An Interactive Approach for Situated Task Teaching through Verbal Instructions

Çetin Meriçli, Steven D. Klee, Jack Paparian, and Manuela Veloso
{*cetin,veloso*}@*cmu.edu*, {*sdklee,jpaparia*}@*andrew.cmu.edu*
Computer Science Department
Carnegie Mellon University

Abstract

The ability to specify a task without having to write special software is an important and prominent feature for a mobile service robot deployed in a crowded office environment, working around and interacting with people. In this paper, we contribute an interactive approach for enabling the users to teach tasks to a mobile service robot through verbal commands. The input is given as typed or spoken instructions, which are then mapped to the available sensing and actuation primitives on the robot. The main contributions of this work are the addition of conditionals on sensory information that the specified actions to be executed in a closed-loop manner, and a correction mode that allows an existing task to be modified or corrected at a later time by providing a replacement action during the test execution. We describe all the components of the system along with the implementation details and illustrative examples in depth. We also discuss the extensibility of the presented system, and point out potential future extensions.

Introduction

Interacting with humans and responding to their instructions in the environment through natural language are very important features for service robots. Though the robot is equipped with some task knowledge, it would be very useful—if not necessary—to also have the ability to teach new tasks to the robot easily, and preferably without having to modify the robot control software. Also, being able to teach new tasks through natural language instructions is a very desired capability as verbal communication is the primary method of information exchange among humans.

There are, however, various challenges in instructing robots using natural language such as robust speech recognition in case of spoken interaction, dealing with the ambiguity in given instructions stemming from the flexibility of natural language, and properly mapping given commands to robot behavior primitives that can be executed by the robot.

We contribute an approach for enabling users to teach the steps needed to perform a task to a mobile service robot in terms of available sensing and actuation primitives. Our approach consists of a natural language input module (either through speech or by typing), a parser that processes the raw language instructions and converts them to a graph-based representation, and an execution module that takes the generated behavior representation and executes the task. Our language supports conditionals and loop structures based on sensing different landmarks in the environment. Main contributions of this paper are:

- A task specification language using the available sensing and actuation capabilities of a service robot that also supports loop structures and conditionals based on sensing.
- A keyword based filtering approach to the natural language input that makes the system robust to different expressions of the same command to some extent as long as the proper keywords are used in correct order.
- A correction feature that allows the teacher to modify parts of an existing task as desired.

In the remainder of the paper, we first give a brief overview of related work in the literature and how our approach differs. We then describe all the components of the contributed approach thoroughly. After presenting implementation details on a real robot as well as a set of illustrative examples of task teaching and correction, we conclude the paper with a discussion of the approach as well as commenting on possible future work.

Related Work

Natural language based interaction with robots has been examined in various different scenarios ranging from commanding robotic forklifts to teaching robots how to give a tour. Recently, approaches that leverage probabilistic models trained on a labelled corpus have been proposed to deal with the uncertainty in the unrestricted natural language instructions. In (Kollar et al. 2010), a system using Spatial Description Clauses (SDC) is proposed for parsing a spoken natural language command and extracting the spatial information. The extracted spatial commands are then grounded to the available actions that can be executed by the robot. The proposed system is evaluated in a navigation scenario. (Tellex et al. 2011) presents Generalized Grounding Graphs (G^3) for parsing commands given through speech using natural language. A Conditional Random Field (CRF) is trained on a manually labelled corpus and then used to infer the most likely G^3 representation for a given natural language command. They applied their method to robotic navigation and manipulation domains. Similarly, a system that uses a parser that learns through statistical machine translation methods is presented in (Matuszek, Fox, and Koscher 2010) for enabling a robot to follow navigation instructions. In (Chuangsuwanich et al. 2010), a speech-based robot operation system is proposed for handling cargo in an outdoor scenario by a robotic forklift. Contrary to the probabilistic approaches discussed above, this approach is not robust against flexible natural language commands. Instead, the proposed system expects commands to be given according to the specified syntax. Another approach for verbal instruction is presented in (MacMahon, Stankiewicz, and Kuipers 2006). Similar to the works discussed above, they also use a parser to convert the given natural language command into an action representation. If an unknown word is encountered in the command, the system first tries to find a similar word with a known concept using WordNet. If no such words can be found, that part of the command is ignored. Ignoring an unknown and unexpected part of the command bears resemblance to our approach for dealing with ambiguity, but our approach deliberately checks for the known commands instead of trying to parse the entire instruction. An approach for updating plans generated by a planner using natural language instructions given through speech is introduced in (Cantrell et al. 2012). A multi-modal spatial language system that utilizes gestures along with the verbal instruction is presented in (Skubic et al. 2004). The parser of the system relies on well-defined grammar, and the extracted lexical items are then mapped on the robot primitives. The main difference between our approach and these aforementioned works is that the other approaches do not allow robots to learn from the provided instructions, as the instructions are merely used to operate the robot without saving them for future use.

Teaching tasks to a robot through verbal instruction has also been studied in various domains. Rybski *et al.* introduced a method for teaching tasks composed of available action primitives for a service robot using spoken verbal instructions (Rybski et al. 2008). Their approach does not perform any disambiguation on the received verbal command converted to text via a speech recognition module; therefore, their system mandates a strict syntax for the commands. As a part of the teaching process, the robot can also learn navigation trajectories by following the demonstrator. The main difference between our approach and this work is that our algorithm allows repetitions (cycles) in the task representation and enables the user to modify and correct an existing task. In (Dzifcak et al. 2009), a system that translates given natural language instructions into formal logic goal description and action languages is presented. The parsing of instructions is done through the use of predefined associations between the lexical items in the instructions and the corresponding λ-expressions.

A system for instructing the robot how to navigate is presented in (Lauria et al. 2001). The proposed system maps the received instruction to the defined action primitives. An interesting aspect of this work is that these action primitives are extracted from a corpus collected from several subjects. In (Nicolescu and Mataric 2003), a method for learning representations of high level tasks is proposed. Their approach allows learning a task composed of non-repetitive sequences of predefined robot primitives; in addition, it supports revisions of the taught task and generalizations of task representations over multiple demonstrations of the same task. Although mapping the instruction to action primitives in these two studies resembles our approach, our language differs as our primitives are parameterized and our instruction language also contains conditionals. Teaching soccer skills via spoken language is addressed in (Weitzenfeld, Ejnioui, and Dominey 2010). In their approach, there is a predetermined set of actions and natural language commands that maps on those actions. An interesting aspect in their proposed framework is that it allows the teacher to query the robot for accessing its internal state. Their vocabulary includes a set of actions like shoot and pass, and if-then-else control expressions that can be coupled with queries about the state features. Among the main differences with this work and our approach is the ability of our system to execute the task step by step to enable verification of the taught task as well as modifying and correcting the taught actions.

Approach

The task teaching takes place in three consecutive operations in our system:

- Processing verbal instruction
- Generating task representation
- Execution and correction

In this section, we review all components of our system in detail.

Instruction Graphs for Task Representation

We represent tasks as a composition of available robot primitives using a special graph structure called an Instruction Graph (IG). An IG consists of a tuple $G = \langle V, E \rangle$ formed from n instructions, where $V = \{v_i \mid \forall\ i \in [0, n]\}$, and v_i corresponds to the i^{th} command given to the robot ranging from 1 to n. The starting node of G is denoted with v_0. E is a set of tuples $\langle v_i, v_j \rangle$ representing a directed edge from v_i to v_j. An edge between two vertices denotes a possible transition from one command to the other. A vertex is a 3-tuple of the form v=$\langle ID, ActionType, Action \rangle$. Each vertex is given an identification number that also specifies their relative order in execution so that $\forall\ v_i \in V$, *ID*= i.

The *ActionType* field describes the type of each vertex whereas the *Action* field tells the interpreter which actions and sensing are necessary. There are four defined action types that a vertex can have:

- **Do:** A vertex is designated with the *Do* action type if it performs an action completely in open loop.
- **DoUntil:** As opposed to the *Do* action type, *DoUntil* refers to an action that has a sensory component.
- **Conditional:** This action type refers to a vertex with more than one outgoing edge, representing a fork in the flow of execution. The $Action$ element stores the specific condition evaluated to determine which branch of execution to follow at runtime.

Action Type	Keyword(s)
Do	No Keyword
DoUntil	"until"
Conditional	"if","while","do while","end if", "end while"

Table 1: Action types and corresponding keywords.

Action Type	Action	Keyword
Do	move forward	"forward"
Do	turn the robot	"turn"
Do	speak to the user	"say"
DoUntil	move forward in closed-loop	"forward"
DoUntil	turn in closed-loop	"turn"

Table 2: Defined actuation commands and corresponding keywords.

- **GoTo:** This action type is used internally to implement loop structures. The $Action$ field contains the ID of the vertex that the interpreter will jump to. $GoTo$ vertices are never created directly by the user.

Our language also has three special commands without specified action types as they are not used in the tasks:

- **Save:** This command saves the current task in memory to a file under a specified name.
- **Load:** This command loads a previously saved task into the memory and appends it to the current node in the execution flow. The details of how the load command works are discussed in the next subsection.
- **Shutdown:** This command terminates the task execution.

Generating Instruction Graphs from User Input

When processing the given instruction to create the instruction graph, we search for specific keywords in the user input to determine what type of command has been requested. Once we infer the action type and the action, we make certain assumptions about the input form such as the existence and order of the expected keywords. Also, we filter out any unknown words in the input, therefore, a command can still be successfully parsed even if it is expressed differently as long as the keywords are correctly used and are in order. After extracting the action type, action name, and corresponding parameters (if any), a new vertex and new edge are added to the current task instruction graph G. Teaching a task is an interactive process as the robot asks for confirmation for each inferred action. After each confirmation, a relevant node is created and added to the current IG. Table 1 and Table 2 shows the supported action types and actuation commands, respectively.

As opposed to the *Do* and *DoUntil* action types which are single-step commands, *While* and *DoWhile* commands create loops over a group of nodes that can also contain nested loops. Therefore, the generation of such conditional nodes differ from the actuation nodes. For a conditional node, a *Conditional* vertex with two children is created. For an *If* node, the execution continues with the first child vertex if the specified condition evaluates to true, and the execution continues from the second otherwise.

For the loops, an additional *GoTo* node is created. The difference between the *While* and *DoWhile* constructs is whether to evaluate the loop condition at the beginning or end of the loop. In the case of a *While* loop, the *Conditional* node is inserted at the beginning. A *GoTo* node is placed at the end of the loop and points back to the *Conditional* node. The body of the loop consists of everything added to the graph between these two nodes. In case of a *DoWhile* loop, the *Conditional* node is placed at the end of the body. When the condition evaluates to true, the *Conditional* node transitions to a *GoTo* node, which jumps to the first node in the body of the loop. This loop implementation guarantees at least one full execution of the loop body.

Algorithm 1 and Algorithm 2 show the algorithms for the creation of actuation and conditional nodes, respectively. The flow of execution through actuation nodes is straightforward; however, closing conditional statements requires knowledge of their starting location. To accomplish this, we utilize a stack to store a record of conditionals in the order they were entered. When the body of a conditional is closed, an element is popped off the stack. This element references the starting location of the conditional to be closed.

Algorithm 1 Creating a Node for Actuation Commands.

function addActuation($v_{current}$,id,$instruction$)
if checkKeyword($instruction$,"forward") **then**
    $action \leftarrow$ "$Move$"
else if checkKeyword($instruction$,"turn") **then**
    $action \leftarrow$ "$Turn$"
else if checkKeyword(instruction,"say") **then**
    $action \leftarrow$ "Say"
end if
if checkKeyword($instruction$,"until") **then**
    $actionType \leftarrow$ "$DoUntil$"
else
    $actionType \leftarrow$ "Do"
end if
$params \leftarrow parseInformation(actionType, action)$
$id \leftarrow id + 1$
$v = createNode(id, actionType, action, params)$
$v_{current}.children[0] \leftarrow v$
$v_{current} \leftarrow v$

Saving and Loading Tasks

Once the teaching is completed, the resulting instruction graph in the memory can be saved to a file for future use by using the *Save* command along with a file name to save the task to.

A previously saved task can be loaded into memory with the *Load* command. The *Load* command works as follows. Given that the user is creating an instruction graph $G = (V_g, E_g)$ with n inputs, we denote the most recently created node as v_{g_n}. The *Load* command loads another instruction graph $H = (V_h, E_h)$ from the specified file. With the newly loaded task at hand, a new instruction graph is formed as $(V_h \cup V_g, E_h \cup E_g \cup \{(v, v_{h_0})\})$. In other words, the resulting graph is the union of G and H with one additional edge connected the most recently created vertex of G to the source-vertex of H. Leveraging this capability, a library of subtasks can be created and used to compose new tasks.

Algorithm 2 Creating Branches and Cycles in the Flow of Execution

function beginConditional($stack$,$v_{current}$,id,$condition$)
$actionType \leftarrow checkActionType(condition)$
$v \leftarrow createNode(id, actionType, condition)$
$stack.push(v)$
$v_{current}.child[0] = v$
$v_{current} = v$
$id = id + 1$

function endConditional($stack$,$v_{current}$,id)
$v_{conditional} \leftarrow stack.pop()$
$v_{conditional}.child[1] \leftarrow v_{current}$
if $v_{conditional}.actionType ==$ "$While$" **then**
    $v \leftarrow createNode(id, \text{"}GoTo\text{"}, v_{conditional}.ID)$
    $v_{current}.child[0] \leftarrow v$
    $v_{current} \leftarrow v$
    $id \leftarrow id + 1$
end if

Executing Tasks

Execution of a task in the form of an instruction graph is a traversal operation that starts from the first node (v_0), and follows defined transitions. The *Do* and *DoUntil* nodes have only one directed edge outward. This edge is followed once the action is performed, and the execution continues with the next node. *Conditionals* have two directed outward edges. Their *Action* field is a conditional statement that evaluates to true or false at runtime. If the condition evaluates to true, the first child of the node is set as the next node to be executed. If the condition evaluates to false, the second child of the vertex is chosen. The algorithm for executing an instruction graph is given in Algorithm 3.

We have three main robot primitives that the given instructions are mapped to. The *Move* primitive is used to execute motion. The supported motion types of moving forward and turning are specified by the tuple $m = \langle \Delta_x, \Delta_y, \Delta_\Theta, v_t, v_r \rangle$, where Δ_x, Δ_y represent the forward and lateral displacement, respectively, and Δ_Θ represents the amount of rotation. v_t and v_r represent the maximum translational and rotational velocities respectively. All translational motion is specified in meters, and all rotational motion in radians. The *Say* primitive allows the robot to speak a given text message. Finally, the *Sensing* primitive makes use of the available sensory information on the robot.

Modifying and Correcting Tasks

A major feature of our approach is the ability to let the teacher correct parts of the task as desired. This can vary

Algorithm 3 Executing a task.

$G \leftarrow loadTask()$
$v_{current} = G.vertices[0]$
while $v_{current} \neq \emptyset$ **do**
    if $v_{current}.actionType ==$ "Do" **then**
        $executeAction(v_{current}.action)$
        $v_{current} \leftarrow v_{current}.children[0]$
    else if $v_{current}.actionType ==$ "$DoUntil$" **then**
        while $v_{current}.senseCondition$ **is not** $true$ **do**
            $executeAction(v_{current}.action)$
            $v_{current} \leftarrow v_{current}.children[0]$
        end while
    else if $v_{current}.actionType ==$ "$GoTo$" **then**
        $v_{current} \leftarrow v_{current}.children[0]$
    else if $v_{current}.actionType ==$ "$Conditional$" **then**
        if $evaluateConditional(v_{current}.action)$ **then**
            $v_{current} \leftarrow v_{current}.children[0]$
        else
            $v_{current} \leftarrow v_{current}.children[1]$
        end if
    end if
end while

from editing the parameters of a node to replacing the node with a new one. We envision three major reasons why a user may want to correct a portion of a learned task:

- Changing open loop parameters to make instructions more accurate
- Switching from open loop to closed loop, or vice-versa
- Modifying a few instructions of an existing task to populate new tasks (code reuse)

The first example is likely to occur when a parameter value is not as accurate as predicted. This often happens due to miscalculations, unexpected changes, or faulty calibration of the robot. A successful framework must be flexible enough to adapt to inconsistencies and uncertainty in its environment. It is also necessary for a framework to easily integrate new sensory information or deal with a particular sensor becoming unavailable. To that end, we support switching from open loop commands to their closed loop equivalents, and vice-versa. For example, if necessary sensory data are no longer available, it is possible to change the code where the data were used from closed loop to open loop without rewriting the entire task. Lastly, when writing a new function that is similar to an existing one, it should be possible to reuse the bulk of the code.

Implementation and Illustrative Examples

In this section, we first describe the actual implementation on our mobile service robot in detail. Then, we present examples of task teaching and correction. When presenting examples, we first describe the tasks, followed by the initial teaching conversation between the teacher and the robot. Next, we illustrate the generated instruction graph after processing the instructions. Finally, we present an example correction scenario and discuss the changes reflected upon the instruction graph for the task as a result of the correction.

We implemented and tested our approach on our CoBot mobile service robot (Rosenthal, Biswas, and Veloso 2010). The CoBot service robot has an omnidirectional mobile base, and is equipped with various sensors including Microsoft Kinect sensors. The users interact with the robot using its touch screen interface and microphones (Figure 1).

Figure 1: CoBot service robot.

In the current implementation of our approach, we use Augmented Reality (AR) tags as the sensory input. AR tags are visual signs that can be detected and recognized uniquely through image processing. In addition to the identification number for a detected AR tag, the relative pose of the tag with respect to the camera is also computed. Leveraging this detailed detection ability, the execution of a task can be conditioned upon the existence, the relative distance, and the relative angle of a certain AR tag in our approach. The user can instruct the robot either through typing the instruction or through speech recognition. For the speech recognition, we use Google Speech API.

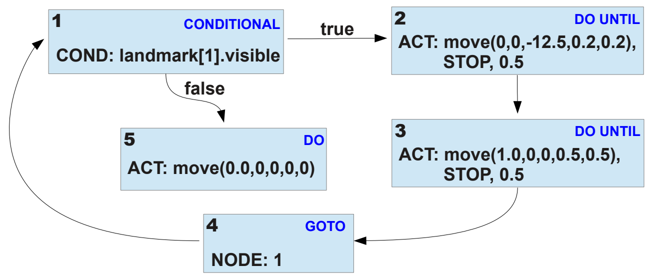
In the remainder of this section, we present examples to task teaching and correction.

Following a Visual Landmark

This task illustrates the use of loop structures to have a continuous behavior as long as the loop condition holds. The purpose of the task is to make the robot follow a visual sign. At each execution cycle, the robot first faces itself toward the specified visual landmark if the landmark is visible, and it performs a turning motion to search for the sign if the sign is not seen by the robot. If the landmark is in the field of view of the robot and the robot is currently facing towards it, then, the robot goes toward the sign while maintaining a certain distance from it. Finally the whole process is repeated as long as the visual landmark stays in the sight of the robot. Both the turning and going forward motions are conditioned upon the specified visual landmark, therefore they both translate to *DoUntil* nodes. The *While* loop translates to a *Conditional* node, and a *GoTo* node. Figure 2 shows the interaction for teaching the robot to follow a visual sign and the corresponding generated instruction graph.

```
User : While landmark 1 is visible
Robot: What should I do in this loop?
User : Turn until landmark 1 is ahead
Robot: I will turn until I am facing landmark 1.
       What should I do next?
User : Forward until 0.5 meters from landmark 1 max 0.2 meters
Robot: What should I do next?
User : End Loop
Robot: Loop Ended. What should I do next?
User : Stop
```

(a)



(b)

Figure 2: (a) The teaching conversation for the "follow the sign" task, and (b) the resulting instruction graph.

Getting Coffee

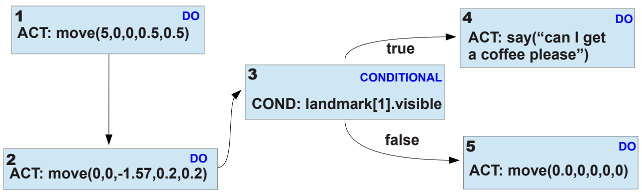
The second example we present teaches the robot how to go to a cafe from a starting point and order a cup of coffee. The task consists of motions performed in open-loop and demonstrates an example use of an *if* clause to determine the course of execution depending on the visibility of a visual landmark. In the task, the robot first approaches the cafe counter by going forward and then turning to left. Then, it checks for the presence of a visual landmark. If the specified landmark is visible, the robot infers that the cafe is open, and therefore it proceeds with the ordering. Otherwise, the robot infers that the cafe is closed, so it terminates the task execution. Figure 3 shows the interaction between the teacher and the robot during teaching and the resulting instruction graph. The motion commands in the beginning of the task are not conditioned on sensing, therefore they are translated into *Do* action type, and are executed in an open-loop manner. The check for the visual landmark is translated into an *If* clause, conditioned on the visibility of the specified visual landmark. Finally, speaking a request for coffee requires no sensing, so it is also translated into a *Do* node.

Task Correction

If the user observes that the parameters for the open-loop actions are imprecise, he or she can make a revision without re-creating the task from scratch using the debugging and correction mode. In the correction mode, the robot speaks the action description for each action, and asks whether the user confirms that action or wants to make a correction. If the user confirms the action, the robot leaves that action intact and proceeds to the next action. However, if the user decides to correct the action, the robot then asks for a replacement action. Once the user specifies the replacement action with its parameters, the original node is updated as to reflect the

```
User : Forward 5 meters
Robot: What should I do next?
User : Turn left
Robot: What should I do next?
User : If Landmark 1 is visible
Robot: What should I do if Landmark 1 is seen?
User : Say I would like to order a cup of coffee
Robot: What should I do next?
User : End If
Robot: What should I do in the other case?
User : Shutdown
```

(a)



(b)

Figure 3: (a) The teaching conversation for the "get coffee" task, and (b) the resulting instruction graph.

correction. Figure 4 shows an example interaction for correcting an action on the "get coffee" task presented above.

```
Robot: Go forward for 5 meters.
Robot: Should I confirm this action or revert it?
User : Revert.
Robot: Please give me a new command to replace this with.
User : Forward 6 meters
Robot: Action revised. The new action is go forward for 6 meters.
```

Figure 4: An example interaction during step-by-step execution to correct an action.

Conclusion and Future Work

In this paper, we presented a system for enabling users to compose tasks for a mobile service robot using spoken or written verbal instructions. Among the main contributions of our approach, we can name the closed-loop actuation primitives conditioned on perceiving visual signs, the ability to represent repetitive tasks using loop structures, and the correction mode that allows step by step execution and modification of the desired portion of the task. By actively seeking for the known keywords instead of trying to parse the entire instruction, our system is partially immune to the flexibility of natural language.

The system is designed and implemented in a modular manner. The task teaching language is agnostic to the underlying robot primitives. Therefore, extending the language with additional sensing and actuation elements is straightforward.

We are planning to extend the current correction mode to the task teaching phase, hence, enabling the trainer to immediately see the outcome of an action and modify or correct it as desired. We are also planning to expand the correction notion to the situation-bounded corrections that can be stored with the state of the system at the time of correction, and then retrieved and re-used when a similar situation is encountered. Finally, we are planning to increase the number of sensing and actuation elements to enable conditional actions based on the sensory information such as the human presence around the robot, proprioceptive sensing, attention, and gestures.

References

Cantrell, R.; Talamadupula, K.; Schermerhorn, P.; Benton, J.; Kambhampati, S.; and Scheutz, M. 2012. Tell me when and why to do it!: run-time planner model updates via natural language instruction. In *Proc. of HRI*.

Chuangsuwanich, E.; Cyphers, S.; Glass, J.; and Teller, S. 2010. Spoken command of large mobile robots in outdoor environments. In *Proc. of Spoken Language Technology Workshop (SLT)*.

Dzifcak, J.; Scheutz, M.; Baral, C.; and Schermerhorn, P. 2009. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *Proc of ICRA*.

Kollar, T.; Tellex, S.; Roy, D.; and Roy, N. 2010. Toward understanding natural language directions. In *Proc. of HRI*.

Lauria, S.; Bugmann, G.; Kyriacou, T.; Bos, J.; and Klein, E. 2001. Personal robot training via natural-language instructions. *IEEE Intelligent Systems* 16:38–45.

MacMahon, M.; Stankiewicz, B.; and Kuipers, B. 2006. Walk the talk: Connecting language, knowledge, and action in route instructions. In *Proc. of AAAI*.

Matuszek, C.; Fox, D.; and Koscher, K. 2010. Following directions using statistical machine translation. In *Proc of HRI*.

Nicolescu, M. N., and Mataric, M. J. 2003. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proc. of AAMAS*.

Rosenthal, S.; Biswas, J.; and Veloso, M. 2010. An Effective Personal Mobile Robot Agent Through Symbiotic Human-Robot Interaction. In *Proc. of AAMAS*.

Rybski, P.; Stolarz, J.; Yoon, K.; and Veloso, M. 2008. Using dialog and human observations to dictate tasks to a learning robot assistant. *Journal of Intelligent Service Robots, Special Issue on Multidisciplinary Collaboration for Socially Assistive Robotics* 1(2):159–167.

Skubic, M.; Perzanowski, D.; Blisard, S.; Schultz, A.; Adams, W.; Bugajska, M.; and Brock, D. 2004. Spatial language for human-robot dialogs. *Trans. Sys. Man Cyber Part C* 34(2):154–167.

Tellex, S.; Kollar, T.; Dickerson, S.; Walter, M.; Banerjee, A.; Teller, S.; and Roy, N. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proc. of AAAI*.

Weitzenfeld, A.; Ejnioui, A.; and Dominey, P. 2010. Human robot interaction: Coaching to play soccer via spoken-language. In *IEEE/RAS Humanoids'10 Workshop on Humanoid Robots Learning from Human Interaction*.