Interactive Multi-Modal Robot Programming

Soshi Iba [†], Christiaan J.J. Paredis [‡], and Pradeep K. Khosla [†‡]
[†)] *The Robotics Institute*
[‡)] *Institute for Complex Engineered Systems*
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890
iba@ri.cmu.edu, paredis@cmu.edu, pkk@ece.cmu.edu

Abstract

As robots enter the human environment and come in contact with inexperienced users, they need to be able to interact with users in a multi-modal fashion—keyboard and mouse are no longer acceptable as the only input modalities.

This paper introduces a novel approach to program a robot interactively through a multi-modal interface. The key characteristic of this approach is that the user can provide feedback interactively at any time—during both the programming and the execution phase. The framework takes a three-step approach to the problem: multi-modal recognition, intention interpretation, and prioritized task execution. The multi-modal recognition module translates hand gestures and spontaneous speech into a structured symbolic data stream without abstracting away the user's intent. The intention interpretation module selects the appropriate primitives to generate a task based on the user's input, the system's current state, and robot sensor data. Finally, the prioritized task execution module selects and executes skill primitives based on the system's current state, sensor inputs, and prior tasks. The framework is demonstrated by interactively controlling and programming a vacuum-cleaning robot.

1. Introduction and Related Work

In the early years of robotics, teach pendants were the most common mode of interaction for manipulator systems. As software capabilities improved, the ability to do off-line programming proved to be a significant step forward. The current state-of-the-art in robot programming is based on an iconic programming [1,2] and/or programming by human demonstration [3,4]. The goal of these paradigms is to translate the burden of programming manipulator systems from robot experts to task experts. To enable task experts who may not be robot experts to interact with the robot, the interface needs to be intuitive and have the ability to interpret the vague specifications of the user.

Although the common modes of interaction for mobile robots have been a joystick or a mouse assisted by a graphical user interface, increasingly, voice or gestures are introduced as input modalities [5,6]. In this paper, we explore the task of interactively programming a vacuum-cleaning robot. To accommodate novice users, the programming framework should encompass a multi-modal interface and preemptive interaction during both programming and execution.

A programming framework with a multi-modal interface has a distinct advantage over conventional methods in that it is more intuitive and robust. Compared to the traditional input modalities, hand gestures have an advantage in specifying geometric objects and spatial (three-dimensional) data, and are more intuitive for conveying information to robots that exist in the three-dimensional world. For symbolic information, on the other hand, speech input is a natural choice. Although it may seem that combining two error-prone technologies leads to a larger error, recent data by Oviatt [7] shows that fusing two or more information sources can effectively reduce recognition uncertainty, thereby improving robustness.

Preemptive interaction, which enables the user to take over the control of a robot at any time, adds a new flavor to the robot-programming problem. It enables the creation of robot programs through interactive composition of program components. It also enables interactive adjustment of a program and accelerates the system's ability to adapt and learn through continuous interaction. In other words, the framework allows the task expert to "coach" the robot and to make adjustments on-line as it performs the new skill. Iba *et al* [8] demonstrated the significance of such interaction through the use of a preemptive gesture-based control scheme.

2. Framework

The programming approach introduced in this paper offers an intuitive interface for the user and the ability to provide interactive feedback to coach the robot throughout the programming process. As input modalities, we support hand gestures and spontaneous speech.

The framework is composed of three functional modules as illustrated in Figure 1. The first module (multi-modal recognition) translates hand gestures and spontaneous speech into a structured symbolic data stream without abstracting away the user's intent. The second module (intention interpretation) selects the appropriate set of primitives based on the user input, current state, and robot sensor data. Finally, the third module (prioritized task execution) selects and executes primitives based on the current state, sensor inputs, and the task given by the previous step. Each module includes two modes of operation: a learning and an execution mode. Depending on the mode of operation, the overall system can provide interactive robot control, adjustment of primitives, or composition of robot programs.

There are three main reasons for implementing the system in a modular fashion as described in Figure 1. First, the implementation follows a functional decomposition of the problem: recognition, interpretation, and execution. Second, in a modular architecture, one can easily replace the implementations of individual modules. For example, if we were to program an industrial manipulator instead of a vacuum cleaning mobile robot, the task execution module can be replaced by another implementation. Finally, because the first and last module can be implemented as slight modification of existing software and hardware products, a modular implementation allows us to work independently on the intention interpretation module, which is the main focus of our research.

2.1. Multi-Modal Recognition Module

The function of the multi-modal recognition module (the first block in Figure 1) is to translate hand gestures and spontaneous speech into a structured symbolic data stream without abstracting away the user's intent. The symbols could be gestures, words, or both. We consider three sub-functions. First, the module needs to translate incoming audio and gesture signals into a structured stream of word and gesture unit symbols with appropriate parameters. Second, the module needs to be able to adapt to new users by reinforcing symbols during recognition. Finally, the module needs to be able to add new vocabulary on-line.

The recognition module generates a parameterized output stream. Examples of such parameters are the direction and velocity of the hand for a *waiving* gesture, or the designated x-y coordinates on the floor for a *pointing* gesture. The types of input modalities discussed throughout the paper are human voice and hand gestures parameterized by two 22-sensor CyberGlove. Other modalities can replace or be added to the current recognition module.

In the recognition module, it is important not to abstract away the user's intent. The module should only make a probabilistic recommendation of recognition symbols rather than a hard decision. Therefore, this module can be considered as a lower layer of a multi-

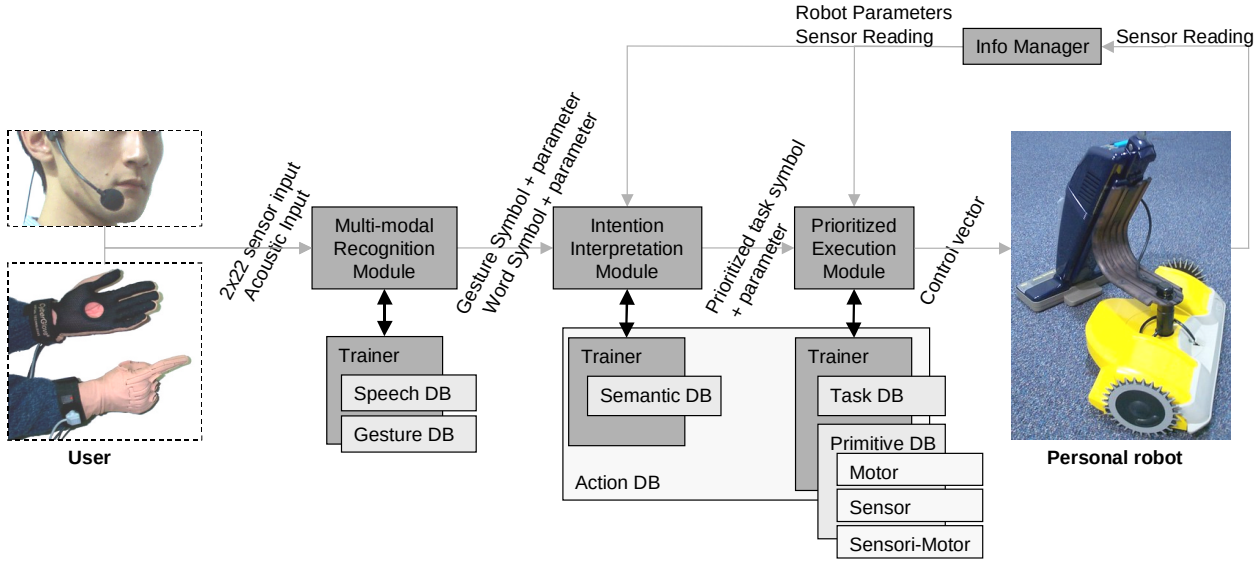

Figure 1: Framework for a Multi-Modal Interface

level decision process, and the high level (hard) decision is made by the intention interpretation module. A similar tactic is often employed in speech and visual recognition problems, where the final interpretation of sensor data is based on the probability propagated through the layers of a higher-level knowledge model.

The second function of the module is to adapt to the data from new users by reinforcing symbols during recognition. The multi-modal recognition module is implemented using a Hidden Markov Model, a stochastic method in which on-line model adaptation and reinforcement are very common.

The third function of the module is to add new vocabulary interactively. Unlike spoken language, the use of gestures is not standardized and listed in dictionaries. Although there exist many task dependent categorizations of gestures, these categorizations do not generalize in a task independent implementation. It is more reasonable to assume that the system is only capable of recognizing a few of the new user's gestures and provide an easy way to register additional gestures.

In our implementation, spontaneous speech is translated into words using an off-the-shelf speech recognition package. For hand gestures, we implemented a word spotting technique using the Hidden Markov Toolkit (*HTK*) by Microsoft. Our current system works only with a basic set of words and gestures and does not include interactive learning of new gestures. Table 1 lists some of the initial candidate gestures and words that such a basic vocabulary could include.

In the future, we plan to rely on cross-modal gestures to implement on-line learning. There are already attempts to automatically discover new gestures [9], however, it is easier to rely on a redundant input mode to manage the learning process. For instance, speech could be used to signal the beginning and end of the learning process for gestures, and vise versa.

Input	*Candidate Symbols*
One-Handed Gestures	Point, Waive, Open, Grasp, Turn, Power Grasp, Precision Grasp
Two-Handed Gestures	Relative Position Specification (*e.g.* two Points)
Speech Vocabulary	Go, Move, Goto, Stop, Turn Forward, Backward, Right, Left Deictics (This, That, There) Names (Yellow, Green, Robot, etc.) Numbers (One, Two, Three, etc.) Sweep, Vacuum, On, Off Program, Execute, Complete

Table 1: Initial Gesture and Speech Vocabularies

2.2. Intention interpretation Module

The intention interpretation module (The second block in Figure 1) has three functions. The first is to select the appropriate task based on the current context. The second is to attach priorities to the task to handle multiple task requests. The third is to expand a semantic database to include new mappings defined by the user.

The problem of intention interpretation can be considered as a mapping problem from the stream of user input, the current state of the system, and the robot sensor data, to the correct robot task. The user input is an incoming stream of structured symbolic data (with parameters) from the multi-modal recognition module. The robot sensor data is an abstracted version of the robot's sensor stream. For a mobile robot, the robot sensor data could include range sensor data, distance to the closest obstacle, the robot's global position and current velocity. For manipulators, the robot sensor data includes the end-effector's position and velocity in the joint space or Cartesian space, and contact data if force/torque/tactile sensors are available.

The output of the intention interpretation module is a task symbol representing a configuration of robot primitives. The usage and definition of the terms *primitive* and *task* are discussed in the next section. In short, a primitive is an encapsulation of a low level robot behavior; that is, a policy $\pi(\boldsymbol{x},t,\alpha)=\boldsymbol{u}$ that maps the state $\boldsymbol{x}$ of a system and its environment into an appropriate action $\boldsymbol{u}$ at a particular time t, with additional parameters α. The task is a robot program composed of various primitives.

A second important function of the intention interpretation module is to prioritize tasks. Not all tasks are of equal importance. For example, the gesture or word that corresponds to an emergency stop has a very high priority, and should be executed even if the robot is already engaged in another task. Similarly, a high-level task, like navigating to a point (x,y), may require assistance from the user to avoid obstacles and dead ends. The task, therefore, has a lower priority than the tasks for interactive user assistance.

Thirdly, the module should support expanding the range of intentions that can be recognized. Expandability is what distinguishes this framework from other systems that can recognize intentions **[10]**. Intention interpretation can be considered as the selection of the correct task policy. Since it is unreasonable to assume that the system designer can anticipate the whole task domain, the set of mapping functions stored in the semantic database should be expandable. This will be accomplished through the interactive addition of mappings. Assume that the voice command "go home" maps to the mobile robot's task primitive that navigates to its home position, and the user wants to map the hand gesture "OK" (assumed to be

recognizable by the recognition module) to this task. The user can train this new mapping, by repeating the "go home" command while making an "OK" sign. Ideally, the semantic trainer should map the "OK" sign gesture to the behavior of navigating to the home position. This particular example illustrates a simple case in which no new behavior is created. A more interesting mapping is described in the next section, where primitives are adjusted, configured and grouped using similar interactive techniques.

2.3. Prioritized Task Execution Module

The prioritized task execution module (the third block in Figure 1) has two functions. The first is to arbitrate and execute primitives based on the current state, the sensor inputs, and the prioritized task given by the previous module. The second is to generate a robot program (task) by configuring primitives.

Before going into the details of each function, we distinguish tasks from primitives based on their levels of abstraction. Primitives are encapsulations of low-level robot behaviors and serve as building blocks of high-level behaviors. They consist of motor (M), sensor (S), or sensori-motor (SM) primitives. Motor primitives generate open loop behaviors that do not depend on sensor feedback. For mobile robots, motor primitives include sensor independent acceleration, stop, turn, beep, and directional motions. Motor primitives for manipulators are purely kinematic. Sensor primitives provide the system with observable sensor signals, such as the current robot position, joint angles, range sensor data, and bumper switch data. Sensori-motor primitives generate closed loop behaviors, such as a guarded move of an end-effecter, or wall following and navigation towards a particular destination for mobile robots. The sensori-motor primitives can be thought of as pre-tailored configurations of motor and sensor primitives.

A task is a configuration of primitives—either a sequence of primitives, or a single primitive. Tasks are stored in a database in the form of a state buffer [11], Markov chain [12] or finite state machine [13]. Because the intention interpretation module requires access to some of the task data also, it shares the semantic, primitive, and task databases with the task execution module, as is shown in Figure 1.

The first important function of the task execution module is task arbitration. As is explained in the section on intention interpretation, not all tasks are equally important. When tasks with different priorities are passed to the prioritized task execution module, the module orders the tasks and executes them according to their priority. The scheme can be described as event driven preemption, where the event (a request from the intention interpretation module to execute a task) triggers an active switch from the running task with lower priority to another with higher priority. This allows the user to handle situations such as making an emergency stop or avoiding an obstacle during the execution of other tasks.

The second function is to generate a robot program (task) interactively. The basic approach is to take a coaching strategy using a redundant input mode. The user sets the module to a learning mode and executes primitives sequentially; the system remembers the sequence as a task. There are two obvious problems to this approach. The first problem is that the robot programs are not always sequential due to conditional branching and looping. It is not desirable to force the user to remember a special gesture command since that would make the system counter intuitive. A tool to convey a program structure and an intuitive interface to edit the program are necessary, unless the system can infer such conditions from multiple examples. The second problem is the lack of generality. The task would be useless if it would only work for the particular parameter value for which it was trained. Somehow, the user must let the module know that some attributes can be generalized while retaining others as important features of the task. For the point and navigate task, the goal coordinates should be variable, while the sequence of primitives used to navigate needs to be retained. Making this distinction is the subject of future research.

Module	*Input*	*Function (Execution and/or Learning mode)*	*Output*
Multi-modal Recognition	CyberGlove-R Polhemus-R CyberGlove-L Polhemus-L Acoustic (8bit-16KHz)	• Translate incoming audio and gesture signals into a structured stream of word and gesture unit symbols with appropriate parameters. (*E*) • Reinforce models during recognition (exec. & learn) • Add new vocabulary on-line. (*L*)	Gesture Symbol-R + param. Gesture Symbol-L + param. Word Symbol + param
Intention Interpretation	Gesture Symbol-R + param Gesture Symbol-L + param Word Symbol + param Robot Data Robot Position Robot Velocity Sensor Readings Knowledge of its current state	• Select the appropriate primitives based on the user input, current state, and robot sensor data. (*E*) • Prioritization of tasks, according to the database (*E*) • Expand semantic database to adjust to new mappings the user comes up with. (*L*)	Task symbol + priority + param
Prioritized Task Execution	Robot Status Sensor Readings Task symbol + priority + param	• Arbitrate and execute primitives based on current state, sensor input, and the prioritized task given by the previous module. (*E*) • Generate a robot program (task) by configuring primitives. (*E* & *L*)	Control vector

Table 2: Functional Summary

Table 2 summarizes the functions offered by each of the three modules in the framework. Three modules work synchronously in the continuous flow of data to provide intuitive multi-modal interaction and programming of robots.

3. Experiment

The main focus at this stage of our research is to establish the connections between all three modules and illustrate the overall operation of the framework with a basic interactive programming example. The framework is implemented using a Cye vacuum cleaning robot, two 22-sensor CyberGloves, and a microphone. We modified the graphical user interface provided with the vacuum cleaning robot, to accept hand gestures and speech input, while retaining its original functionalities: mapping, iconic programming, and path-planning [14]. As a result, Cye can be controlled via mouse, speech, and hand gestures.

The multi-modal recognition module is implemented using the *Sphinx-II* speech recognition engine and the Microsoft Hidden Markov Toolkit (*HTK*) that has been customized to recognize gestures at 10Hz. A discussion of the gesture recognition methodology is outside the scope of this paper; however, the method is similar to the one in [15], where parameters of Hidden Markov models for each gesture are obtained from known strings of gesture examples. Each gesture consists of gesture phonemes that take into account finger-joint positions, joint velocities, the hand's Cartesian position and velocities. The vocabulary of gestures is listed in Table 1. The on-line addition of vocabulary is not implemented at this point.

The intention interpretation module is implemented with a semantic database (Table 3). The semantic database connects inputs such as gesture and speech symbols, the robot's sensor readings, and the current state to the most likely task for the robot to execute. A task, which can be considered as a robot program, is a set of one or more primitives. Each task has predefined priorities attached to specify the importance of the task over the others in the event of preemption. At this point, the semantic database is fixed and does not support the

Input Symbol ("voice" and 'gesture')	*Candidate Task*	*Priority*
"Stop" or two 'Closed' fists	Stop()	High
"Go" + "This", "That", *etc* + 'Point'	Goto(*P*)	Medium
"Go" or "Move" + Direction ("Right" "Forward", "Left", "Back")	Move(*v*)	Medium
'Waive' or "Go" + 'Waive'	Move(*v*)	Medium
"Go Home"	Gohome()	Medium
"Vacuum" + "On" or "Off"	Vacuum(On/Off)	Medium
"Turn" or "Turn" + direction ("Right", "Left)	Turn(ω)	Medium
"Cover Area" + two 'Point's	AreaCoverage (P_1,P_2)	Medium
"Program" *n*	Program a task *n*	Low
"Complete" *n*	End of program	Low
"Execute Program" *n*	Execute a task *n*	Low

Table 3: Semantics Database

Primitive	*Parameter*	*Action*
Goto	*Position P*	Move to the position, w/path-planning
Vacuum	*On/Off*	Toggle the state of the vacuum cleaner
AreaCoverage	Rectangular Area (P_0, P_1)	Traverse the area specified
GoHome	*N/A*	Move the robot back to the home position
Move	*Velocity(v)*	Apply additional velocity *v* to the robot
Turn	*Angular Velocity(ω)*	Apply additional angular velocity ω to the robot
Stop	*N/A*	Stops the robot motion

Table 4: Primitives Database

on-line addition of entries.

The prioritized task execution module handles tasks by taking primitives out of the task, and executing them according to their priority. The primitives used in the current scenarios are listed in Table 4. Primitives such as *GoHome* and *AreaCoverage* provide high-level navigation, whereas primitives such as *Move*, *Turn*, and *Vacuum* give low-level control of the robot. Primitives are executed in order of arrival except when a high-priority task is introduced; such tasks pre-empt the current task and execute immediately. In the current implementation, tasks are represented as a set of primitives governed by a finite state machine.

For the current implementation, we have considered two interactive programming scenarios. The first scenario is to have a user register numerous via-points to which the robot should navigate using its path planning capability. The second scenario is to use a two-handed gesture to specify an area that the robot should vacuum; the robot then vacuums the area using its area coverage primitive. In both scenarios, the robot can accept the user's preemptive speech and hand gesture commands to deal with unforeseen events. Figure 2 and Figure 3 illustrate the sequences of the first and second scenario. Each figure contains a sequence of camera snapshots with the corresponding conceptual illustrations of the framework, and the cropped images of the GUI.

In the first scenario, the user first verbally commands that the subsequent actions be stored as "Program One". The user then executes the *Goto* primitive by combining the voice command "Go There" with the gestural command 'Point' to indicate the destination. In general, deictic terms such as "This", "That", and "There" must be accompanied by a referential gesture to specify the corresponding task parameters. For the *Goto* primitive, the Cartesian coordinates are extracted from the intersection between the extension of the index finger and the ground [8]. In step 2, the user enters another *Goto* primitive but with a different end-position. After having saved these two primitives in "Program One" with the "Complete" command, the user can re-execute the program through with the voice command "Execute Program One". However, in step 4, when the robot

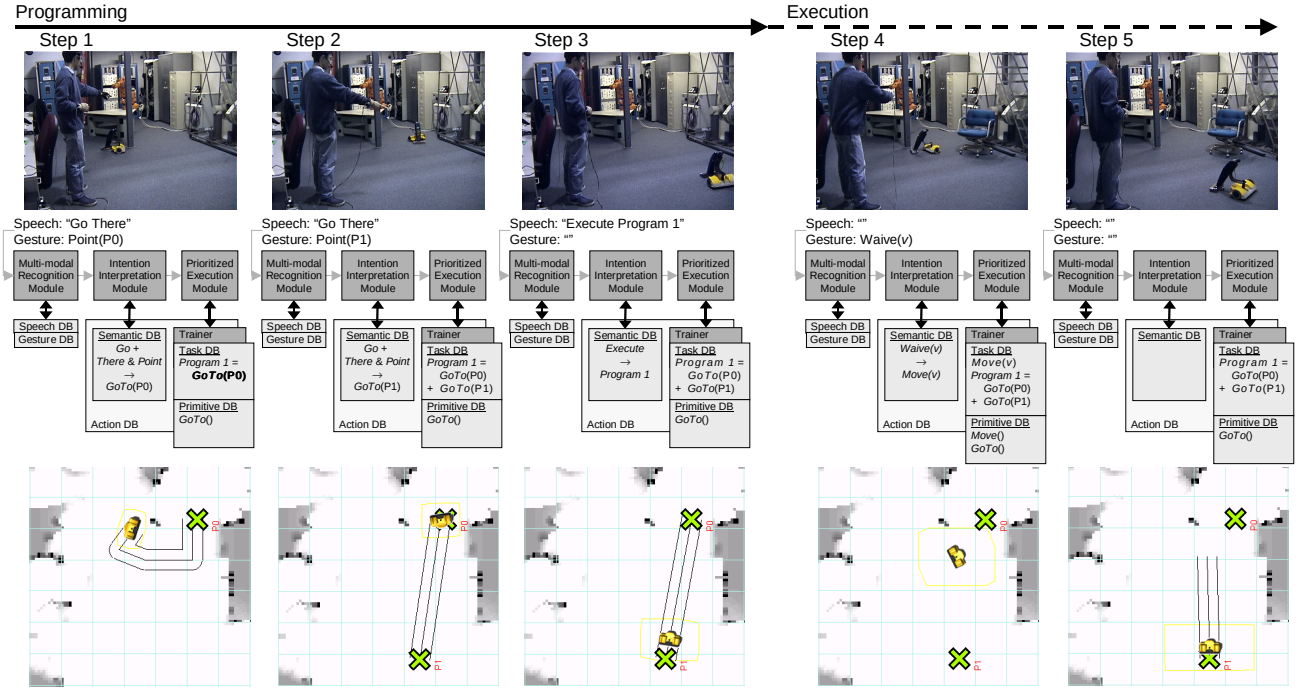

Figure 2: Experiment Scenario 1

navigates to the second position from the first, it encounters an unknown obstacle. At this point, the user gestures the 'Waive' command, which has a higher task priority and can be used to control the robot around the obstacle. When the obstacle has been cleared and the user stops waiving, the robot returns to the execution of "Program One" (Step 5).

In the second scenario, illustrated in Figure 3, the user defines the task "Program Two." After turning on the vacuum attachment with the voice command "Vacuum On" (step 1), the user issues the *AreaCoverage* command with one two-handed gesture; each hand performs a 'Point' gestures to specify the diagonally opposite corners of the area (with the direction aligned along the axes of the GUI). Steps 3 to 5 show the execution of the *AreaCoverage* command. As in the first scenario, at any point can the user re-execute "Program Two", interrupt the execution, or interactively adjust the execution with higher-priority commands.

4. Conclusion and Future Work

In this paper, we have described the overall framework for interactive multi-modal robot programming and have illustrated the framework with preliminary result from two programming scenarios: point-to-point navigation and area coverage. Although the current implementation is limited in the sense that none of the databases, except for the task database, can be expanded on-line, we feel that this preliminary result clearly illustrates the usefulness of multi-modal interaction, including the capability to interrupt commands preemptively.

To obtain a comprehensive multi-modal interactive robot programming system, several elements still need to be added in the future. Although the programs generated by the current system can be re-executed, they are limited to fixed task sequences. To expand the generality of the paradigm, we need to add the ability to re-configure the task parameters interactively and define non-sequential flow structures such as conditional branching and looping. Secondly, we plan to investigate how the system can determine the most likely high-level goal the user is trying to achieve, given a limited, initial sequence of task primitives. Lastly, the system performance needs to be quantitatively evaluated through user studies, to determine the benefits provided by multi-modal programming, interactivity, and intention interpretation.

5. References

[1] Anderson, R. J., Shirey, D. L., and Morse, W. D., "A telerobot control system for accident response," *IEEE International Conference on Robotics and Automation*, San Francisco, CA, USA, pp. 2000-6, 2000.

[2] Gertz, M. W., Stewart, D. B., and Khosla, P. K., "A human machine interface for distributed virtual laboratories," *IEEE Robotics & Automation*

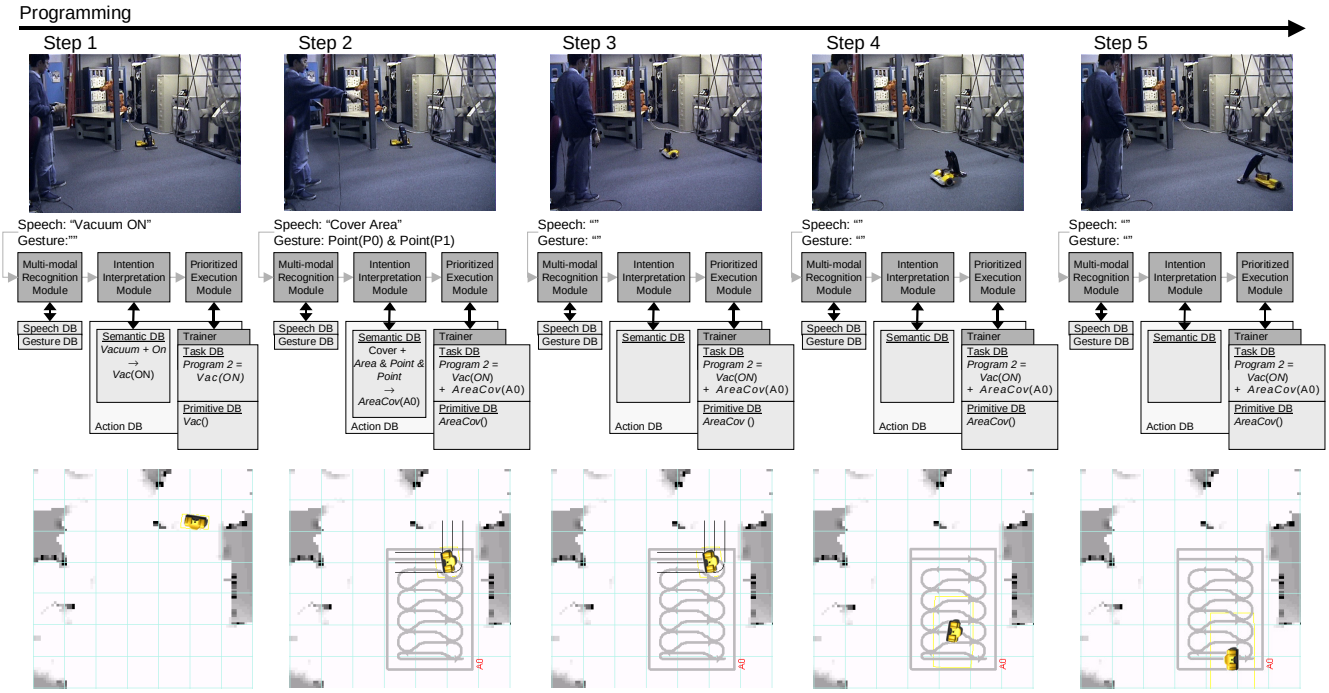

Figure 3: Experiment Scenario 2

Magazine, vol. 1, no. 4, pp. 5-13, 1994.

[3] Kang, S. B. and Ikeuchi, K., "Toward automatic robot instruction from perception-mapping human grasps to manipulator grasps," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 1, pp. 81-95, 1997.

[4] Voyles, R. M. and Khosla, P. K., "Gesture-based programming: a preliminary demonstration," *IEEE International Conference on Robotics and Automation*, Detroit, MI, USA, pp. 708-13, 1999.

[5] Fong, T., Conti, F., Grange, S., and Baur, C., "Novel Interfaces for Remote Driving: Gesture, Haptic and PDA," *SPIE Telemanipulator and Telepresence Technologies VII*, Boston, MA, 2000.

[6] Perzanowski, D., Schultz, A. C., Adams, W., and Marsh, E., "Goal tracking in a natural language interface: towards achieving adjustable autonomy," *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pp. 208 -13, 1999.

[7] Oviatt, S., "Taming recognition errors with a multimodal interface," *Communications of the ACM*, vol. 43, no. 9, pp. 45-51, 2000.

[8] Iba, S., Vande Weghe, J. M., Paredis, C. J. J., and Khosla, P. K., "An Architecture for Gesture-Based Control of Mobile Robots," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Kyongju, Korea, pp. 851-57, 1999.

[9] Wren, C. R., Clarkson, B. P., and Pentland, A. P., "Understanding purposeful human motion," *Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, pp. 378 -83, 2000.

[10] Yamada, Y., Umetani, Y., Daitoh, H., and Sakai, T., "Construction of a human/robot coexistence system based on a model of human will-intention and desire," *IEEE International Conference on Robotics and Automation*, Detroit, MI, USA, pp. 2861-7, 1999.

[11] Kimura, H., Horiuchi, T., and Ikeuchi, K., "Task-Model Based Human Robot Cooperation Using Vision," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Kyongju, Korea, pp. 701-06, 1999.

[12] Rybski, P. E. and Voyles, R. M., "Interactive task training of a mobile robot through human gesture recognition," *IEEE International Conference on Robotics and Automation*, Detroit, MI, USA, pp. 664-9, 1999.

[13] Morrow, J. D. and Khosla, P. K., "Manipulation task primitives for composing robot skills," *IEEE International Conference on Robotics and Automation*, Albuquerque, NM, USA, pp. 3354-9, 1997.

[14] Batavia, P. H. and Nourbakhsh, I., "Path planning for the Cye personal robot," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 15-20, 2000.

[15] Ogawara, K., Takamatsu, J., Iba, S., Tanuki, T., Kimura, H., and Ikeuchi, K., "Acquiring hand-action models in task and behavior levels by a learning robot through observing human demonstrations," *IEEE-RAS International Conference on Humanoid Robots*, Boston, 2000.