Teaching Task Flow Through Dialog and Observation

Kevin Yoon Paul E. Rybski

CMU-RI-TR-07-18

Submitted in partial fulfillment of the requirements for the degree of Master of Science

May 2007

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

© Carnegie Mellon University

Abstract

In order for robots to act as valuable assistants for non-expert users, they need to be able to learn new abilities and do so through natural methods of communication. Furthermore, it is often desirable that tasks be learned quickly without having to provide multiple demonstrations. Training should also be conducted so that the user has a clear understanding of the manner in which environmental features affect the behavior of the learned activity, so that execution behavior is predictable.

We present an interactive framework for teaching a robot the flow of an activity composed of elements from a set of primitive behaviors and previously trained activities. Conditional branching and looping, order-independent activity execution, and contingency (or interrupt) actions can all be captured by our activity structures. Additional convenience functionality to aid in the training process is also provided.

By providing a natural method of communicating production rules analogous to rigid programming structures, tasks can be trained quickly and easily. We demonstrate our task training procedure on our CMAssist mobile robot.

Contents

1 Introduction **1**

2 Related Work **2**

3 System Overview **3**

4 Activity Structures **4**
- 4.1 Behaviors 4
- 4.2 Tasks 4
- 4.3 Todolists 5
- 4.4 Combining Tasks and Todolists 6

5 Training **7**
- 5.1 Training Tasks 7
- 5.2 Training Todolists 10

6 System Implementation **11**

7 Experimental Results **12**
- 7.1 Patrol the lab 13
- 7.2 Give the lab tour 13

8 Summary and Future Work **18**

A CMAssist Task Training Guide **19**
- A.1 CMAssist File Descriptions 19
- A.2 Starting top-level behavior 19
  - A.2.1 Hardware setup 19
  - A.2.2 Software setup 19
- A.3 Task Training 22
- A.4 Todolist Training 22
- A.5 Notes for future development 22

1 Introduction

One of the most natural applications for robots is to employ them as assistants, and as assistants they have uses in a wide variety of domains. They could provide both assistant and nursing services that would enable the elderly to live more independently. They could also act more as partners in mixed human-robot teams whether it be in the military, construction, or search-and-rescue. However, if such robots are to gain widespread and long term acceptance, they will need to be capable of not only learning new tasks, but learning them from non-expert users.

We previously introduced a method for task training via dialog and demonstration in [12]. Therein we described a collaborative natural language procedure for constructing tasks from a set of primitive behaviors and/or previously-trained tasks, which in turn could be used to build other tasks. This modular task architecture supports an expanding repertoire of abilities. Different training modes enable different features, such as the ability to attach locational context to a given command through teacher observation, reducing the explanatory responsibilities of the human trainer. Preconditions on task actions serve as a failure-handling mechanism that appropriately directs task flow should an action fail. The robot also engages the human in a verification dialog to resolve ambiguities in task flow and, in so doing, brings about mutual understanding of the task representation.

This understanding can be desirable, sometimes essential, in situations where the time or opportunity to provide multiple demonstrations and/or make corrections through practice trials is unavailable, and the chance of the robot exhibiting unexpected behavior due to conditions unencountered during training is unacceptable. The training dialog described herein enables the human to quickly construct rigidly-formulated tasks where the features that affect task flow must be explicitly conveyed and not inferred. Additionally, because tasks are symbolically referenced with natural language labels, they are transferable across heterogeneous robots that share the same or a similar primitive behavior set.

In this paper, we present some enhancements and modifications to this task training technique that include the ability to capture conditional looping so that repetitive, or cyclic, tasks can be created. Interrupt events that should be checked for the duration of a task can also be specified to trigger contingency actions. Additionally, a new construct called a todolist has been added, which permits order-independent activity execution. Moreover, tasks can now be trained “on the fly” — that is, while training another task that uses it — to support a top-down design approach while still permitting the bottom-up construction of tasks. Furthermore, locational context is no longer inferred automatically as this is not always desirable in some situations. However, location-specific actions can still be specified explicitly with a simple grounding utterance.

This paper is organized as follows. Section 2 describes related work. A high-level system overview in terms of activity execution is provided in Section 3. A description of activities, which include what we refer to as behaviors, tasks, and todolists, is provided in Section 4 followed by a description of how tasks and todolists are trained in Section 5. Section 6 contains a brief overview of our CMAssist[1] robot and experiments

[1] http://www.cs.cmu.edu/∼coral/cmassist/

are shown in Section 7. A summary and plans for future work are in Section 8.

2 Related Work

Robot task learning and programming-by-demonstration (PBD) has been explored by several groups. In [1], [3], and [13], robots learn actuator trajectories or control policies from user task demonstrations. In [2] and [14], a task is built by determining which primitive actions, from a base set of capabilities, should be combined to conduct the task demonstrated. Our work has the ability to discern, to a limited extent, which primitive actions should be combined to execute a given task by way of inferring locational context on actions. We note, however, that this is not the main focus of this work, nor is the intent to derive low-level control strategies. It is instead a method for constructing tasks by sequencing higher-level primitive actions, as well as previously learned tasks, through natural language keyphrases that map to production rules for task flow structures.

Our work is largely inspired by [9] and [8]. In [9], a mobile robot is joysticked through multiple demonstrations of a task from which it generates a generalized task representation in the form of a directed acyclic graph (DAG). The task is then pruned down to a linear sequence through teacher feedback in the form of verbal cues over multiple practice trials. In [8], a stationary humanoid robot that understands some speech maintains multiple hypotheses of the intended representation of the task which is refined through a structured dialog with the user. Neither of these robots can speak and instead communicate with the user through only gestures and facial expressions. Our approach employs a similar turn-taking framework for instruction and task refinement, but we endow the robot with the capability of speech which conveys more directly the robot's understanding of the task and guides the human more effectively in resolving ambiguities. In this way, we reduce the need to refine a learned task through practice.

Similar dialog-driven interaction mechanisms have been developed, though primarily in the area of plan recognition. A theoretical discourse structure introduced in [4] is applied in [11] and [7] where characteristics of the collaborative setting are exploited to reduce the amount of input required of the user for plan recognition. They introduce a framework to make user interfaces more intelligent, but which relies upon prior knowledge in the form of *recipes*, or action plans for achieving goals. This work differs from ours in that the goal is to help the user carry out tasks according to perceived intent whereas as we are striving to teach a robot entirely new tasks with no pre-conceived notions of intent. Our approach could potentially be used instead to build the recipes necessary for this plan recognition method to work.

An augmentation-based learning approach is described in [10]. The task structure, including conditional branching and looping, is inferred from user demonstration. Manual edits can also be made to fix incorrect task structures and constrain the induction procedure on subsequent demonstrations. Again, this approach is explored in the software application domain and there is no effort to conduct a collaborative discourse with the user for natural interaction. Additionally, in our work, branching and looping structures are explicitly and quickly communicated by the user, rather than being inferred over multiple demonstrations.

A multi-modal interface for programming tasks is described in [5] that additionally allows the user to control task priority during execution. Instruction-Based Learning [6] is similar to our work in that it uses a base set of behaviors that are associated with natural language symbolic labels and a modular architecture for symbolic tasks.

None of these works, however, describe the ability to convey branching or looping flow constructs within the task structure that are conditioned on explicitly-communicated features. Nor do they address the issue of structuring tasks for activities that need not necessarily be executed in the order in which they were communicated. This severely limits robustness and the types of tasks that can be trained. Through speech one can very compactly format instructions for execution based on detectable environmental states. No intention beliefs are maintained that may result in unexpected behavior during execution, but rather, by engaging the user in a true spoken dialog, we can quickly train tasks with clearly defined execution flow that is transparent to the user.

3 System Overview

Figure 1 depicts a simple overview of the system architecture we employ in our CMAssist robots. Within the top-level behavior is an *Activity Selector* that, upon parsing given speech commands, places the appropriate activities in the *Activity Repertoire* onto the *Current Activity List*.

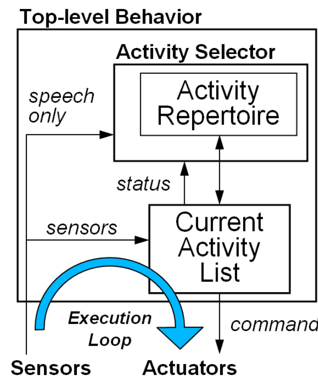


Figure 1: CMAssist software architecture

An *activity* is the encompassing term for behaviors, tasks, and todolists which are described in more detail in Section 4. The *Activity Repertoire* is the collective map that associates natural language symbolic labels to known activities. For example, “Go to” in the phrase “Go to the door” maps directly to the navigation behavior which would be put onto the *Current Activity List* with the location parameter “door”. An activity building behavior can also add new activities to the *Activity Repertoire* as will be shown in Section 5.1.

Though the various activity types have differing internal structures, they are all

executed by the same function form where the inputs are the $sensors$ and a $command$ object, and the outputs are an integer $status$ flag and a new $command$ object.

$$(status, command) = \text{Activity}(sensors, command)$$

The $sensors$ object gives an activity module access to sensory data (including uttered speech) while $command$ is an object that can be modified by an activity to store actuator commands, such as motor velocities or speech output. A single $command$ object is passed through each of the activities in the *Current Activity List* so that commands requested by activities of lower priority are visible to higher priority activities. Activities can take this information into account when actuator commands need to be overridden. For example, when the obstacle avoidance behavior needs to decide whether to veer left or right to circumvent an obstacle, it can check the $command$ object to see in which direction the navigation behavior was trying to drive the robot and choose to go in a similar direction. The main execution loop then involves processing all of the activities in the *Current Activity List* with the given sensory data. When the last activity on the *Current Activity List* is completed, $status$ is routed back to the *Activity Selector* which determines if behaviors need to be removed from the *Current Activity List*. The $command$ object is processed to drive the actuators.

The *Activity Selector* is triggered on speech input and is responsible for inserting commanded activities, removing conflicting ones, and removing completed or failed activities.

4 Activity Structures

4.1 Behaviors

A behavior maps low-level sensory data to actuator trajectories in order to accomplish some high-level goal(s). The robot is assumed to be preprogrammed with some basic set of behaviors. For a mobile platform, these primitive skills might include obstacle avoidance and high-level navigation capabilities.

4.2 Tasks

The basic building block of a task is the task item (Figure 2). A task item consists of three main components: a (potentially empty) precondition list, a reference to an activity and a list of its execution parameters, and a pointer list to subsequent task items.

The precondition list contains the conditions that must be satisfied before the activity can be executed. There are two types of preconditions: enabling and permanent. Enabling preconditions are evaluated only once before the task item's activity is executed. Permanent preconditions are monitored continuously for as long as the activity is being executed. Preconditions serve as a failure-handling mechanism that prevents an activity from being executed when conditions necessary for success are not met. For example, the task item containing a "Open the fridge" command would be reasonably preconditioned on *being in the kitchen*.

Depending on the completion status of the activity, the associated link is followed to the next task item to be executed. Typically, there is only a single link to the next task item, but in the case of conditional task item nodes (i.e. *if* and *while* statements) there are pointers associated with the true and false cases. Task execution terminates at leaf nodes in the task graph that contain no pointers to subsequent task items.

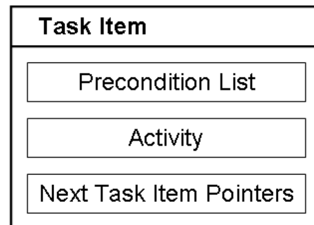


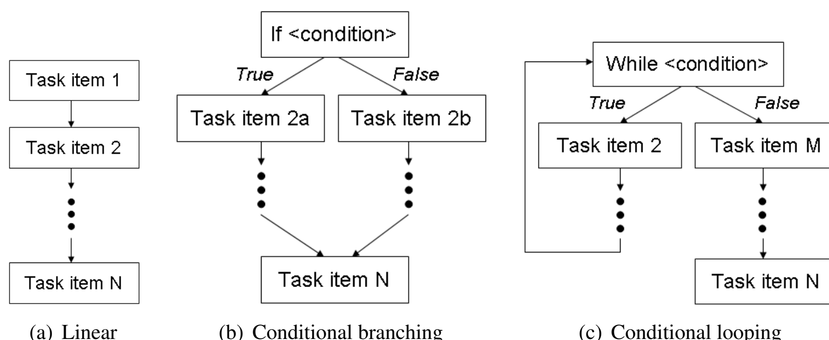
Figure 2: Task item

A task then is a temporally ordered sequence of task items captured in a directed graph structure. They can represent simple linear sequences such as in Figure 3(a). Here, the robot executes Task items 1 through N in order. Tasks can also represent conditional branching as shown in Figure 3(b). Depending on the evaluation of $<$*condition*$>$, either Task item 2a or Task item 2b will be evaluated followed by whichever tasks follow it until the branches reconnect at Task item N. Cyclic tasks can be represented by loops as shown in 3(c). For as long as $<$*condition*$>$ is true, Task item 2 and the subsequent task items inside the loop are executed. This is made possible by applying the while-condition as a permanent precondition on all task items inside the loop.

For some tasks it may be necessary to execute contingency activities, such as when some event occurs requiring special action and the current task be put on hold. Rather than inserting *if* and *while* statements throughout the task, the user can optionally specify contingency event-action pairs that are checked for the duration of the task execution. Unlike the previous conditional constructs, a contingency table (Figure 3(d)) is not represented within the directed graph itself but is an attribute of the task object. The contingency table is an associative structure that maps an interrupt event k to an action tuple (a, r), where a is the activity to execute when k is true and r is a boolean value determining whether or not the original task should be resumed when either k is no longer true or a has completed.

4.3 Todolists

Todolists (Figure 4) are a special type of activity that allows the user to specify a list of items that are to be executed in no particular order. These todolist items, as with task items, can refer to any activity: behaviors, tasks, and other todolists. There is nothing unique about the structure of a todolist. It is simply a list of disconnected activities that, unlike tasks, cannot capture conditional branching and looping. It is rather the manner



(a) Linear (b) Conditional branching (c) Conditional looping

Event	(Activity, Resume?)
⋮	⋮

(d) Contingency (Interrupt) events

Figure 3: Task flow structures

in which a todolist is executed that distinguishes it from the other activities enabling it to accomplish unordered tasks as people do on a daily basis.

Activity	Attempts	Complete
A	0	False
B	0	False
C	0	False

maxNumTries = 2

Figure 4: Example todolist

We currently employ a round-robin execution scheme where we loop through the list until each item has either successfully completed or failed $maxNumTries$ times, where $maxNumTries$ is specified during training.

Clearly, some optimal scheduling strategy to minimize failed attempts could be applied here when taking into account information like estimated todolist item durations and reasons for past failures. Item priority could be an additional constraint that such a strategy might take into account. This is beyond the scope of this work where we simply provide a construct in which order-independent execution of activities is made possible.

4.4 Combining Tasks and Todolists

As previously mentioned, an activity can refer to a behavior, a previously-trained task, or a todolist. Since, tasks and todolists are activity containers as well as being activities

themselves, one can be composed of the other. By combining these two structures, we are able to represent very expressive tasks in a hierarchical fashion. See Figure 5 for an example.

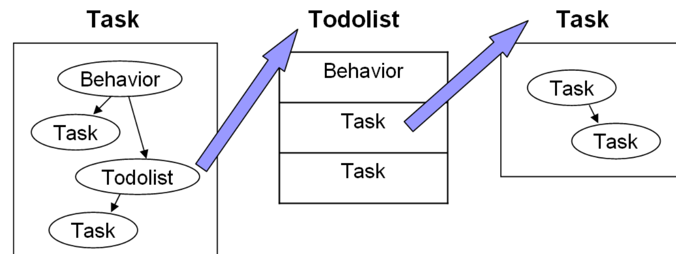


Figure 5: Example of a task that contains a todolist that is itself composed of more tasks

5 Training

The basic method behind the training approach we employ is to allow the user to convey production rules through the primary mode of speech. Each recognized user utterance is mapped to one of three things: (1) an activity in the activity repertoire that is to be appended to the current activity structure[2], (2) a flow control structure that affects where and how subsequent activities are appended to the current activity structure, or (3) a "special" command, such as a question that the user might ask during the training procedure.

Throughout the training procedure, the robot responds with an affirmative "ok" after every user utterance to indicate understanding. The robot will also ask the user questions about parameters that were not defined when the user has finished training, thus guiding the user through dialog towards a well-defined activity structure.

5.1 Training Tasks

Task training is itself a behavior that can be invoked in one of two modes: dialog-only and dialog-and-observation. The former is invoked with the keyphrase "When I say T" and the latter with "Let me show you what to do when I say T", where T is the name of the task to be trained and is typically an imperative statement. In dialog-only mode, all commands must be issued to the robot verbally. In dialog-and-observation mode, the robot invokes its person-following behavior such that it is always in the vicinity of the human trainer as he moves around the environment. In this manner, the robot can interpret deictic utterances like "come here". In the previous work [12], this mode was used to automatically attach locational context to each command given by the user. This was done by inserting a locational precondition into all task items corresponding

[2] A task or todolist. Behaviors are not trained.

to the location in which the command was given and by prepending each task item with navigational "goto" commands to ensure that the robot would go to location in which it was meant to execute a particular task item. In an effort to provide a framework for the training of more general tasks, where it is not necessarily appropriate to assume that actions should be executed where they were demonstrated, locational preconditions are no longer assumed but can be easily attached to subsequent commands with the "come here" phrase.

Task flow control is communicated by keyphrases summarized in Table 1. An example of a user utterance that creates a conditional branching structure (Figure 3(b)) is "If you see Kevin, say 'Hi Kevin'. Otherwise, say 'Where is Kevin?' before looking for Paul". The resulting task would cause the robot to say either "Hi Kevin" or "Where is Kevin" depending on whether Kevin was detected. It would then begin the activity called *looking for Paul*.

Table 1: Task flow commands

Command	**Description**
"If <*condition*>"	Appends a conditional node to the task graph. Subsequent commands are added to *True* branch.
"Otherwise"	Causes subsequent commands to be added to the *False* branch of the current *if* node.
"before"	Connects *True* and *False* branches of current *if* node with the following command. (Ends *if* block.)
"While <*condition*>"	Appends a conditional node to the task graph. Subsequent commands are added to *True* branch and preconditioned on <*condition*>.
"After that"	Routes execution flow to the current *while* node and appends subsequent commands to the *False* branch. (Ends *while* loop.)
"Meanwhile if <*condition*>"	Adds a contingency event to the task object and maps it to the next activity command. If <*condition*> becomes true at any point during task execution, the specified activity is exectued.
"Exit Task"	Appends a node that exits task with *success* flag.
"The task has failed"	Appends a node that exits task with *fail* flag.
"Come here"	Appends a *goto* node where the parameter is the location at which the user said "Come here". Subsequent tasks are preconditioned on being at this location until the user moves to another location in the robot's map.

Cyclic constructs (Figure 3(c)) can be specified with a phrase like "While Kevin is around, do a dance. After that charge batteries". Executing the resulting task would make the robot conduct the *do a dance* activity for as long as it sees Kevin. If Kevin leaves, the loop exits and the robot begins the *charge batteries* activity.

Contingency event-action pairs are specified with the "meanwhile" keyphrase. If we appended "Meanwhile if you see Paul, sing a song" to the previous example, the

robot would begin the *sing a song* task if Paul was detected at any time during the task (i.e. while dancing or charging batteries) and would continue to do so until the *sing a song* task completed or Paul became no longer visible. During training, the robot also asks the user if it should resume the original task after executing the contingency action.

Special utterances can be used to indicate that the task should exit. “Exit task” and “The task has failed” create task items that when reached during execution will terminate the task, the first with a success flag and the latter with a failure flag. (The task exits with a success flag by default even when “Exit task” is not said.) This is particularly useful when tasks are used in a todolist where the return status indicates whether a todolist item should be reattempted or not.

As can be seen, this approach to task training places more of the design burden on the user than some of the PBD techniques mentioned in Section 2, but it comes with the added benefit of increased mutual task understanding between the user and robot and consequently more predictable execution behavior. Also, tasks cannot be overfitted to training set conditions because task flow depends on features that are specified explicitly. Moreover, the natural interaction framework provides for quick and easy construction of these tasks.

Figure 6 shows a simple schematic for this Task-building behavior where we can see the speech input being processed by the *Speech Parser*. Therein, we first check if the utterance is a special helper command, such as those shown in Table 2. If it is not, then we check if it is a flow control command such as those shown in Table 1 and add nodes or update pointers to the task under construction as appropriate. If it is not that, then we check if it corresponds to an activity that already exists in the *Activity Repertoire*. If so, then we add a task item containing the activity to the task under construction. Finally, if the user has ended the task training sequence, the robot engages the human in a verification dialog to confirm the task description by reading it back to the human and to acquire any additional information that might be necessary, such as what to do when an *if* condition does not hold and the *otherwise* case was not specified, before saving the task to the *Activity Repertoire*. The $command$ object that is passed out of the *Speech Parser* contains speech output commands as well as the motor commands set by the Follow behavior.

The constructs in Figure 3 can be combined and nested to create activities that can richly capture task flow. It can also be seen that activities can become arbitrarily complex. While our task training approach is well-suited for composing complex tasks from simpler subtask, the robot can provide descriptive feedback and verify with the user the flow of the trained task to minimize errors during a long and potentially confusing training sequence. In Table 2 are some phrases that can be understood by the robot to aid the user during the training process.

Note that, in training mode, if the user says a phrase that is unrecognized, the robot will give the user the option of training a new task under the assumption that he may have been referring to a task that has not yet been created. In this way, the user can follow a top-down approach and train tasks “on the fly” as their need becomes apparent without having to plan all the required lower level subtasks ahead of time.

The user can say “Thank you” to simply end the training process and the learned task is saved to the *Activity Repertoire* as is. Or, by asking “Is that understood?”, the

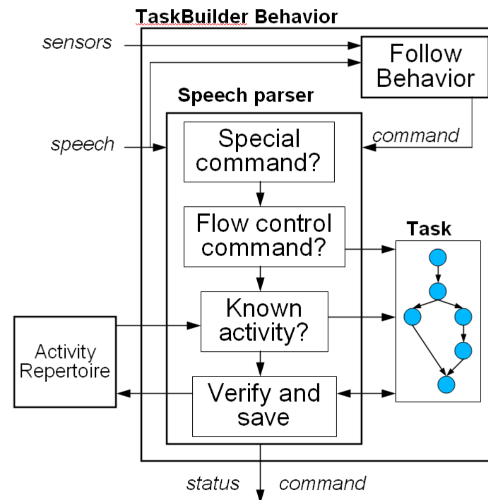


Figure 6: Task builder

Table 2: Training helper functions

Command	**Robot Function Description** (**T** = Training mode, **E** = Execution mode)
"Describe T"	**T,E**: Describes the task/todolist T
"What did you say?" "Can you repeat that?"	**T,E**: Repeats the last thing it said
"Where was I?"	**T**: Repeats the last two task items appended to the current task/todolist
Unrecognized/Misheard utterance	**T**: Asks if the user was referring to the name of a new task/todolist to train, and starts a new training process if this is so. **E**: Robot says that it did not understand and asks the user to repeat himself.

robot will dictate the task description and await confirmation from the user. If the task is correct, the robot then attempts to clarify ambiguities. Currently this involves asking the user for instructions for unspecified "otherwise" cases.

5.2 Training Todolists

The todolist training behavior is invoked with the keyphrase "Let's make a todolist called L", where L is the name of the todolist to be trained. The user then simply lists the activities that are to be added. When finished, the user says "Thank you" to end training or asks "Is that understood?" to have the robot repeat the todolist items

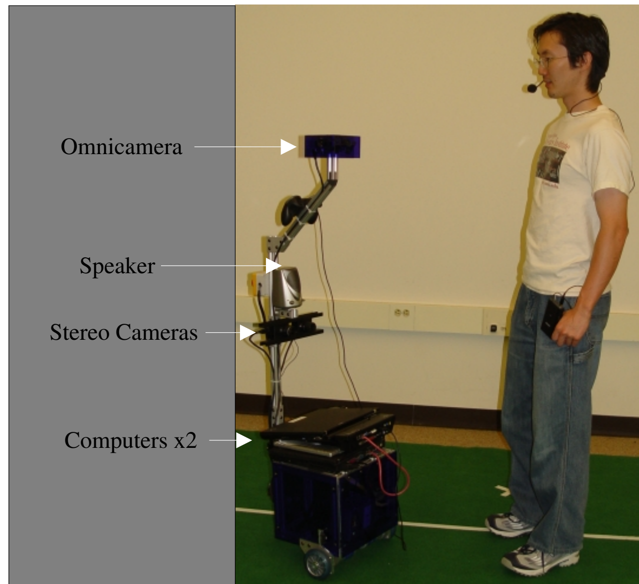


Figure 7: The CMAssist robot interacts with a user.

dictated. If the user confirms that the todolist is correct, the robot then asks the user for the number of times it should attempt to repeat failed tasks.

Unlike tasks, todolists are learned exclusively through dictation only, since todolist items themselves are typically high-level actions that can be trained as tasks.

6 System Implementation

The task training procedure was evaluated on our CMAssist robot, pictured in Figure 7, that was expressly developed as our research platform for human-robot interaction. An earlier version of this task training work was demonstrated at the Robocup@Home competition in June 2006 where our team placed 2nd out of 11 teams.

The robot has modular hardware and software architectures to enable rapid prototyping and integration of new sensory, actuator, and computational components. An omnidirectional camera and stereo camera allow it to sense the presence of people wearing color-coded shirts, while the stereo camera and laser range finder together are used for navigation and obstacle avoidance. The robot can also recognize a subset of natural English language speech and speak through a Text-To-Speech (TTS) engine. These capabilities equip the robot with sufficient spatial and environmental information to execute our interactive training algorithm.

A list of relevant behaviors used by our robot is as follows:

- **Goto(x,y)/Goto(name)** Drives the robot to a location specified either by global coordinates or a location label.

- **Say(s)/Ask(s,p)** Generates speech output from the TTS engine. **Say(s)** causes the robot to speak an utterance **s**. **Ask(s,p)** requires that the robot identify and speak the utterance to a particular person **p** if present and wait for a response.

- **Follow(p)** Causes the robot to follow person **p** while maintaining a fixed distance of approximately 1m.

- **StateChecker(a)** Unique in that the useful output is the $status$ flag rather than the $command$ object, which is not modified at all. Uses sensors to calculate a $status$ flag indicating whether or not an assertion **a** is true or false. Used by task items containing conditional statements such as *if* and *while* nodes.

- **TaskTrain(f)** Invokes the task training procedure. The **Follow** behavior is simultaneously executed if **f** is true causing the robot to follow the teacher and learn the task based on both the spoken utterances as well as the locations of the teacher.

- **TodolistTrain()** Invokes the todolist training procedure.

In order for the locations in the environment to be semantically meaningful as part of the training process, a map of the environment is provided to the robot which contains location labels. For instance, the locations of named objects such as couch, table, and television can be added to the map as well as general locations of rooms such as lab or living room. This a priori information is used to ground locations that are either mentioned in the human's speech or are visited as the human walks about the environment.

7 Experimental Results

The robot was trained to conduct a series of tasks that highlight the expanded capability of this task training framework. We focus on capabilities not already described in [12]. The first task is a security activity called *Patrol the lab*. This example illustrates both conditional branching and looping, makes use of an interrupt event, and demonstrates the training of tasks "on the fly".

The transcripts for the training procedure are shown below. First, a task called *Sound the alert* (Figure 8) is trained, which is then used as the contingency action triggered when someone is detected by the robot in the *Patrol the lab* task (Figure 9).

For brevity, the "ok" feedback from the robot after every user utterance is omitted. Quoted phrases are those uttered by the user while phrases in <> are those uttered by the robot. Unquoted phrases describe what is physically happening in the scene. The numbers on the left in the transcripts are simply timestamps that denote where actions were executed on the robot's path depicted in the corresponding scenario visualizations. Figure 10 shows the visualization for the training of *Patrol the lab* in a top-down representation of our lab.

7.1 Patrol the lab

The training process for the *Patrol the lab* task is initiated in dialog-and-observation mode. Kevin says "Drive around the lab" which was not understood by the robot, so it begins a new training process. Kevin then proceeds to teach it how to drive around the lab by leading it to different locations and saying "come here" which the robot automatically converts into a *Goto* X command where X is the symbolic label for the current location of the human as determined from a given map. Kevin then asks "Is that understood?" to begin a verification process after which he resumes training the original *Patrol the lab* task. Having forgotten his place in the task he asks "Where was I?" and the robot reports the last two tasks items that were added. Finally, Kevin adds a contingency action, *Sound the alert*, in the event that the robot sees someone.

"When I say sound the alert" "If you see Kevin" "say hi Kevin" "Otherwise" "say intruder alert" "Thank you" <Task training is complete. Thank you.>

Figure 8: Training transcript for the *Sound the alert* task

By saying "Patrol the lab", the task is invoked. Figure 11 shows the transcript of what happened and the scene is depicted in Figure 12. The robot first enters the while loop after determining that Kevin is not there — the robot cannot see people when an obstacle obstructs its view — and travels from the door, to the QRIOs, to the AIBOs, and back to the door again (Figure 12(a)). As the robot travels from the QRIOs to the AIBOs, Kevin walks in triggering the interrupt condition "meanwhile if you see someone". The robot stops and executes *Sound the alert* causing the robot to say <Hi Kevin> (Figure 12(b)). Kevin then leaves and the robot continues with the patrol task and travels to the AIBOs. As it travels towards the door, Paul steps into view triggering the interrupt event once more. Paul is considered to be an intruder (by virtue of not being Kevin) and the robot says <Intruder alert> (Figure 12(c)). Paul then leaves and the robot again continues with the patrol task (Figure 12(d)).

7.2 Give the lab tour

The second task is *Give the lab tour* where the robot acts as a tour guide by taking the visitors to each exhibit and then asking an expert to present it in more detail. The task involves greeting visitors at the door, executing its subtasks *Give the QRIO tour* and *Give the AIBO tour*, and then saying goodbye at the center of the lab. The task is similar to the one described in [12] but includes two main differences: (1) the phrase "The task has failed" is used in the subtasks and (2) the subtasks are contained in a todolist. This enables the robot to recognize when a task on the todolist has failed and to reattempt it at a later time. The training transcript of the task *Give the QRIO tour* is

0	“Let me show you what to do when I say patrol the lab”
	<Ok, let’s begin>
6	“While Kevin is not here”
	“Drive around the lab”
	<I don’t understand. Did you want to teach me *Drive around the lab*?>
	“Yes”
	<Ok, please begin teaching me *Drive around the lab*>
41	Kevin walks to the door
	“Come here”
63	Kevin walks to the QRIOs
	“Come here”
81	Kevin walks to the AIBOs
	“Come here”
	“Is that understood?”
	<Yes. You said go to the door. Go to the QRIOs. Go to the AIBOs. Is this correct?>
	“Yes”
	<Thank you. Let’s continue training *patrol the lab*>
	“Where was I?”
	<You previously said while Kevin is not here, drive around the lab. What shall I do next?>
	“Meanwhile if you see someone”
	“sound the alert”
	<Ok. When I’m finished with that shall I resume *patrol the lab*?>
	“Yes”

Figure 9: Training transcript for the *Patrol the lab* task

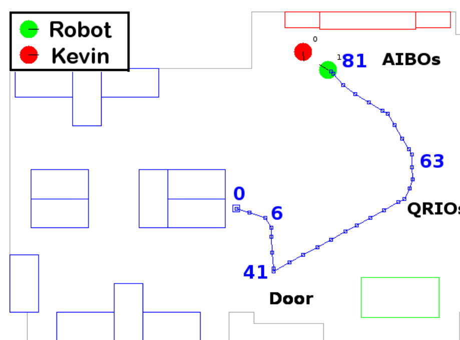
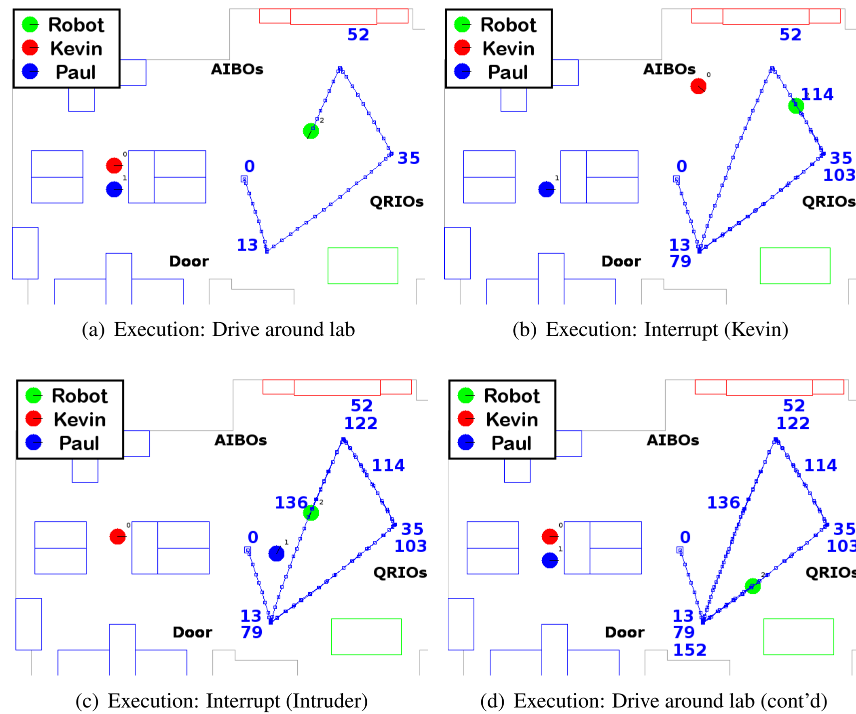


Figure 10: *Patrol the lab* task training. The colored blocks represent furniture and the labels **Door**, **QRIOs**, and **AIBOs** denote pre-specified map regions.

0	“Patrol the lab”
	Goto door
13	Goto QRIOs
35	Goto AIBOs
52	Goto door
79	Goto QRIOs
103	Goto AIBOs
114	Kevin walks in. Robot stops.
	<Hi kevin>
	Kevin leaves
122	Goto door
136	Paul walks in. Robot stops.
	<Intruder alert>
	Paul leaves
152	Goto QRIOs
	Continue *drive around the lab* task

Figure 11: Execution transcript for the *Patrol the lab* task



(a) Execution: Drive around lab

(b) Execution: Interrupt (Kevin)

(c) Execution: Interrupt (Intruder)

(d) Execution: Drive around lab (cont’d)

Figure 12: *Patrol the lab* task

shown in Figure 13. The task *Give the AIBO tour* is trained similarly except that it asks Paul, instead of Kevin, to talk about the AIBOs. The training transcript for the todolist *Talk about the robots* and the top-level task *Give the lab tour* are shown in Figures 14 and 15, respectively. The visualization of *Give the lab tour* is shown in Figure 16.

"Let me show you what to do when I say give the QRIO tour" <Ok, let's begin> Kevin goes to the QRIOs "Come here" "Say these are the QRIOs" "If Kevin is there" "ask Kevin to talk about the QRIOs" "Otherwise" "say unfortunately Kevin is not around to talk about the QRIOs" "The task has failed" "before" "say that concludes the QRIOs tour"

Figure 13: Training transcript for the *Give the QRIO tour* task

"Let's make a todolist called talk about the robots" <Ok, let's begin> "Give the QRIO tour" "Give the AIBO tour" "Is that understood?" <Yes, you said give the QRIO tour. Give the AIBO tour. Is this correct?> "Yes" <How many times shall I attempt each item before giving up?> "Two" <Ok. Todolist training is complete. Thank you.>

Figure 14: Training transcript for the *Talk about the robots* task

The execution of the *Give the lab tour* task is transcribed in Figure 17 and visualized in Figure 18. After greeting the visitors at the door, the robot begins the *Talk about the robots* todolist and goes first to the QRIOs where it finds that Kevin is not there and consequently the *Give the QRIO tour* task fails (Figure 18(a)). It then travels to the AIBOs where it sees Paul and requests that he speak about them (Figure 18(b)). When done, Paul tells the robot he is finished and the robot decides to retry the failed items in the todolist, i.e. *Give the QRIO tour*. This time Kevin is there and the task is completed successfully (Figure 18(c)). The tour is completed when the robot goes to the lab center and thanks the visitors for coming (Figure 18(d)).

0	"Let me show you what to do when i say give the lab tour"
	<Ok, let's begin>
16	Kevin goes to the door
	"Come here"
38	"Say welcome to the multi-robot lab"
	"Talk about the robots"
50	Kevin goes to the lab center
	"Come here"
	"Say thank you for visiting"

Figure 15: Training transcript for the *Give the lab tour* task

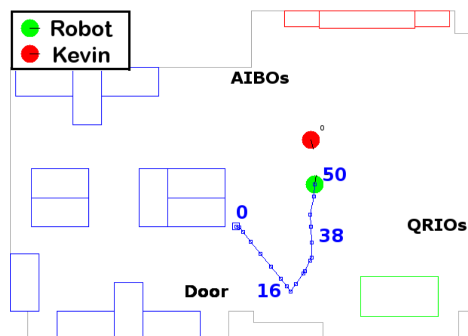
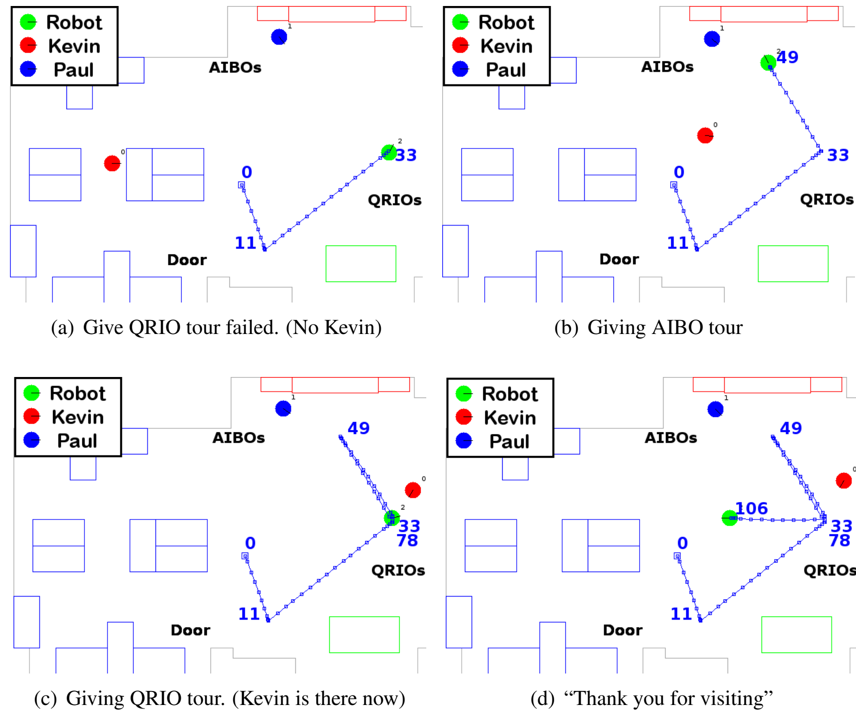


Figure 16: *Give the lab tour* task training

0	"Give the lab tour"
	Goto the door
	<Welcome to the multi-robot lab>
11	Goto the QRIOs
	<These are the QRIOs. Unfortunately, Kevin is not around to talk about them.>
33	Goto the AIBOs
49	<These are the AIBOs. Paul, could you please talk about the AIBOs?>
	Paul talks about the AIBOs and then tells the robot that he is finished.
	Goto the QRIOs
78	<These are the QRIOs. Kevin, could you please talk about the QRIOs?>
	Kevin talks about the QRIOs and then tells the robot that he is finished.
	Goto the lab center
106	<Thank you for visiting>

Figure 17: Execution transcript for the *Give the lab tour* task



(a) Give QRIO tour failed. (No Kevin)

(b) Giving AIBO tour

(c) Giving QRIO tour. (Kevin is there now)

(d) “Thank you for visiting”

Figure 18: *Give the lab tour* task execution

8 Summary and Future Work

We have presented an enhanced task training procedure that permits the user to easily communicate a rich set of task flow structures.. Through dialog and observation of the user as he moves around, this framework allows for natural methods of conveying rigid production rules to construct these flow structures when training a task.

The next steps in this work include field deployment and evaluation of the system under the use of non-technical users. In addition to identifying new types of flow structures and convenience functionality we also hope to identify alternative keyphrases that users might use to map to the different flow structure production rules and add these to our grammar of recognized phrases.

The inference of preconditions beyond the locational ones which we are currently capable of generating also poses an interesting challenge as we explore different kinds of tasks and domains.

It would also be advantageous to parameterize tasks so that relevant features can be specified at execution time. That is, instead of training the tasks *Give the AIBO tour* and *Give the QRIO tour* separately, a “template” task called *Give the* **R** *tour* could be trained, where **R** is the feature parameter that is propogated through the task definition and is specified when the task is executed to yield the two tour tasks. This could not only decrease training times, but require fewer resources as well due to code sharing.

A CMAssist Task Training Guide

The following outlines the procedure for operating the CMAssist robot in task training mode as of the date this document was written. Relevant notes for further development of the task training system are also included.

A.1 CMAssist File Descriptions

The table below briefly describes those files in the CMAssist software that are relevant to task training. File paths are specified with respect to the software root directory specified by the environment variable $ER1ROOT$.

A.2 Starting top-level behavior

The following procedure outlines how to start up the CMAssist robot with the top-level demo.py behavior. This is not specific to task-training, but is required in order to learn tasks and execute learned tasks. These instructions do not include the activation of all sensors and actuators, but only those that are relevant to task-training. Table 4 shows the designations for the computers used by CMAssist.

A.2.1 Hardware setup

1. Connect motor controller to serial port on PC-LINUX-BOTTOM.
2. Connect stereo camera firewire cables and power cord to PC-LINUX-BOTTOM.
3. Connect CAMEO firewire cable and power cord to PC-LINUX-TOP.
4. Connect speaker to PC-LINUX-TOP.
5. Connect Shure wireless microphone or VoiceTracker Array Microphone to PC-WINDOWS.

A.2.2 Software setup

Typically, the following is done remotely where the operator connects to these machines from a single terminal on PC-TELEOP. Exceptions are those modules that output high-bandwidth data like the stereo server. These applications should not be executed through a remote shell.

On PC-WINDOWS:

1. Start Nautilus voice server. (This is not a part of the er1/ codebase.)

On PC-LINUX-BOTTOM:

1. Start motor server ($robot/er1server.py$)

Table 3: Task Training Files

File	Description
behaviors/ demo.py	Main (top-level) behavior.
*behaviors/ *.wp*	Files specifying labels of points in the environment map as well as pre-defined collision-free connections between them. Read in by demo.py.
behaviors/ waypoint2.py	Navigation behavior. Performs simple Djikstra path-planning between known waypoints (in .wp file). Also contains function defining regions, allowing the robot to ground actions at particular locations. Currently, a region is a circle specified by some radius about a known waypoint.
behaviors/ task_train.py	Task-training behavior. Can be initiated with follow behavior on or off.
behaviors/ tasklist.py	Task and task builder classes. (The term 'list' refers to the first version of the task structure which was a simple list. This is no longer an accurate descriptive.)
behaviors/ todolist.py	Todolist and Todolist builder classes.
behaviors/ activity_repertoire.py	Consolidating container class for behaviors, tasks, and todolists.
behaviors/ cmd_parser.py	Recognizes all main phrases that can be understood by the CMAssist robots and returns an appropriate command list object typically structured like [command, param1, param2,...]. In demo.py, the 'command' string is checked against the *Activity Repertoire*. If such an activity exists, it is executed with the corresponding parameters.
behaviors/ state_checker.py	Uses the featureset to process True/False queries.
user_scripts/ color_channel_map.csv	File that maps CAMEO color channels to person names. Used by the state_checker.
user_scripts/ robotIPs.py	Contains IPs and ports of all featureset components. Make sure these are correct otherwise behaviors may not be able to access sensor data or control actuators!
user_scripts/ saved_activities.txt	Contains saved activities. This is loaded by demo.py on startup into the *Activity Repertoire* and overwritten on shutdown with the contents of the *Activity Repertoire*.

Table 4: Computer Designation

Computer Designation (Hostname)	Description
PC-LINUX-BOTTOM (On Carmela: melonhead On Erwin: orca)	Linux laptop onboard the robot.
PC-LINUX-TOP (On Carmela: bowhead On Erwin: trilobite)	Linux laptop onboard the robot. Executes more processor-intensive modules.
PC-WINDOWS (narwhal)	Windows laptop, soon to be onboard the robot. Executes speech recognition only.
PC-TELEOP (e.g. pufferfish)	Linux/Windows machine from which teleoperator connects to the above machines to execute modules.

2. Start stereo camera server ($stereo/bin/stereo_server$)

On PC-LINUX-TOP:

1. Start CAMEO server ($vision/yuv_multicolor_server$)
2. Start Cepstral speech ($cepsral/cepstral_server.py$). (Cepstral TTS software must be installed already.)
3. Check that server IPs and ports are correct in $user_scripts/robotIPs.py$.
4. Start top-level behavior ($behaviors/demo.py$). Be sure to execute it with the correct number of color channels enabled. (e.g. For two colors, python2.4 $./demo.py$ -c 2)
5. Connect to featureset with text client. Default port is 50020. (e.g. python2.4 $./tcp_async_client_demo2.py$ -n <PC-LINUX-TOP hostname> -p 50020). If there is a $handle_expt()$ error, it means that the port is not open or the specified hostname is not recognized and may need to be added to $featureset/sensors/er1/text_er1.py$.

All commands can now be entered through the text console or verbally through the microphone. There are some commands that can only be recognized through the text console because they were not yet added to the speech grammar or there was never a need[3]. In order to have a command recognized through speech, it must be added to the Nautilus grammar file, located in $MMI/recosock/grammar/cmassist_utterances.fsg$ (not in er1/ codebase). The corresponding .bnf ASCII file shows what phrases are currently recognized. Currently, new utterances can only be compiled by CMAssist's parnter at Naval Research Labs.

[3]Expanding the grammar tends to lead to less accurate speech recognition, so only those commands that are essential should be added. Commands useful only for development are generally not added to the grammar.

A.3 Task Training

The following summarizes how to train the robot for a task. Training can be initiated in two modes:

Dialog-only Robot does not move and listens only to dictated commands. To initiate training in this mode, say “When I say <task name>”.

Dialog-and-Observation Robot starts up its following behavior so that commands like “Come here” can be ground to different map locations. To initiate training in this mode, say “Let me show you what to do when I say <task name>”.

Training can also be ended in one of two ways:

Force save Robot saves the learned task structure as is and exits training mode. To end training in this way, say “Thank you”.

With verification Robot begins a verification dialog wherein it checks for unspecified *otherwise* statements. To end training in this way, ask “Is that understood?”

The training of the task itself involves using a combination of task flow (Table 1) and convenience (Table 2) commands outlined in this paper. People are currently recognized as color blobs on CAMEO color channels. To add a new person, add a new color channel to the look-up table (.lut) file passed in to *vision/yuv_multicolor_server*. Whichever channel number is associated with that color should be mapped to the person's name in *user_scripts/color_channel_map.csv*. New regions can be specified in a waypoint (.wp) file, like *behaviors/testgraph.wp*. For a list of all commands that are understandable by the robot, see *behaviors/cmd_parser.py*.

A.4 Todolist Training

The training of todolist is very similar to that of tasks. The only thing that is different is that no task flow commands can be used — todolists ae simple list structures — and there is no verification dialog. Training can be initiated by saying “Let's make a todolist called <Todolist name>”.

A.5 Notes for future development

- Not all of the commands recognized by *behaviors/cmd_parser.py* are recognized during training. This is because when training mode is active, only commands relevant to training are recognized. It may be desireable in some cases to pass those commands recognized by *cmd_parser.py*, but not relevant to training, back to top-level demo.py behavior instead of just discarding them. See how the featureset text/voice sensor *peek*() function is used to achieve something similar in *snackbot_teleop.py*.

- Regions are currently defined as circles defined by a 1.5m radius about waypoints specified in *behaviors/testgraph.wp*. The ability to specify more general region shapes should be integrated. This may require new region functions to be

defined in $behaviors/waypoint2.py$ and/or a GUI through which regions can be specified.

- Problems may arise if any of the following occur.

  1. Activity A contains Activity B and B contains A.
  2. Activity A contains Activity C and Activity B contains Activity C, and Activities A and B are executed simultaneously.

  Though it is conceivable that case 1 may be a desirable task structure, recursion checking should be implemented to alert the user to such cases. To allow for both cases 1 and 2, however, the creation of multiple instances of activities may need to be implemented. Currently, only a single instance of a given behavior, task, or todolist is ever stored in, and executed from, the *Activity Repertoire* even though a single activity can be used by multiple other activities. When activities that use the same sub-activity are executed simultaneously, erroneous execution can follow unless a separate instance is created for the shared sub-activity. The current strategy to avoid case 2 is to never allow non-preprogrammed activities to be executed simultaneously. This is a restrictive limitation which should be corrected.

- Tasks and Todolists should be made in such a way that a todolist can be trained on-the-fly while training a task and vice versa. Currently, only tasks can be trained on-the-fly while training tasks and todolists while training todolists.

- Conduct user study with non-programmers to gauge ease of using this training system. Identify common vocabularly usage, expectations, areas of frequent user confusion, etc.

References

[1] C. Atkeson and S. Schaal. Robot learning from demonstration. In *Proc. 14th International Conference on Machine Learning*, pages 12–20. Morgan Kaufmann, 1997.

[2] D. Bentivegna, C. Atkeson, and G. Cheng. Learning from observation and practice at the action generation level. In *IEEE International Conference on Humanoid Robots*, Karlsruhe and Munich, Germany, September/October 2003.

[3] S. Calinon and A. Billard. Incremental learning of gestures by imitation in a humanoid robot. In *Proceedings of the 2007 ACM/IEEE International Conference on Human-Robot Interaction*, Washington, D.C., March 2007.

[4] B. Grosz and C. Sidner. Attention, intentions, and the structure of discourse. In *Computational Linguistics, Vol. 12, No. 3*, September 1986.

[5] S. Iba, C. J.J. Paredis, and P. K. Khosla. Interactive multi-modal robot programming. In *Proceedings of IEEE International Conference on Robotics and Automation*, Washington D.C., May 2002.

[6] S. Lauria, G. Bugmann, T. Kyriacou, and E. Klein. Mobile robot programming using natural language. *Robotics and Autonomous Systems*, 38(3–4):171–181, 2002.

[7] N. Lesh, C. Rich, and C. Sidner. Using plan recognition in human-computer collaboration. In *Proceedings of the Seventh International Conference on User Modelling*, Banff, Canada, June 1999.

[8] A. Lockerd and C. Brezeal. Tutelage and socially guided robot learning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 2004.

[9] M. Nicolescu and M. Matarić. Natural methods for robot task learning: Instructive demonstration, generalization and practice. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Melbourne, Australia, July 2003.

[10] D. Oblinger, V. Castelli, , and L. Bergman. Augmentation-based learning: combining observations and user edits for programming by demonstration. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 202–209, Sydney, Australia, January-February 2006.

[11] C. Rich, C. Sidner, and N. Lesh. Collagen: Applying collaborative discourse theory to human-computer interaction. In *AI Magazine, Special Issue on Intelligent User Interfaces*, November 2001.

[12] P. E. Rybski, K. Yoon, J. Stolarz, and M. Veloso. Interactive robot task training through dialog and demonstration. In *Proceedings of the 2007 ACM/IEEE International Conference on Human-Robot Interaction*, Washington D.C., March 2007.

[13] J. Saunders, C. L. Nehaniv, and K. Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. In *Human-Robot Interaction*, Salt Lake City, Utah, March 2006.

[14] R. M. Voyles, J. D. Morrow, and P. K. Khosla. Towards gesture-based programming: Shape from motion primoridal learning of sensorimotor primitives. *Robotics and Autonomous Systems*, 22:361–375, November 1997.