000.

[15] Stefanie Tellex, Pratiksha Thaker, Joshua Joseph, and Nicholas Roy. Learning perceptually grounded word meanings from unaligned parallel data. *Machine Learning*, 94(2):205–232, 2014.

[16] Adam Vogel and Dan Jurafsky. Learning to follow navigational directions. In *Association for Computational Linguistics (ACL 2010).*, 2010.

[17] Yuk Wah Wong and Raymond Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*, 2007.

[18] Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403, 1997.

[19] Luke Zettlemoyer and Michael Collins. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Association for Computational Linguistics (ACL'09)*, 2009.

[20] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of UAI-05*, 2005.