Robot Programming by Demonstration with Situated Spatial Language Understanding

Maxwell Forbes, Rajesh P. N. Rao, Luke Zettlemoyer and Maya Cakmak

***Abstract*— Robot Programming by Demonstration (PbD) allows users to program a robot by demonstrating the desired behavior. Providing these demonstrations typically involves moving the robot through a sequence of states, often by physically manipulating it. This requires users to be co-located with the robot and have the physical ability to manipulate it. In this paper, we present a natural language based interface for PbD that removes these requirements and enables hands-free programming. We focus on programming object manipulation actions—our key insight is that such actions can be decomposed into known types of manipulator movements that are naturally described using spatial language; *e.g.,* object reference expressions and prepositions. Our method takes a natural language command and the current world state to infer the intended movement command and its parametrization. We implement this method on a two-armed mobile manipulator and demonstrate the different types of manipulation actions that can be programmed with it. We compare it to a kinesthetic PbD interface and we demonstrate our method's ability to deal with incomplete language.**

I. INTRODUCTION

General-purpose robots that can manipulate everyday object could take on many useful tasks in human environments. However, programming such robots to robustly function in *every* possible environment, with *every* possible user is extremely challenging. Instead, our research seeks to develop robots that can be programmed by their end-users to function in their particular environment.

Programming by Demonstration (PbD) [5], [3] is an intuitive method that enables users to program new capabilities on a robot simply by demonstrating the desired behavior. Kinesthetic teaching (*i.e.,* physically moving the robot through desired states) [11], [12], [1], [8] allows users to directly demonstrate desired arm configurations and motions at a fine grained level and is often the most efficient interface for PbD. However, it requires physical effort from the user and it is not accessible to persons with certain motor impairments. Furthermore, it requires the user to be co-located with the robot. In this paper, we present a natural language interface for programming a robot to perform manipulation tasks purely through verbal commands. We exploit the fact that task-oriented motion is often relative to landmarks in the environment and that it can be described with spatial language (*e.g.,* object reference expressions and prepositions). We implement a set of robust parametrized motion procedures on a PR2 mobile manipulator and we present a situated language understanding model that takes a natural language utterance and infers the intended procedure and its parameter instantiation. We demonstrate that our method allows programming different manipulation tasks, even with ambiguous language and is on par with kinesthetic programming in terms of efficiency.

The authors are with the Computer Science and Engineering Department, University of Washington, Seattle, WA 98195, USA.

II. RELATED WORK

Natural language interfaces to robots have seen several recent developments. Tellex et al. used syntactic parsing to construct a graphical model that grounds language into actions executed on a robotic forklift (*e.g., pickup, move, place*). Their approach involves learning from a corpus of data [17]. Jia et al. developed a tabletop manipulation system that takes natural language commands as input and uses visual adjectives to distinguish between objects. They demonstrate their approach with the *pick up* command [10]. Misra et al. developed a system that learns how to ground language instructions in actions from a corpus of pairs of natural language and virtual task demonstrations [15].

In the context of *programming by instruction* Mohan et al. developed an explanation-based task learning approach. They utilize domain knowledge along with situated instructions to teach a robot novel hierarchical tasks. The programming is interactive, but it is conducted in advance of execution [16]. Our system executes actions incrementally as they are programmed. Brick et al. proposed a model of incremental language processing as a foundation for natural language interactions in human-robot interaction [6].

Recent work in natural language processing has explored semantic parsing as an interface between language and robotic commands, as well as a tool for generating and understanding object referring expressions. Artzi et al. show a system that uses semantic parsing to ground natural language text into lambda calculus expressions that fully specify the task to be executed by the robot. Their approach was only tested virtually and in the navigation domain, whereas our uses a much richer set of actions on a robot [4]. Matuszek et al. also use semantic parsing to recover meaning from language. They address the language grounding problem with a joint language and perception model that attempts to generate ground object referring expressions to locations in an image [14]. FitzGerald et al. utilize a semantic parser to generate referring expressions for objects using a more effective inference technique for learning than in Matuszek's work. They generate a lambda-calculus description of the predicted properties of the objects in an image [9].

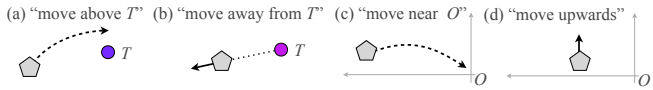

Fig. 1. Illustration of four types of movements: (a) relative destination, (b) relative direction, (c) absolute destination, and (d) absolute direction.

III. APPROACH

This paper demonstrates that it is possible to program robotic manipulation capabilities using natural language utterances. This section describes our approach, by first defining the space of possible commands the robot can execute (Sec. III-A and III-B) and a generative probabilistic model for providing natural language commands to the robot in a situated context (Sec. III-C). We then describe in more detail the individual parts of this model, including distributions for a prior model of what commands are most likely in different situations (Sec III-E) and a generative model for producing contextually appropriate command descriptions in natural language (Sec. III-F). Finally, we describe how we do efficient inference to understand user utterances during actual interactions (Sec III-H).

A. Robot motion commands

Our work builds on Alexandrova et al.'s PbD framework in which actions are represented as a sparse sequence of end-effector poses relative to landmarks in the environment [2]. Alexandrova et al.'s implementation involves kinesthetically moving the robot's arms to the desired end-effector configurations and saving poses through verbal commands. In this paper, we replace the physical interaction with verbal commands. Our key insight is that such task-oriented motion can be decomposed into known types of manipulator movements that are naturally described using spatial language.

We identify two dimensions in which manipulator motions can be categorized.

- *Absolute versus relative:* Most tasks-oriented actions involve moving relative to objects that are part of the action (*i.e.,* they are relative). However, some motions are independent from the objects in the environment (*i.e.,* they are absolute); instead they are relative to the robot's body or to absolute entities (*e.g.,* gravity).
- *Destination versus direction:* While some motions are identified with a target destination (with no constraints on how it is reached), others are identified as directions that correspond to differential movements.

These result in four types of movements illustrated in Fig. 1. Destinations are specified with prepositions of place (above, near, on, to, *etc.*), whereas directions are specified with prepositions of direction (towards, away from, up, down, *etc.*). Relative movements require referencing landmarks in the environment that are perceivable to the robot, whereas absolute movements require known places or directions that can be referenced with a common name.

In this work, the four types of movements are implemented as separate parametrized procedures that are candidates that can be commanded with natural language. All distances for relative and absolute movements are fixed; they are chosen experimentally to balance speed of execution with accuracy of movements. We assume a robot with two arms, hence each procedure has an additional `arm-side` parameter.

- p_1: `move(abs-dest, arm-side)`
- p_2: `move(rel-dest, object, arm-side)`
- p_3: `move(abs-dir, arm-side)`
- p_4: `move(rel-dir, object, arm-side)`

The destination and direction parameters of these procedures are chosen from a finite set of relative positions that are relevant for the particular type of motion. A relative position is a set of 6-dimensional end-effector poses in the given coordinate frame. For instance, we can model the meaning of the preposition *above* with a relative position that corresponds to end-effector poses sampled around a pose that is higher than the landmark object on the vertical axis and at the center of the object on the table plane. The rotation of the end effector is chosen to make the gripper opening face the landmark object. The `object` parameter is chosen from the set of available objects, which differs based on the situation.

Given sufficient granularity all movements could be decomposed into a long sequence absolute directional movements. Such movements are straightforward to implement on a robot with simple cartesian control of the end-effector. However, this would be an inefficient way to cover the space of movements that can be demonstrated. Instead, we expect relative destination movements to cover most spatial translations and directional movements to act as stitching between these larger movements.

B. Additional commands

In addition to commands that directly control the robot's end-effector motions, we make several additional procedures available for users to command. This includes `pickup` and `place` procedures that are the most common object interactions and have robust implementations on most manipulators for certain objects. We have two commands, `point` and `lookat`, that allow testing and resolving object references without interacting with them. The next three commands give direct control on particular joints of the robot: `rotate` moves the robot's wrist joint in clockwise or counter-clockwise direction; `open` and `close` move the gripper to either extreme. The final command, `record`, instructs the robot to look at the table and update its representation of the objects in the world.

- p_5: `pickup(object, arm-side)`
- p_6: `place(abs-dest, arm side)`
- p_7: `place(rel-dest, object, arm-side)`
- p_8: `point(object, arm-side)`
- p_9: `lookat(object)`
- p_{10}: `rotate(rot-dir, arm-side)`
- p_{11}: `open(arm-side)`
- p_{12}: `close(arm-side)`
- p_{13}: `record()`

Finally, we have four additional commands that are provided for the user to navigate the system itself:

- p_{14}: `create-new-action()`
- p_{15}: `switch-to-action(number)`
- p_{16}: `execute()`
- p_{17}: `stop()`

C. A Model for Situated Language Understanding

The core computational problem in our approach is the inference of the intended procedure and its parametrization, given the input command and the current world state. In formulating and solving this problem, we exploit the fact there is a finite number of possible procedure parametrizations in a given state. We refer to a fully instantiated procedure as a *command*. We represent the distribution over these possible procedure parametrizations as a discrete random variable C. Similarly we represent the world state with the random variable S as a distribution over discrete states that captures the currently perceived objects in the environment and the history of commands (last referred object, last moved arm). Finally, we define L a random variable representing the users natural language utterance.

Our goal is to compute the joint distribution $P(C, S, L)$ that models the relationships between these quantities. We factor the distribution with the chain rule, producing:

$$P(C, S, L) = P(L|C, S)P(C|S)P(S)$$

This exact decomposition allows us to define the full model in convienent parts. For example, it is relatively easy to specify a basic model for how the state S influences the probability distribution over commands C, independent of the language. This happens in many ways, for example some of the commands are less likely because they involve relative movements that are not reachable due to the robot's kinematic constraints, while some commands are more likely given the recent command history.

Similarly, we can directly model $P(L|C, S)$ to define what the user would say, L, given the world S and the actual command C. We assume a template-based generative language model, where different templates associated with each command and state allow for compact modeling of situated phenomena, such as how the objects are references and the fact that users will often reword what they say to avoid sounding too repetitive.

D. Inference problem

To interact with users, we will ultimately need to find the best command $c* \in C$, given an utterance L in a specific world state S. More specifically, we need to find:

$$c^* = \arg\max_C P(C|S, L) \quad (1)$$

$$= \arg\max_C \frac{P(S)P(C|S)P(L|C, S)}{P(S, L)} \quad (2)$$

$$= \arg\max_C P(C|S)P(L|C, S) \quad (3)$$

The next two subsections give more details on how these distributions are specified. Section III-H provides the details of this computation, which is efficient in our domain and can be done at interactive speed.

E. Situated command model

We implement the situated command mode $P(C|S)$ as a scoring function that starts with equal scores on all possible commands and changes the scores of certain commands based on predefined rules. First, all commands instantiated with an object-preposition pair that is unreachable to the robot is penalized. Commands with absolute directional prepositions (*e.g.,* moving the arm up) will also receive a lower score if they push the robot outside its reachable space. Similarly, the pickup command instantiated with objects that are outside the robot's pickup range are penalized.

Other rules for scoring commands take into account the history of commands issued so far. The most recently referred object is considered most likely to be referred to next, so the scores of commands that refer to other objects are negatively penalized. Similarly, the last-used side of the robot (right or left arm) is preferred over the other. Finally, the state of the robot's end-effectors are factored into additional rules. For example, a `close` command receives a far lower score if the gripper that it refers to is already closed.

We chose the scoring parameters experimentally based on our desired emergent behavior of the system. While the exact values are available in our released code, their magnitudes are only important relative to each other. For example, the combined score penalties incurred when referring to a novel object with an unused arm are less than referring to an unreachable object, which are all less than asking the robot for an impossible movement (such as opening an open hand).

After all rules are applied, the final scores are normalized to provide a probability distribution.

F. Language model

Each procedure introduced in Sec. III-A and Sec. III-B has an associated template with a verb and zero or more parameter clauses. All parameters in the specified procedures have a finite number of options. Our model includes several alternative phrases for each option. Hence, given a fully instantiated command, language generation involves choosing a phrase for each parameter option that occurred in that instance. For most options this corresponds to randomly choosing a phrase from a list. However for objects, the phrase depends on the current context. These phrases are generated with the referring expression generation method described next.

G. Referring expression generation

Our referring expression have a fixed set of object properties. When the robot scans its environment, it perceives objects and computes a descriptor for each object. This descriptor involves continuous features such color, size and location.

Next, we rank all objects in the environment in terms of each feature of the descriptor. Each object that gets ranked at the top or bottom with a value surpassing the next object by a certain threshold is assigned a distinct property label. These labels and the associated continuous features are:

1) "rightmost" and "leftmost" (x-position)

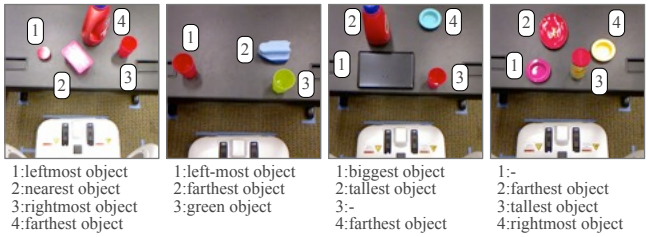

Fig. 2. Referring expressions generated by our language model for different objects in different scenes.

2) "nearest" and "farthest" (y-position)
3) "tallest" and "shortest" (z-dimension)
4) "biggest" and "smallest" (volume)

Color features are converted to discrete values and each object receives a color label with its value.

After all salient object labels are computed, each object ends up with a discrete vector of these word options that captures its distinguishing properties, which we can use to produce our final set of plausible referring expressions. We prioritize the distinct property labels as (1) location (left/right-most, middle, and farthest-nearest), (2) then size (biggest/smallest), (3) height (tallest/shortest), and (4) color. We then pick the least possible number of adjectives required to uniquely specify the particular object used in the command instance.

H. Inference

To use the generative language model for inference we first enumerate all possible sentences that can be produced by our language model. Since each command refers to at most one object there is a relatively small number of possible fully-specified commands which makes this enumeration easy to compute. The user's natural language utterance is matched against all generated sentences using a simple weighted bag-of-words approach. Each sentence that has a word match with the input utterance increases the score of the corresponding command instance. Relative to nouns, verbs are weighted more heavily when matching commands, whereas adjectives are weighted more heavily when grounding objects with referring expressions. The exact weights we used were experimentally determined and are available in our code. Once the scores for all commands are computed, they are normalized to obtain a probability distribution. The final step is to combine the language model with the situated command model as per Eqn. 3. This returns the most likely command.

I. Implementation

1) Hardware: We implement our natural language based PbD framework on a PR2 robot. PR2 is a mobile manipulator with two 7 degree-of-freedom (DoF) arms, each with a 1-DoF under-actuated gripper. For perception of objects we use the Kinect sensor mounted on the PR2's pan-tilt head. For speech commands, we use a Shure wireless microphone with a headset.

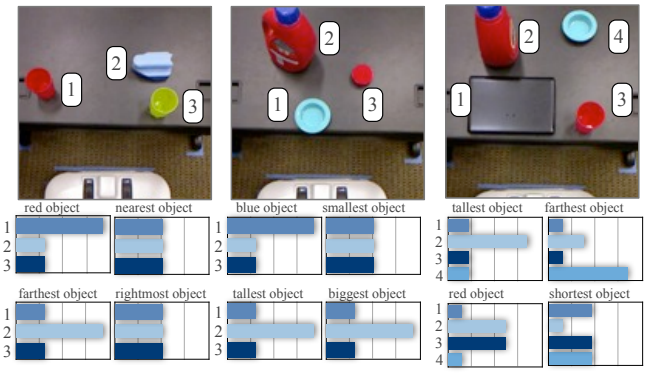

Fig. 3. Referring expression understanding in three different scenes. The bars show the probability distribution over the detected objects in the scene for the given object reference phrase.

2) Software: Our software is written within the ROS (Robot Operating System) framework and is open-source.[1] It is built as an extension to the open-source PR2 Programming by Demonstration package [7]. We use PocketSphinx [13] for speech recognition. Objects are segmented and localized using the PR2 tabletop object detection stack. We compute object descriptor attributes (dimensions and color) for each detected segment. The different commands also implemented using existing ROS packages such as tabletop object manipulation (`pickup` procedure) and MoveIt! (`move` procedures).

IV. EVALUATION

Next we describe how we evaluated our approach and present our findings from this evaluation. Although our system supports speech as input, our evaluations were done using text-based input. This allowed for easier repeatability and did eliminate speech-recognition errors. All evaluations were done by expert users of the system.

A. Referring expression generation and understanding

We start our evaluation by demonstrating the operation of smaller components of the complete system. First we investigate the generation and understanding of referring expressions. We created several scenes with different configurations of 2 to 4 objects, including ones later used for programming manipulation actions (Sec. IV-C). In each scene we used our language model to generate an expression to refer to each object in the scene. Then we tested our referring expression understanding technique by providing several expressions that target different objects in the scene.

Fig. 2 shows sample test scenes and the generated expressions for each scene. These examples demonstrate that our model chooses reasonable expressions and highlights its preference towards location. When location is ambiguous, the model is able to use other attributes such as color, size, and height. When an expression generated by the model

[1]The robot-specific half of the code is available at `https://github.com/mbforbes/pr2_pbd/tree/hf`, and the language-model half of the code is available at `https://github.com/mbforbes/hfpbd-parser`.

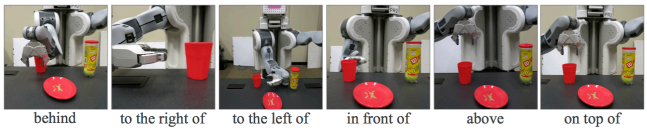

Fig. 4. Executions of the object relative destination `move` command (p_2) with different different prepositions (`rel-pos`), in response to the command "more right arm <`rel-pos`> to the right-most object"

(*i.e.,* the candidate expression with the highest weight) is used as an input reference expression in the same scene, our system is always correctly identifies the target object. In most cases these objects are identifiable by more expressions. Some examples are shown in Fig. 3, which presents sample test scenes with the probability distribution over the different objects in the scene for different input referring expressions. Expressions that result in even distributions among candidate objects, are indeed ambiguous for humans as well.

B. Object relative command understanding and execution

Next we verify understanding and execution of object relative commands. For one of the of the scenes from Sec. IV-A, we test all possible prepositions of the `move` procedure that have an `object` parameter, instantiating it with one target object in the scene. The robot's final pose after executing the move procedure with different prepositions is shown in Fig. 4.

C. Action programming

Next we demonstrate the programming of five different tasks using our system:

- Task 1: *Stack two cups*
- Task 2: *Place two objects side-by-side*
- Task 3: *Place an object in a box*
- Task 4: *Pick up a large box with two arms*
- Task 5: *Push plate to the edge of table and pick up*

The tasks involve various manipulation actions, that are variants of the benchmark in [2]. The first three involve picking up an object and placing it relative to another object. The fourth is a two-armed non-prehensile pick-up. The final one is a non-prehensile manipulation (push) followed by a pick-up. Fig. 5 shows one complete programming sequence for the task of pushing the plate to the edge of the table and picking it up. Fully specified commands are used in this programming sequence. The sequence demonstrates the successful execution of various types of procedures.

D. Comparison with Kinesthetic Programming

We program two actions described in Sec. IV-C–Task 1 and Task 3–and time how long it takes to program each in three conditions. The first condition is the presented system, programmed using fully-specified language. That is, for a command `move`(**above, object-0, right-hand**) we use language "Move right hand above the leftmost object." The second condition is also the presented system, but programmed using minimal language: we use the shortest possible sentences to achieve the desired command. For the same command, we might use "above blue," if **move**, and **right-hand** could be inferred from context, and the object could also be uniquely referred to by that property. The third condition is the kinesthetic PbD system, where we program the tasks by physically moving the robots arms to carry them out.

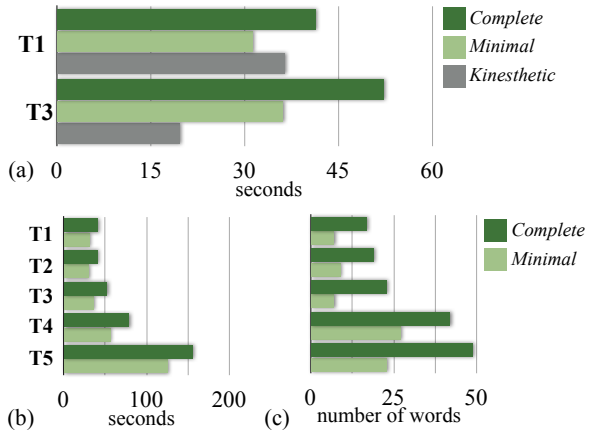

Fig. 6. (a) Comparison of time taken to program two tasks using fully-specified and minimal language and kinesthetic PbD. Comparison of (b) time taken and (c) words used to program five tasks using fully-specified and minimal language.

The results are shown in Fig. 6(a). We can see that using minimal language is considerably faster than fully-specified language, and for Task 1, faster than kinesthetic PbD. For Task 3, kinesthetic PbD is faster than both conditions of the proposed system. The extra time taken in Task 3 for the proposed system is due to difficulty in specifying the object to pick up in a crowded environment. In kinesthetic PbD, this trouble is lessened as one must merely move the robot's arm to the object without uniquely describing it.

E. Dealing with Incomplete Language

1) Quantitative results: One of the strengths of the presented system is its ability to handle incomplete language. It does this by applying the score functions described in Sec. III-E to incorporate context into the decision of which command to select. When using the system, this manifests as shorter programming times with more succinct utterances.

To test this, we program the actions from Sec. IV-C using both fully and minimally-specified language. We measure both the time taken to program the action as well as the number of words used during programming. The results are shown in Fig. 6(b-c). The time saved is roughly proportional to the total time spent. In Tasks 1 and 2, about 10 seconds are saved, whereas in Tasks 4 and 5, about 20 seconds are saved. The results are even more significant for the number of words used. In all tasks except Task 4, less than half of the words are needed in the minimal-language case compared to fully-specified language. Task 4 is exceptional in that it involves mostly low-level movements of moving each hand up in turn. Even still, the words used in Task 4 drops from 42 to 27, a savings of about 45%.

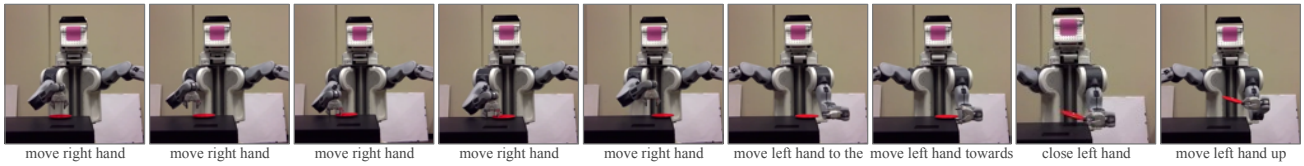

Fig. 5. Programming sequence for Task 5: used verbal commands and snapshots taken after the chosen procedure has been executed.

2) Qualitative results: Here are some key ways context and our language model help disambiguate language in the tasks shown to enable shorter utterances:

1) Context: The last-used arm is used if not specified
2) Context: The last-used object is used if not specified
3) Context: The gripper state (open, holding, or closed) can influence between `move`, `pickup`, and `place` commands.
4) Language model: If all of the parameters for a command are specified or inferred without specifying an object, and there is only one object, that command will be selected.
5) Language model: Certain parameters are only available in a single command, so specifying an option for that parameter automatically determines the command.

As an example, the utterance "up" can fully-specify `move`(**up, right-arm**) because of points 1 and 5. An even richer example is that the utterance "above" can fully-specify `move`(**above, object-0, left-arm**) using points 1,3 and 5, and either 2 or 4 depending on how many objects are in the scene.

F. Additional features

1) Interactivity: It can sometimes be difficult to be sure that what one intends the robot to do is what the robot will do. Getting this right is especially important in this framework, where the only interface to the robot is natural language. Due to this, we designed the set of procedures to support interactive workflows that allow the user to verify the robot's intent before executing, without switching "modes" into and out of programming.

Two such procedures are `point` and `lookat`. A possible sequence of events might go as follows:

1) **User: "Point to the biggest object."**
2) *Robot: "Please clarify object."*
3) **User: "Point to the tallest object."**
4) *Robot points to an object.*
5) **User: "Pick it up."**
6) *Robot picks up the object.*

The previous scenario is also supported by context in two ways. First, the user never specifies which arm the robot should use. We assume that the user used one of the arms previously (in 1-4), and optionally that the robot could only reach the object with one arm (5-6). Second, the user refers to the object as "it" in 5. This is automatically resolved to the last-referred object, which happened in 3.

2) Dialog: An additional feature of the system presented is the care taken to give helpful feedback to the user when problems arise. This is manifested in four ways.

First, as touched on briefly in Sec. III-H, the robot provides a specific clarification request when this is possible. This happens after user input the system determines all procedures that it deemed most likely (but couldn't choose one best from) come from the same command template in the language model. In that case, the robot asks the user to clarify one of the parameters that is in contention between the top choices.

Second, the system gives specific error messages when it selects a single command from user input but fails to execute it. For example, if the command is `move`(**above, object-0, right-hand**), and the robot cannot reach that location, it will respond with a statement similar to "Cannot move right had above the right most object." This detailed response is especially help when in the middle of executing a long action; a message like "Error." does not give the user a clue as to what specifically went wrong.

Third, object-referring phrases are generated dynamically and always as a part of feedback given to the user. Whether it is part of an error message like in the previous point, or during the robot's normal narration of its execution, the system will generate a referring expression for each object that is involved in each command that is executed. This means that if the objects change between executions, the robot will still accurately describe what it is attempting to do.

Finally, the robot tracks the phrases that were used to program it. If a user uses the phrase "grab" instead of "pick up" to issue a command, then for error messages, motion feedback, and future executions, the robot will use "grab" as well. This allows a user to use supported vocabulary that is most natural to him or her, and the robot follows his or her lead transparently, without enforcing language constraints.

V. LIMITATIONS

A. Detail of attributes

Our implementation extracts a relatively small number of properties from objects. We do not attempt any object classification, even shape recognition. We choose from only three colors (red, green, or blue) for an object's color category. Especially in scenarios where there are multiple types of objects, an objects type, such as whether it is a cup or a box, is perhaps its salient attribute. Furthermore, there is a vast body of work in computer vision to support such efforts. Integrating more attributes of objects would greatly improve the implementation of our framework.

B. Richness of referring expressions

We model only the properties of objects that have "absolute saliency." That is, if some but not all objects could be described with a property, then no object receives that property. Though this model simplifies referring expressions in that most attributes are manifested as distinct properties, it also precludes us from using other rich expressions. For example, allowing non-distinct properties would enable compound adjective expressions. It could be that an object is not the "leftmost" object, nor is it the "farthest" object, but it is the "farthest leftmost" object. Also, allowing non-distinct properties could allow other sources of inference to fully specify an object. If the user commands a pickup of a red object and only one red object is within reach, it might be reasonably to pick that one up. Currently, the robot will instead ask the user to clarify to which object they were referring.

C. Collision avoidance in movement

Frequently, the robot needs to move its arms around in a tabletop environment that is populated with objects. Under the current implementation, the robot does not take into account the existence of objects when planning movement, except for in the `pickup` procedure. This either leads to collisions with objects, or with the user explicitly adding semantically unnecessary steps (such as "move arm up") in order to avoid objects. Software is available for motion planning with basic collision avoidance, so integrating this feature as future work would be a cost of time.

D. Modeling the held object

As an extension of the previous limitation, the robot should also model an object that it is holding. It must do this to avoid colliding the object with other objects or the table while moving, as collisions may happen even if the robot itself is not touching any other surfaces. In addition, modeling the help object is vital for accurate placement. Placing a small object at a certain location requires a different end-effector location than placing a large object. Similar to the previous limitation, software is available that implements this feature, so adding it to the system would be a question of time.

E. Sophistication of language model

The current language model is brittle from a linguistic perspective, as it relies on exactly matching a hand-selected set of vocabulary. Other work exists that incorporates syntactic [17] or even semantic parsing [4] in creating instructions. Though our language model is intentionally simple to avoid introducing errors, making use of a more sophisticated language model would increase the flexibility of the system by allowing less constrained input.

VI. CONCLUSION

We contribute a method for programming object manipulation tasks using natural language commands. The key insights exploited by our method are that (i) task-oriented motion can be efficiently decomposed into a sequence of motions procedures with different types and (ii) such motion can often be described with spatial language, using object reference expression and positional and directional prepositions. Our method uses its perception of the current context, as well as its historical context (*e.g.,* last used command), in conjunction with the input natural language command, to infer the intended procedure and its parameters. The context helps resolve referring expressions and ambiguities in the input language. We present a full implementation of our method on a PR2 robot, demonstrate its capabilities in different tasks and scenarios, and compare it to a kinesthetic teaching alternative.

REFERENCES

[1] B. Akgun, M. Cakmak, K. Jiang, and A.L. Thomaz. Keyframe-based learning from demonstration. *(In press) Journal of Social Robotics, Special issue on Learning from Demonstration*, 2012.

[2] S. Alexamdrova, M. Cakmak, K. Hsaio, and L. Takayama. Robot programming by demonstration with interactive action visualizations. In *Robotics: Science and Systems (RSS)*, 2014.

[3] B. Argall, S. Chernova, M.M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[4] Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *TACL*, 1:49–62, 2013.

[5] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. *Robot Programming by Demonstration*, chapter 59. Springer, December 2008.

[6] Timothy Brick and Matthias Scheutz. Incremental natural language processing for hri. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 263–270. ACM, 2007.

[7] PR2 Programming by Demonstration in ROS. `http://ros.org/wiki/pr2_pbd`.

[8] M. Cakmak and L. Takayama. Teaching people how to teach robots: The effect of instructional materials and dialog design. In *Proceedings of the International Conference on Human-Robot Interaction (HRI)*, 2014.

[9] Nicholas FitzGerald, Yoav Artzi, and Luke S Zettlemoyer. Learning distributions over logical forms for referring expression generation. In *EMNLP*, pages 1914–1925, 2013.

[10] Yunyi Jia, Ning Xi, Joyce Y. Chai, Yu Cheng, Rui Fang, and Lanbo She. Perceptive feedback for natural language control of robotic operations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[11] Jens Kober and Jan R Peters. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856, 2009.

[12] Petar Kormushev, Dragomir N Nenchev, Sylvain Calinon, and Darwin G Caldwell. Upper-body kinesthetic teaching of a free-standing humanoid robot. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3970–3975. IEEE, 2011.

[13] P. Lamere, P. Kwok, W. Walker, E. Gouvea, R. Singh, B. Raj, and P. Wolf. Design of the cmu sphinx-4 decoder. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, 2003.

[14] Cynthia Matuszek, Nicholas FitzGerald, Luke Zettlemoyer, Liefeng Bo, and Dieter Fox. A joint model of language and perception for grounded attribute learning. *arXiv preprint arXiv:1206.6423*, 2012.

[15] DK Misra, J Sung, K Lee, and A Saxena. Tell me dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *Robotics: Science and Systems, RSS*, 2014.

[16] Shiwali Mohn and John Laird. Learning goal-oriented hierarchical tasks from situated interactive instruction. In *Proceedings of the The Twenty-eighth National Conference on Artificial Intelligence (AAAI)*, 2014.

[17] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth J Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.