GESTURE-BASED PROGRAMMING FOR ROBOTICS: HUMAN-AUGMENTED SOFTWARE ADAPTATION

Richard M. Voyles
Computer Science and Engineering
University of Minnesota
Minneapolis, Minnesota

J. Dan Morrow
Universal Instruments, Inc.
Binghamton, New York

Pradeep K. Khosla
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

Gesture-Based Programming is a paradigm for programming robots by human demonstration in which the human demonstrator directs the self-adaptation of executable software. The goal is to provide a more natural environment for the user as programmer and to generate more complete and successful programs by focusing on *task experts* rather than *programming experts*. We call the paradigm "gesture-based" because we try to enable the system to capture, in real-time, the *intention* behind the demonstrator's fleeting, context-dependent hand motions, contact conditions, finger poses, and even cryptic utterances in order to reconfigure itself. The system is self-adaptive in the sense that knowledge of previously acquired skills (sensorimotor expertise) is retained by the system and this knowledge facilitates the interpretation of the gestures during training and then provides feedback control during run-time.

1. INTRODUCTION

The text-based programming paradigm relies exclusively on the expertise of the human programmer and his or her ability to learn from and remember past lessons on system development. Beyond on-line help files, the applications themselves and the programming environments that facilitate their creation acquire no knowledge of themselves and carry-over no useful knowledge from project to project or even within the development of a single project. The burden of retaining the expertise that results from the creation of new applications or new configurations of existing applications falls entirely on the human programmer.

The idea behind self-adaptive software is to empower the system itself to participate in its own development. If a software environment can retain a knowledge base of its own capabilities and usefulness, it can assist the human developer during re-application or reconfiguration and potentially perform autonomous adaptation. The allowable degree of "self-determination" will certainly vary across applications and operating environments. For example, the user interface to a consumer application program may have wide latitude for self-adaptation on an ongoing basis. A manufacturing system, on the other hand, would likely have rigid constraints on the degree of allowable self-adaptation and even on the periods during which self-adaptation is enabled.

The focus of this paper is not the grand goal of self-adaptive software, but a smaller step toward it we call human-augmented adaptation. We further limit our efforts to the task of robot programming. In particular, we are interested in the programming paradigm itself and in finding alternative approaches to programming that are more intuitive for human users. Our basic model is human-to-human training, which is very intuitive to users. In this model, the trainee is active in the process, so, when applied to the programming of robotic systems, it naturally leads to systems that are more participative in their own development. In this sense, human-augmented adaptation is an important step toward, and a useful complement to, self-adaptive software.

This approach is bolstered by successful examples of robots in the manufacturing sector. Two of the most common industrial applications are spray painting and spot welding. Both of these use natural programming methods such as "lead-through teaching" [Todd 1986] (a primitive form of human demonstration) and point-teaching, respectively, that are easily mastered by semi-skilled workers because of their intuitiveness. Lead-through teaching, in particular, allows anyone who can perform a "valid" task to program the robot to perform the same task simply by demonstrating it. (The same as in human-to-human training.) As such, programming changes can be made reactively or proactively by a *task expert*, not by calling in a *programming expert*. Spray painting is unique among the successful robotic applications because, unlike applications such as spot welding, it involves *skill transfer* from the teacher (i.e., programmer) to the robot. Lead-through teaching provides an intuitive mechanism for a task expert to accomplish this. Unfortunately, it is limited to simple tasks the skill component of which is purely kinematically encoded.

Gesture-Based Programming (GBP) leverages this approach to more complex tasks -- in particular, tasks that involve contact with the environment. It is a paradigm that integrates object-oriented controller design, sensorimotor primitive and robotic skill encapsulation, inconic sub-task specification, and intuitive human/computer interaction. This approach allows the rapid tasking

and re-tasking of complex systems for meaningful, *contact-intensive* applications requiring skill transfer. The paradigm builds on prior work in layered, multiprocessor, real-time operating system development, multi-agent control architectures, graphical programming tools, human gesture interpretation, and robotic *instruction* by human demonstration (as opposed to programming by human demonstration). We will summarize these foundational works and their results as well as describe our efforts to integrate the complete system.

2. GESTURE-BASED PROGRAMMING OVERVIEW

GBP attempts to leverage the benefits of lead-through teaching, mentioned in the spray painting example above, to a much higher level. We want to enable task experts in contact-intensive applications, such as assembly, to program robots by simply demonstrating the tasks with which the experts are so familiar. This is intuitive for the teacher because demonstration followed by supervised practice is the most effective method of skill transfer between humans themselves [Patrick 1992].

However, teaching by demonstration requires a “shared ontology” -- common knowledge about the world that is retained by *both* teacher and student. Among humans, this equates to a set of shared experiences that result in autonomous skills we use for interacting with the world around us. To demonstrate the assembly of a carburetor, for example, it is not necessary to teach the mating of planar surfaces, the insertion of a peg into a hole, or the tightening of a screw. These are basic skills humans acquire during their day-to-day experiences and become part of the shared ontology between teacher and student. These shared skills, even though we may implement them slightly differently from person to person, are important in interpreting the demonstration and must be retained by both parties. When programming robots by demonstration, it is also necessary to assume a set of previously acquired skills that exhibit some commonality between system and system developer. This implies the system must not only be programmed, but it must learn from the process of being programmed.

To understand the paradigm, let us assume the existence of the requisite shared ontology based of *a priori* skills. A major portion of this paper is dedicated to explaining how that expertise database evolves but, for now, assume it is given. The act of programming begins with a human performing the task. (Illustrated by the stick figure on the left of Figure 7.) Observation of the human’s hand and fingertips is achieved through a sensorized glove with special tactile fingertips. The modular glove system senses hand pose, finger joint angles, and fingertip contact conditions. Objects in the environment are sensed with computer vision and, though not described in this paper, a commercial speech recognition system can extract “articulatory gestures” to provide model-based data. (Gestures will be described later.) Primitive gesture classes are extracted from the raw sensor infor-

mation and passed on to a gesture interpretation network. The agents in this network extract the demonstrator's "intentions" -- defined as the underlying skills that produced the motions of the demonstrator as opposed to the observable motions themselves -- based upon the knowledge they have previously stored in the system's skill library from prior demonstrations. Like a self-aware human trainee, the system is able to generate an abstraction of the demonstrated task, mapped onto its own skills. In other words, the system is not merely remembering everything the human does, but is trying to understand -- within its scope of expertise -- the subtasks the human is performing ("gesturing"). These primitive capabilities in the expertise database take the form of *encapsulated expertise agents* -- semi-autonomous agents that encode sensorimotor primitives and low-level skills for later execution. (Encapsulated expertise agents are described in sections 4 and 5.)

The output of the GBP system is the executable program for performing the demonstrated task on the target hardware. This program consists of a network of encapsulated expertise agents of two flavors: a primary set for acting and a secondary set for monitoring. The primary agents implement the primitives required to perform the task and come from the pool of primitives represented in the expertise database. The secondary set of agents includes many of the same gesture recognition and interpretation agents used to interpret the demonstration. These agents perform on-line observation of the human to allow supervised practicing of the task for further adaptation. (Stick figure on the right of Figure 7.)

As mentioned above, the human model for teaching by demonstration most often involves a practice phase. The reason for this is that passive observation of a task rarely provides accurate parametrization for the trainee's deduced task "model" (in this case, the model is represented by the collection of primary encapsulated expertise agents) and sometimes the deduced model is wrong (e.g. missing or incorrect encapsulated expertise agents). Incorporating gesture recognition and interpretation agents into the executable provides an intuitive way for the demonstrator -- or another user -- to direct the further adaptation of the program without having to demonstrate the entire task over again. Because all our agents are implemented as autonomous software modules, these observation agents can easily be disabled without recompiling the program. This might be done as a quality measure to ensure consistent operation of the robot or it might be done as a security measure to prevent sabotaging the robot program.

In the real world, it will not be possible to represent most useful tasks with one network of encapsulated expertise agents. Therefore, it is necessary to segment the demonstration into a series of discrete subtasks (e.g. a grasping subtask followed by a manipulation subtask). Each subtask will be embodied by a network as described above. In this case, the executable program will consist of a sequence of networks rather than a single, static network.

3. RELATED WORK: PROGRAMMING PARADIGMS

Numerous attempts have been made to ease the discomfort of robot programming with varying degrees of success. Behavior-based [Brooks 1986] and multi-agent systems [Wooldridge and Jennings 1995] seek to modularize software for easier programming by programming experts. Visual programming environments like Chimera/Onika [Stewart, Schmitz and Khosla 1992][Gertz, Stewart and Khosla 1993] and commercial packages such as MatrixX/SystemBuild [Integrated Systems], ControlShell [Real-Time Innovations], and LabView [National Instruments] capitalize on modular, reconfigurable software blocks by iconifying them within "software assembly" environments. These visual environments allow programming at a much higher level, hiding many details of the particular implementation, and lessening the burden of programming expertise. For convenience, they also provide point-and-click interaction during run-time.

A more recent approach to human/robot interaction is the field of *learning by observation* [Kang and Ikeuchi, 1996],[Kuniyoshi, Inaba and Inoue 1994], [Kaiser and Dillmann, 1996]. By forcing the robot to observe a human interacting with the world, rather than forcing the human to interact with a *textual representation* of the robot interacting with the world, a more natural, "anthropocentric" environment results. This approach is much like lead-through teaching and, in fact, most of these systems are kinematically-based and operate off-line; if the robot misinterprets the desired trajectory, the whole sequence must be re-taught.

4. SENSORIMOTOR PRIMITIVES

A sensorimotor primitive [Morrow and Khosla 1995] is an encapsulation of sensing and action which can form *domain-general* building blocks for task strategies. A primitive is not an autonomous agent which can intelligently intervene at appropriate places in the task, but rather a powerful and task-relevant command. The goal is to provide a library of sensor-driven commands which effectively integrate sensors into a robot system for a particular class of tasks. In a subsequent section, we will discuss encapsulating the *expertise* of a primitive within an autonomous agent which we will call an "encapsulated expertise agent".

In previous sections, we used the term "skill" loosely. We hereby differentiate between skills and primitives. A skill is a parameterized, stand-alone, robust solution to a specific task. A sensorimotor primitive is a parameterized, domain-general command which integrates sensing and action and can be applied to many skills within a task domain. The goal of sensorimotor primitives is to capture important domain information or capabilities so that robust skills can be quickly developed. Therefore, skills are generally composed of multiple sensorimotor primitives.

Primitives can be encapsulated as port automata. When implemented as independent periodic tasks under our Chimera real-time operating system [15], they acquire specific methods for real-time task management. We call these software modules "port-based objects" [16]. Skills can then be encapsulated as tightly-coupled groups of port-based objects and both primitives and skills populate the "expertise database" for GBP.

5. ADAPTING THE EXPERTISE DATABASE

From the overview description of GBP it should be clear that the nature of the expertise database is critical to the domain of application of the programming system. For example, if the expertise database only contained primitives for kinematic motion of the manipulator, a system for lead-through teaching would result. (As in the spray painting example above.) This is a good starting point because it is our goal to emulate the basic ideas and intuitiveness behind the success of lead-through teaching. However, it is also our goal to extend this success to new, more complex domains. To accomplish this we must adapt the expertise database to the new domain and we employ two distinct approaches for so doing: explicit programming and learning.

5.1Adapting the Expertise Database Through Explicit Programming

Adapting the expertise database can mean two things. It can either mean modifying existing primitives to accommodate a new domain or adding entirely new primitives to the existing set. We focus on the latter because we're primarily interested in extending into new domains and capabilities. Modifying existing primitives involves much subtler issues of deciding which primitive to modify and when it needs modification. For example, consider a robot that already has "guarded move" (approach until a force threshold is reached) and "straight-line move" primitives in its repertoire. Suppose it then observes a demonstration in which two parts approach each other obliquely and then slide across one another, maintaining a constant force threshold (as in erasing a chalkboard). This has both similarities and differences to the guarded move and straight-line move primitives and can be dealt with in three distinct ways:

- first, assume the observation is a skill composed of some combination of the two lower-level primitives and attempt to interpret the contribution of each raw primitive to the higher-level skill

then

- assume the observation is a new primitive and record it as such

or

- assume the observation is a new instance of one of the existing primitives and incrementally re-teach the corresponding primitive to include these new observations

The first approach is the identification problem and does not involve adapting the expertise database. See [22] for a description of our approach to the identification problem. The second and third approaches involve adapting the expertise database, but in approach three, how does one determine which primitive to tune? Presumably, if we can identify the components it is not necessary to tune the existing primitives. But if the identification approach fails, the new observation is not sufficiently similar to either existing primitive. These issues are still open areas of research and their solution will result in true software self-adaptation. For the purposes of this paper on human-augmented adaptation, we will focus on the problem of acquiring new primitives as identified by the human.

Explicit programming involves a human "oracle" determining a relevant set of primitives required for a particular application domain and then coding those primitives by hand. This differs from creating a monolithic application program only in the decomposition of the task. Primitives are decomposed and instantiated as parametrizable but structurally invariant code blocks with broad utility within an application domain. To extend the capability of the system to the domain of connector insertion tasks, several primitives were conceived and coded in addition to the basic cartesian and joint-interpolated move commands. These new primitives (described in detail in [Morrow and Khosla, 1995]) include:

- rotational and linear dithers
- a guarded move (move until contact is acquired)
- a "stick" move (move until contact is lost)
- correlation (active sensing that correlates actuator commands with sensor values)

These new primitives have been rapidly assembled to successfully complete a range of tasks within the domain of connector insertions including BNC connectors and a variety of different size D-connectors. The "software assembly" [16] was performed manually through the Skill Programming Interface (SPI).

The SPI is a graphical environment for simplifying the construction of robotic skills and their encapsulation into encapsulated expertise agents. It assumes a set of sensorimotor primitives is available and provides a graphical method of assembling them into agents as shown in Figure 10. Figure 10 is a screen dump of the SPI user interface and depicts a graphical representation of the BNC insertion skill. Each "bubble" represents an agent that consists of several primitives. The configuration of each agent is shown in Figure 1 along with the commands each produced during an experimental run of the robot system. Many complex robotic skills have been created using this approach including the connector insertion skills mentioned above using force and/

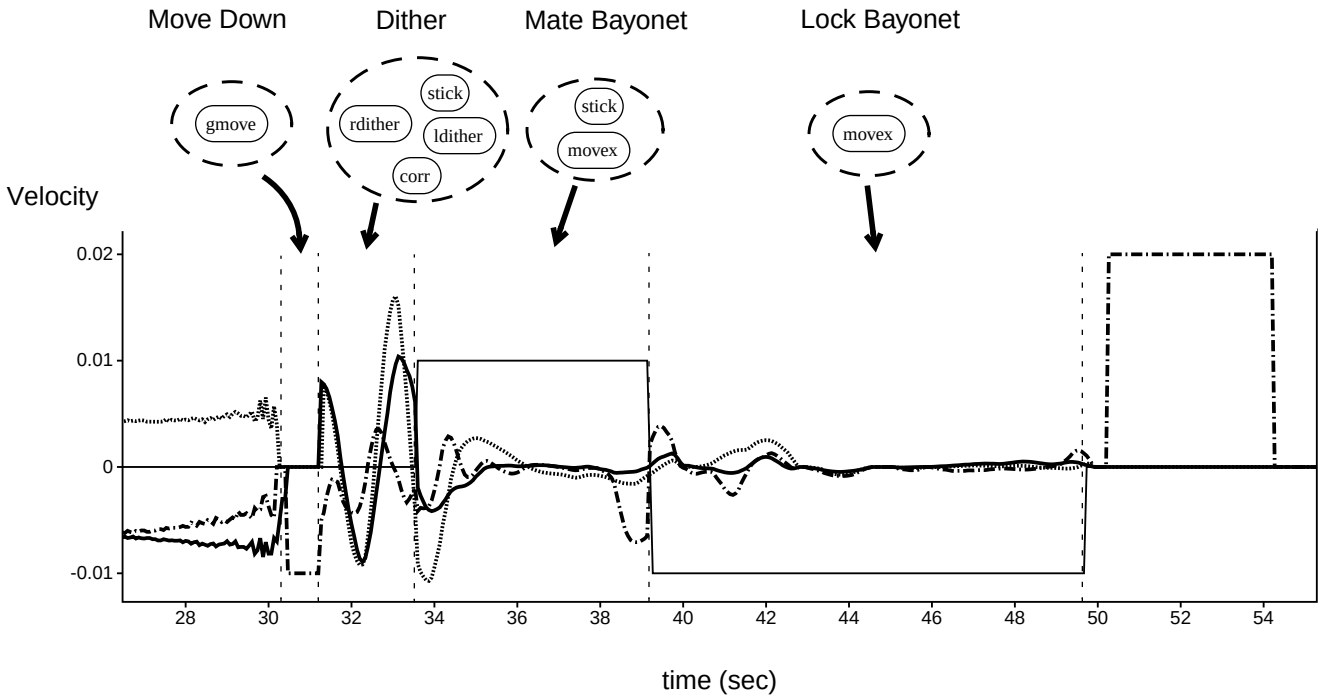


FIGURE 1: VELOCITY COMMAND PLOTS FOR BNC INSERTION SKILL ANNOTATED WITH THE AGENTS (COLLECTIONS OF SENSORIMOTOR PRIMITIVES) THAT PRODUCED THE COMMANDS

or vision feedback. The details of the skill implementations and their performance can be found in [Morrow and Khosla 1995] and [Morrow, Nelson and Khosla 1995].

The above primitives execute on the robot and become part of the expertise database. We have also explicitly coded a handful of primitives for gesture recognition tasks in the domain of kinematic motion commands. These primitives do not directly control the robot and, therefore, are not part of its sensorimotor expertise. Instead, they operate to identify intentions of the human during demonstrations and may operate in parallel to fine-tune operation during practice. These primitives include recognition of the following types of gestures:

- straight-line motion gestures (gross movements of the hand)
- turning in place motion gestures (gross re-orientations of the hand)
- attenuating tactile gestures (“nudges” sensed through force/torque transducers opposing motion)
- magnifying tactile gestures (“nudges” sensed through force/torque transducers enhancing motion)
- contact tactile gestures (static contact sensed through force/torque transducers)

The interpretation of intention with respect to these types of gestures is application dependent and outside the scope of this paper, but is described in [23]. To summarize, sub-networks of primitives for parsing raw sensor streams into “gestural words”

are grouped with primitives for interpreting those gestures in context within simplified “gestural sentences.” The interpretation primitives use internal fuzzy models to support or refute hypotheses on the demonstrator’s intention. These multi-agent interpretation networks are based on the fine-grained tropism system cognitive architecture [23][1] which selects actions based on stochastic beliefs (in this case, built from the fuzzy models). The result is used to help identify action primitives or to parametrize corrective feedback during practice. For example, we have used these primitives to interpret physical nudges on the robot end effector to fine tune the trajectory [23].

5.2Adapting the Expertise Database Through Learning

The other approach to adapting the expertise database is through learning. Again, for the purposes of this paper, we are concerned with adaptation through the addition of primitives, not through modification of existing primitives. (Although, this learning paradigm can be applied to the latter.) We have developed a technique called *shape from motion primordial learning,* based on eigenspace analysis, for human augmented adaptation of the system’s internal expertise database. Closely related to a technique in computer vision [Tomasi and Kanade, 1992], the technique extracts the intrinsic relationships between sensor inputs and actuator commands via principal components analysis. By demonstrating several examples of a manipulation primitive (fewer than ten), the human teaches the robot to execute the primitive on its own, adding a new capability to its repertoire.

Learning is the result of identifying patterns of sensory input that are correlated to actuator output. The eigenspace captures the principal component vectors -- or dominant correlations -- of a dataset. We applied this technique to an anthropomorphic robot arm within the domain of simple force-based manipulations. Sense data includes resolved force/torque components and the position information of the robot itself. For our experiments there were no redundant groups of very similar and correlated sense elements such as a visual retina (as in [Hancock and Thorpe, 1995]) or sonar ring (as in [Pierce, 1991]) as has been employed in other eigenspace approaches. The only structure imposed at all on the learning process is linearity, so the result learned is dependent on what the robot is allowed to observe. We use teleoperation of the robot to demonstrate each primitive which allows the robot to observe both sensory inputs and the actuator outputs of the expert demonstrator.

The training data consists of a matrix of input/output vectors sampled periodically during teleoperation. The input/output vector is composed of the actuator commands concatenated to the sensor data. The mean of this vector over the entire sequence was subtracted out to normalize the data values. Next, the matrix was batch-processed using singular value decomposition to extract the eigenvectors and the largest n eigenvectors were selected using the largest ratio of adjacent singular values as the threshold for n. Underlying this heuristic threshold is the assumption that there is a group of “significant” eigenvectors with high

correlation (large singular values) and there's a bunch of uncorrelated noise with small singular values. Examining the ratios of adjacent, ordered singular values indicates the dividing line between the groups.

After training, the learned eigenvectors serve to implement the primitive at runtime by reprojection of new sense vectors. The orthogonal eigenvectors define a "correlation space" between sensors and actuators. By projecting new sensor vectors onto this space, actuator commands can be extracted. This projection is described mathematically as:

$$\mathbf{v} = actuator\left(\mathbf{a} + \sum_{i=1}^{n} ((\mathbf{x} - sensor(\mathbf{a})) \bullet sensor(\mathbf{e}_i))\mathbf{e}_i\right)$$

where **v** is the actuator command vector, **x** is the new sensor vector, **a** is the average vector determined during training, and $\mathbf{e}_i$ is the set of learned eigenvectors. Summarizing this equation, one sequentially takes the dot product of the new sensor vector and the sensor part of each and every eigenvector. The result of each dot product produces a scale factor that is applied to each and every eigenvector which are summed across all scale factors. The "actuator part" of the resultant sum plus average vector is the actuator command.

5.3Guarded Move

As a first demonstration, we attempted to learn a one-dimensional guarded move in the manipulation domain. Although this is a fairly trivial primitive that can easily be hand-coded, it is not quite as obvious how to robustly identify it during human demonstration. This is the strength of an integrated approach like shape from motion primordial learning; it provides primitive identification and transformation (more on this later) as well as the basic learning of the primitive.

To learn "**zguard**" -- a guarded move along the z-axis -- we teleoperated a PUMA robot with a force/torque sensor (calibrated with the shape from motion paradigm, incidentally), making it come in contact with the table and pressing on it with a specified force. We repeated this one-dimensional guarded move ten times, logging data only during the guarded move, not during the retraction phase when the gripper was moved away from the surface. Teleoperation input was provided by a 6-axis joystick and operator feedback was visual inspection of the task as well as a real-time graphical display of the force/torque components.

The input/output vector from which the data matrix for training was generated consisted of the useful data -- 6 measured force components, the total force and torque magnitudes, and 6 commanded cartesian velocities -- plus some irrelevant data -- 3 cartesian position elements and 9 cartesian orientation elements. The irrelevant data was thrown in to demonstrate that we were not implicitly supplying additional structure for the learning algorithm by limiting its observations.

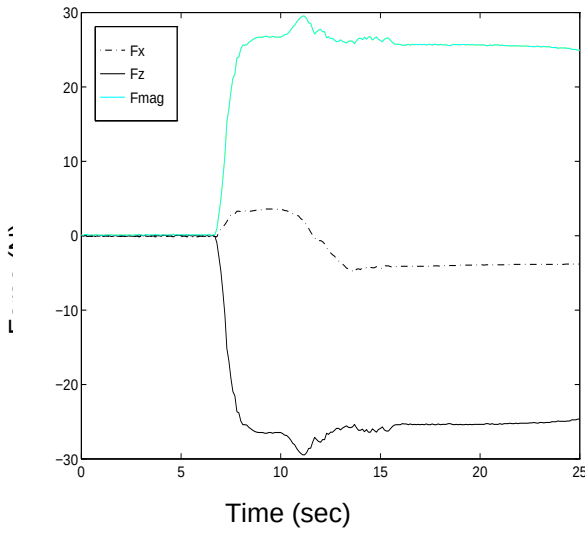
The output of the training was a primitive of one eigenvector that performed very well when embedded by hand into a port-based object. Figure 2 shows the measured force components as the guarded move primitive autonomously acquires a hard surface. 20 N was the target force threshold applied during training and this can be varied by scaling the output portion of the extracted eigenvector. The primitive worked equivalently in all trials performed regardless of region of workspace, orientation of the end effector, compliance of the surface, or external perturbations. Analysis of the meaning of the components of the eigenvector support this. The primitive is looking at the *z*-component of force as well as the total magnitude of the force, much as a hand-coded damping control law would do to accomplish the same task.

5.4Edge-to-Surface Alignment

The guarded move is a “move until force threshold” operation. Movement is an explicit part of the goal. Alignment operations, on the other hand, only move in order to accommodate. Since no explicit motion is part of the primitive, it is difficult to teleoperate. What we’d like to do is use the previously learned guarded move to bring the edge and surface together while we teleoperate only the alignment primitive. This makes the training task much easier for the human.

Learning the “**yroll**” alignment task -- alignment of an edge to a surface by accommodating rotationally around the y-axis -- was accomplished in the same manner as the **zguard** primitive with the exception that **zguard** was running during the training phase. This complicates the eigenvector extraction because we expect the **zguard** eigenvector to appear in the new set of trained vectors. In order to learn a “clean” version of the **yroll** primitive, the **zguard** must first be identified and removed from the set

FIGURE 2: AUTONOMOUS OPERATION OF THE LEARNED, ONE-DIMENSIONAL GUARDED MOVE.



of eigenvectors. This is possible as a direct result of our eigenvector representation of primitives. The vectors corresponding to primitives are unique in the sensor/actuator space so a simple test of parallelism (dot product) between the vectors of a candidate demonstration and the vectors of all possible primitives yields a similarity measure and mechanism for identification. (The interested reader is invited to consult [Voyles, Morrow, and Khosla, 1997] for additional detail.)

6. SYSTEM DEMONSTRATIONS

So far, our discussion has focused on the human-augmented adaptation of the expertise database itself. The motivation for having this database of primitives is to enable GBP, so now our attention turns toward the use of this database for such purposes. GBP is naturally applied in the real world, but there are complementary benefits to applying GBP in the virtual world, as well. We now discuss two variants of GBP for virtual demonstrations as well as physical demonstrations.

6.1Virtual Demonstrations

Coupled closely with the SPI, and an extension of it, is the Virtual Skill Constructor (VSC). The VSC is a set of utilities that integrates TeleGrip, a commercial solid modeling application for robotic workcells, Coriolis, a dynamic simulation package developed at CMU [Baraff 1995], and Chimera to provide a virtual environment for programming robotic skills by demonstration (Figure 11). Solid models of prototypical parts used in a particular instantiation of a robotic skill are manipulated in the virtual world displayed by TeleGrip. Coriolis computes the dynamics to generate realistic forces of interaction to simulate a real robot with real parts for the agents running in the Chimera real-time environment. The human first demonstrates the skill, taking the place of the robot in the virtual environment. A set of design agents observes the motions and suggests a prioritized list of sensorimotor primitives from the expertise database that may be appropriate for implementing the demonstrated subtasks of the skill (Figure 12). The demonstrator approves a set of primitives at each segment of the complete skill and the result is displayed graphically using the SPI. When the skill is complete, appropriate encapsulated expertise agents can be spawned on the real-time system and the skill executes in the virtual world with Coriolis supplying the dynamics of both the parts and the simulated robot. Finally, the virtual environment is replaced with a real robot and sensors and the developed skill is tested without modification.

The reason we are trying to combine both virtual demonstrations and real demonstrations is because they have different strengths. The real environment is good for extracting detailed contact information during fine manipulation without putting the operator in a force-feedback-deprived setting. (Force reflection and tactile feedback in virtual environments are still in their in-

fancy.) The virtual environment requires no calibration and can be used quickly for training or re-training with existing parts. It also requires placement of the viewpoint for proper interaction which provides information for visual servoing primitives that are difficult to extract from the real demonstrations.

6.2Physical Demonstration

We have implemented two different types of demonstrations in the physical realm. First is an explicit approach which is more closely related to robotic *instruction* by demonstration as opposed to *programming*. It is not programming because the executable is explicitly programmed in advance; the demonstrator just provides command instructions. The second system actually develops programs for simple contact-based tasks by physical demonstration.

6.2.1 Explicit Physical Demonstrations

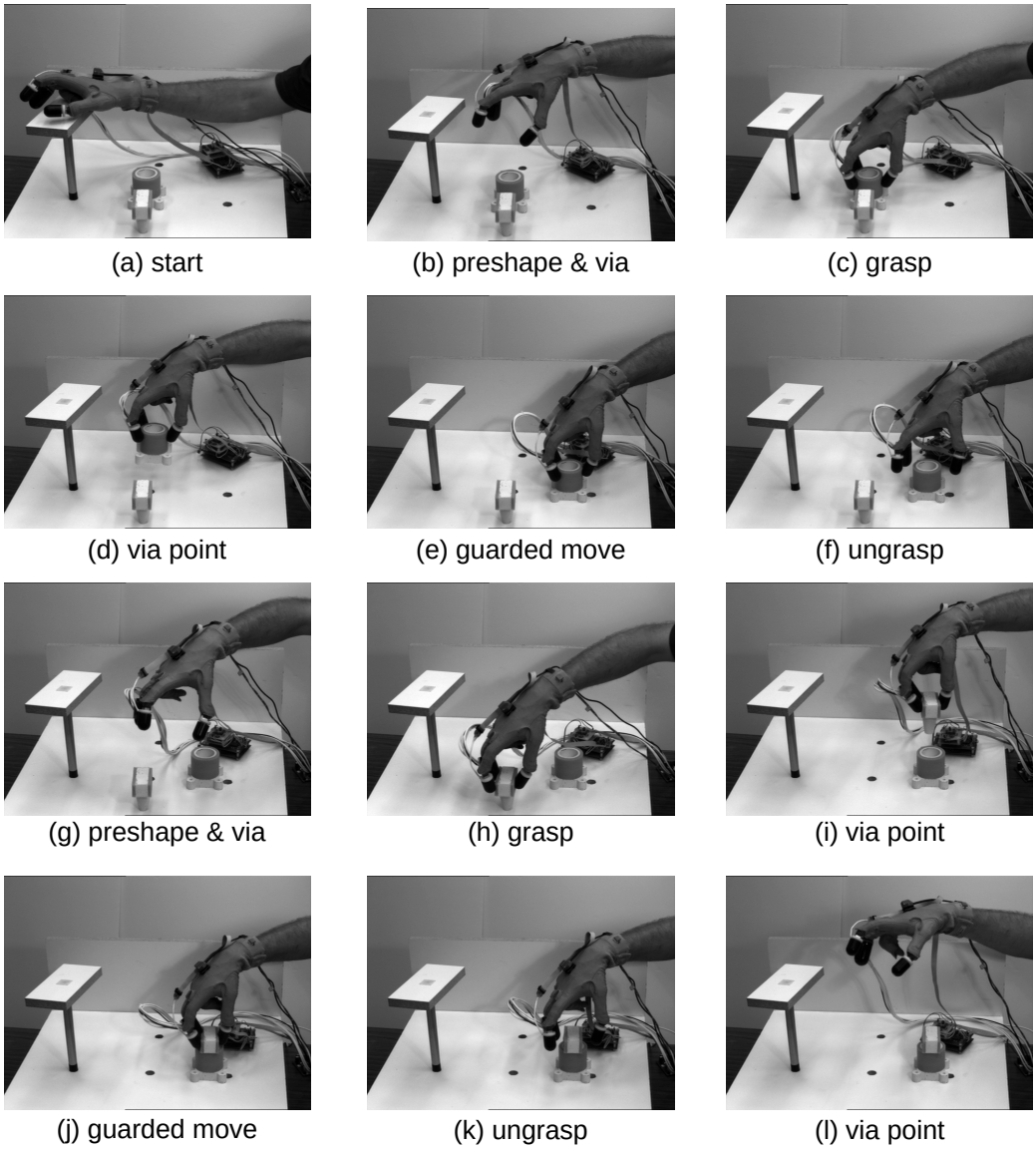
Using tactile and motion gesture recognition agents and kinematic execution agents, we developed a system for wire harness routing which was reported in [Voyles and Khosla 1995b]. The range of tactile gestures we were able to interpret for this implementation was quite limited due to the low resolution sensors we were using. (See [Voyles, Fedder and Khosla 1996] for novel sensor development.) For wire harness routing, the application involves pulling a wire through a bed of pegs such that groups of wires trace different paths through the pegs. Conceptually, the wires run straight through the harness but have different "pick-off" points to connect to various sensors and actuators along the length of the wiring harness.

To demonstrate the task, the user dons a CyberGlove with special pressure-sensitive tips (to detect grasping the wire) and physically demonstrates the path for each wire group by pulling a wire through the jig. A multi-agent network of gesture recognizers and interpreters deciphers the demonstration. The robot then executes the paths when instructed by symbolic user gestures in menu fashion.

6.2.2 Automatic Physical Demonstrations

Explicit physical demonstration is not GBP because the task is programmed *a priori*. GBP requires automatic generation of task programs, which is what we describe here. In the example below, a human demonstrates a low-tolerance peg-in-hole task in Figure 3. First, the hole is grasped, transported, and placed on the table. Then the peg is grasped, transported, and inserted into the hole. In both cases of placing the hole and the peg, a guarded move is used. In effect, the hole is pressed onto the table (as opposed to being dropped) and the peg is pressed into the hole. Contact force is used as the terminating condition rather than reaching a location in space.

FIGURE 3: DEMONSTRATION OF THE LOW-TOLERANCE PEG-IN-HOLE TASK.



(a) start (b) preshape & via (c) grasp

(d) via point (e) guarded move (f) ungrasp

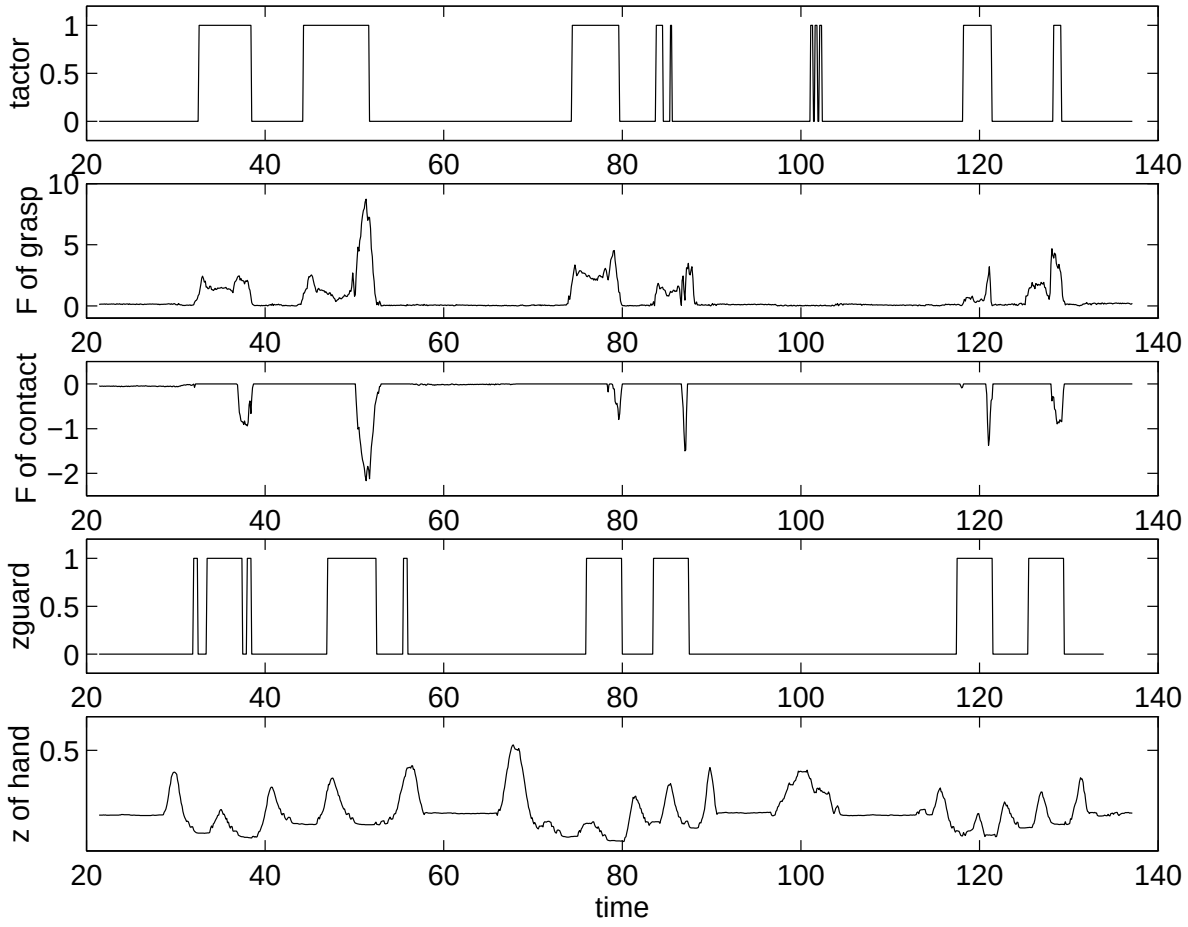
(g) preshape & via (h) grasp (i) via point

(j) guarded move (k) ungrasp (l) via point

To abstract the task a multi-agent network, based on the fine-grained tropism system cognitive architecture (Voyles et al 1997), interprets the demonstrator's "intention" during the various task phases.

Figure 4 illustrates some of the data from the abstraction process during three consecutive demonstrations of this task. The

FIGURE 4: IDENTIFICATION OF GUARDED MOVES IN THREE SEPARATE, CONSECUTIVE TASK DEMONSTRATIONS

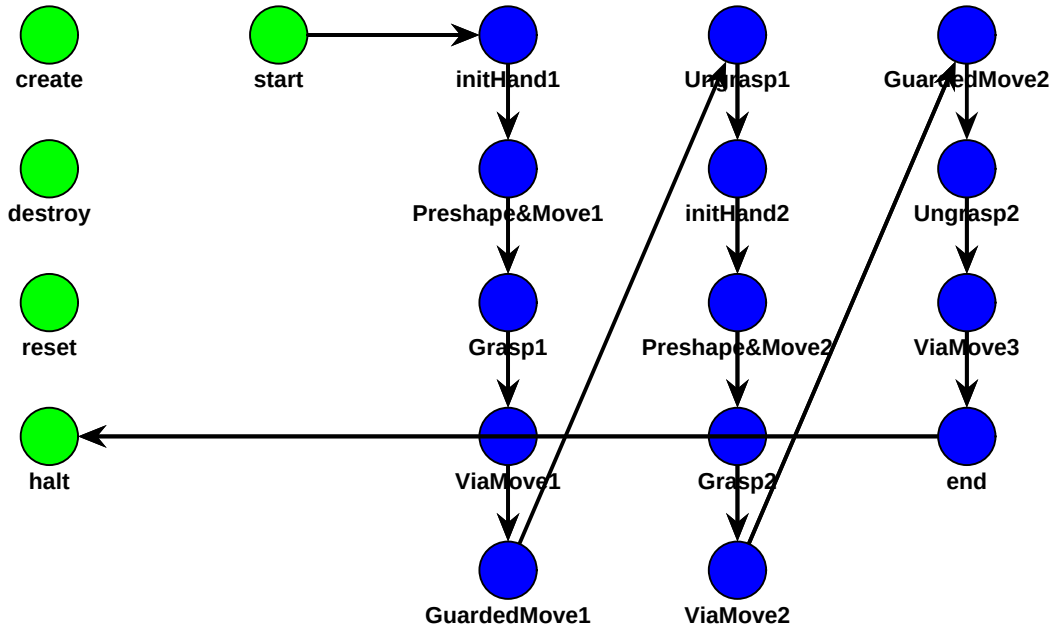


top plot is thresholded, binarized output of the extrinsic tactile sensor indicating the presence/absence of an object in the hand. The next plot shows the actual force of the grasp from the intrinsic tactile sensor. These agents aid the volume sweep rate agent [Kang and Ikeuchi, 1996] in temporally segmenting the gross phases of the task. The next plot shows the vertical component of the force vector from the intrinsic tactile sensor. This plot, along with the bottom plot which shows the vertical height of the hand are included only to help visually pinpoint when the guarded moves are being demonstrated (ground truth). The fourth plot from the top indicates the output of a gesture interpretation agent that identifies the presence of the **zguard** guarded move primitive described in Section 5.3.

Note the “blips” in the data around the time equal to 100 seconds. These resulted from extraneous motion to untwist some wires around the demonstrator’s wrist and were appropriately rejected by the volume sweep rate agent and tactile agents as not being recognizable pregrasp and manipulation phases of the task.

The program that resulted from the physical demonstration is displayed in Figure 5 as a finite state machine using the SPI. Each bubble in the state machine represents a node that consists of several agents executing simultaneously to achieve the desired action. The autonomous execution of the above program on a PUMA robot with a Utah/MIT hand is shown in Figure 6.

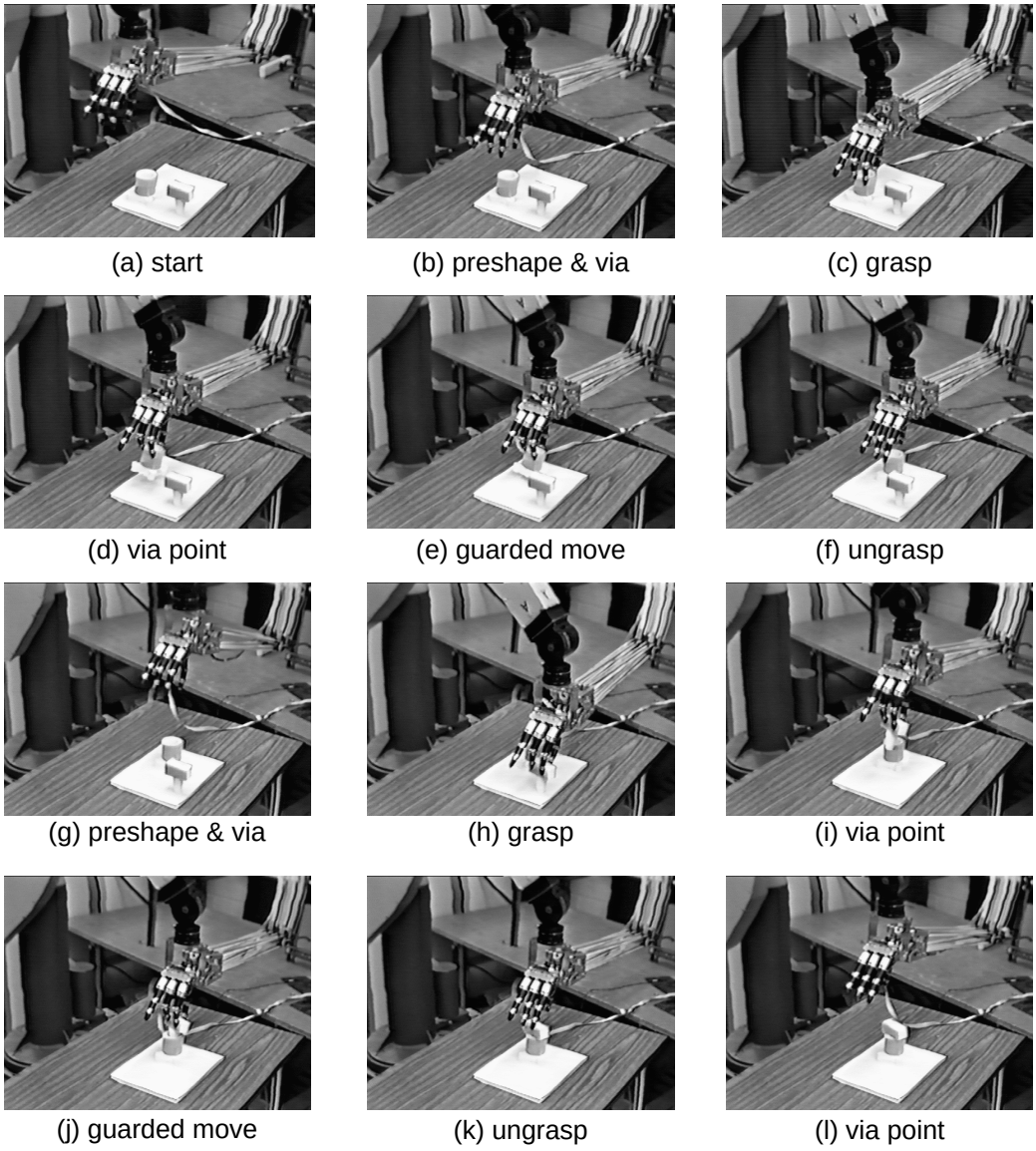
FIGURE 5: SPI GRAPHICAL DISPLAY OF PROGRAM RESULTING FROM PEG-IN-HOLE DEMONSTRATION.



7. DISCUSSION

We have presented our concept for a gesture-based programming system for robotic applications as well as portrayals of the work we have completed on individual components of that system. By employing human-augmented software adaptation, which employs the complementary strengths of both man and machine, the program development system evolves along with applications to provide a more powerful and more intuitive environment as time goes on. This is the GBP paradigm, which is a valuable step toward fully self-adaptive software.

FIGURE 6: ROBOTIC EXECUTION OF THE LOW-TOLERANCE PEG-IN-HOLE TASK.



Specifically, the software is adapted by the acquisition of sensorimotor primitives that provide targeted functionality for achieving task objectives. Once acquired, these primitives reside in a knowledge base and become part of the development environment, evolving the development environment over time. We have demonstrated primitive acquisition by explicit programming and by learning. In the case of learning, primitive identifiers are automatically generated that can help identify subsequent

instances of the primitives during human demonstrations of other tasks. We have also demonstrated the recomposition of these primitives both explicitly and automatically by human demonstration to complete meaningful, though simplified, operations.

In a nutshell, gesture-based programming attempts to take advantage of the wealth of human experience we all gain through our day-to-day physical interactions to specify complex tasks rather than relying on translations to text. In a loose sense, gesturing involves "acting out" a scenario so the software and the system can adapt itself to emulate it. But, because gestures are not necessarily explicit representations of actions, they can also be used on-line to tune the robot's actions and improve its performance as it executes the task after initial teaching. Therefore, gestures can augment both the programming and the commanding of the system.

However, there is still much theoretical and practical work to be done to enable the system to be useful for complex tasks in real-world environments. One of the biggest unsolved problems with skill-based paradigms in general is being able to predict what tasks are spanned by a given set of primitives. Likewise, the inverse problem of predicting what set of primitives is necessary to accomplish a range of tasks is unsolved. Furthermore, the idea of demonstration followed by supervised practice requires the automatic construction of monitoring agents in conjunction with the learning of execution agents. This, too, is an unsolved problem. Finally, there is much practical work in learning more complicated and non-linear primitives. Fortunately, the adaptation of highly non-linear systems to linear representations is something with which engineers have a great deal of experience.

8. REFERENCES

[1] Agah, A. and G.A. Bekey, "Phylogenetic and Ontogenetic Learning in a Colony of Interacting Robots," *Autonomous Robots*, v. 4, 1997, pp. 85-100.

[2] D. Baraff, "Interactive Simulation of Solid Rigid Bodies." in *IEEE Computer Graphics and Applications*, v. 15, n.3, pp 63-75, 1995.

[3] Brooks, R.A., "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, v.RA-2, n.1, March 1986, pp. 14-23.

[4] Gertz, M.W., D.B. Stewart and P.K. Khosla, "A Software Architecture-Based Human-Machine Interface for Reconfigurable Sensor-Based Control Systems," in *Proc. of the 8th IEEE Symp. on Intelligent Control*, Chicago, IL, August 1993.

[5] Hancock, J. and C. Thorpe, 1995, "ELVIS: Eigenvectors for Land Vehicle Image System," in *Proc. of 1995 IEEE/RSJ International Conf. on Intelligent Robots and Systems*," Pittsburgh, PA, Aug., pp 35-40.

[6] Integrated Systems, Inc. web site, www.isi.com

[7] Kang, S.B., and K. Ikeuchi, "Toward Automatic Robot Instruction from Perception -- Temporal Segmentation of Tasks from Human Hand Motion," in *IEEE Transactions on Robotics and Automation*, v., n. Mar. 1996.

[8] Kuniyoshi, T., M. Inaba, and H. Inoue, "Learning by Watching: Extracting Reusable Task Knowledge from Visual Observation of Human Performance'' *IEEE Transaction on Robotics and Automation*, vol.10, no.6, pp.799-822, Dec., 1994

[9] Morrow, J.D. and P.K. Khosla, "Sensorimotor Primitives for Robotic Assembly Skills," in *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, Nagoya, Japan, May, 1995.

[10]Morrow, J.D., B.J. Nelson, and P.K. Khosla, "Vision and Force Driven Sensorimotor Primitives for Robotic Assembly Skills," in Proceedings of 1995 IEEE/RSJ Int. Conf. on Intelligent Robots and Sys., Pittsburgh, August 5-9, 1995.

[11]National Instruments web site, www.natinst.com
[12]Patrick, J., *Training: Research and Practice*, Academic Press, San Diego, CA, 1992.
[13]Pierce, D., 1991, “Learning a Set of Primitive Actions with an Uninterpreted Sensorimotor Apparatus,” 8th International Workshop on Machine Learning, Evanston, IL, pp 338 - 342.
[14]Real-Time Innovations web site, www.rti.com
[15]Stewart, D.B., D.E. Schmitz, and P.K. Khosla, “The Chimera II Real-Time Operating System for Advanced Sensor-Based Control Applications,” in *IEEE Transactions on Systems, Man and Cybernetics*, v. 22, n.6, Nov./Dec. 1992, pp. 1282-1295.
[16]Stewart, D.B. and P.K. Khosla, “The Chimera Methodology: Designing Dynamically Reconfigurable and Reusable Real-Time Software Using Port-Based Objects,” *International Journal of Software Engineering and Knowledge Engineering*, v.6, n.2, June, 1996, pp. 249-277.
[17]Todd, D.J., Fundamentals of Robot Technology, John Wiley and Sons, 1986, chapters 3, 7.
[18]Tomasi, C. and T. Kanade, “Shape and Motion from Image Streams Under Orthography---A Factorization Method,” *International Journal on Computer Vision*, 9(2):137-154, November 1992
[19]Tomovic, R., “Transfer of Motor Skills to Machines,” in *Robotics and Computer-Integrated Manufacturing*, v. 5, n. 2/3, 1989, pp. 261-267.
[20]R.M. Voyles, Jr. and P.K. Khosla, "Tactile Gestures for Human/Robot Interaction," in *Proceedings of 1995 IEEE Conf. on Intelligent Robots and Systems*, Pittsburgh, PA, Aug. 1995a.
[21]R.M. Voyles, Jr. and P.K. Khosla, "Multi-Agent Gesture Interpretation for Robotic Cable Harnessing," in *Proceedings of 1995 IEEE Conf. on Systems, Man, and Cybernetics*, Vancouver, B.C., Oct. 1995.
[22]Voyles, R.M., J.D. Morrow, and P.K. Khosla, "Towards Gesture-Based Programming: Shape from Motion Primordial Learning of Sensorimotor Primitives," in *Robotics and Autonomous Systems*, v. 22, pp. 361-375, 1997.
[23]Voyles, R.M., A. Agah, P.K. Khosla, and G.A. Bekey, 1997, “ Tropism-Based Cognition for the Interpretation of Context-Dependent Gestures,” in *Proceedings of 1997 IEEE International Conference on Robotics and Automation*, Albuquerque, NM, v. 4, pp. 3481-3486, Apr.
[24]R.M. Voyles, Jr., G. Fedder and P.K. Khosla, “ A Modular Tactile Sensor and Actuator Based on an Electrorheological Gel,” in *Proceedings of 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, MN, v. 1, Apr. 1996, pp 13-17.
[25]Wooldridge, M. and N.R. Jennings, “Intelligent Agents: Theory and Practice,” *Knowledge Engineering Review*, v. 10, n. 2, June 1995, pp. 115 - 152.
[26]Kaiser, M. and Dillman, R., “Building elementary robot skills from human demonstration,” in Proceedings of the 1996 IEEE International Conference on Robotics and Automation. v. 3, 1996. pp. 2700-2705.

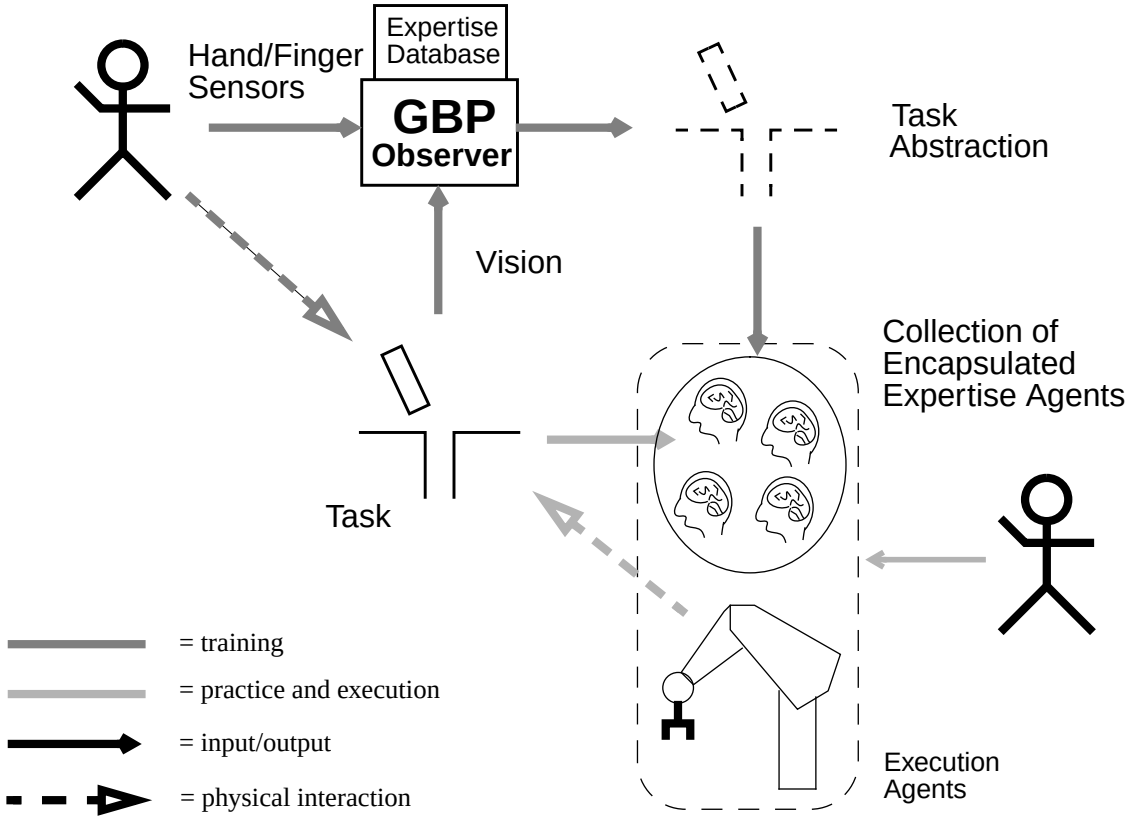


FIGURE 7: GESTURE-BASED PROGRAMMING (GBP) SYSTEM OVERVIEW

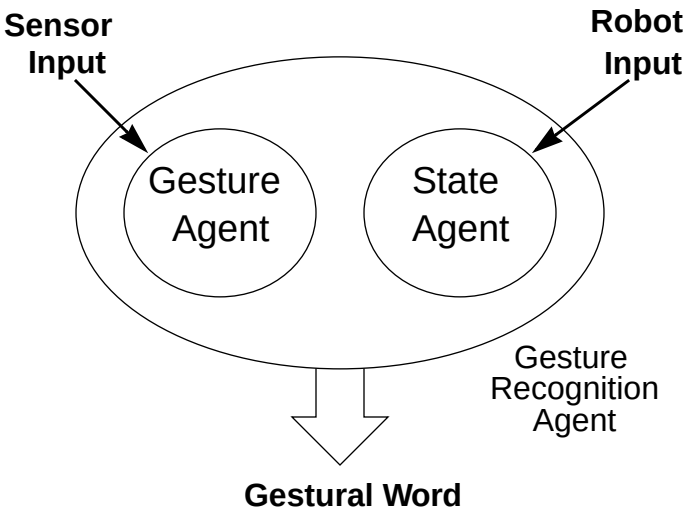


FIGURE 8: STRUCTURE OF THE GESTURE RECOGNITION AGENT

.

i

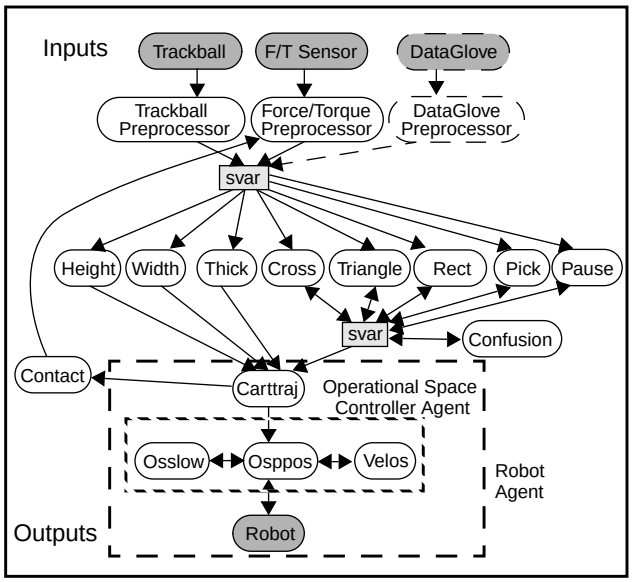


FIGURE 9: THE AGENT HIERARCHY.

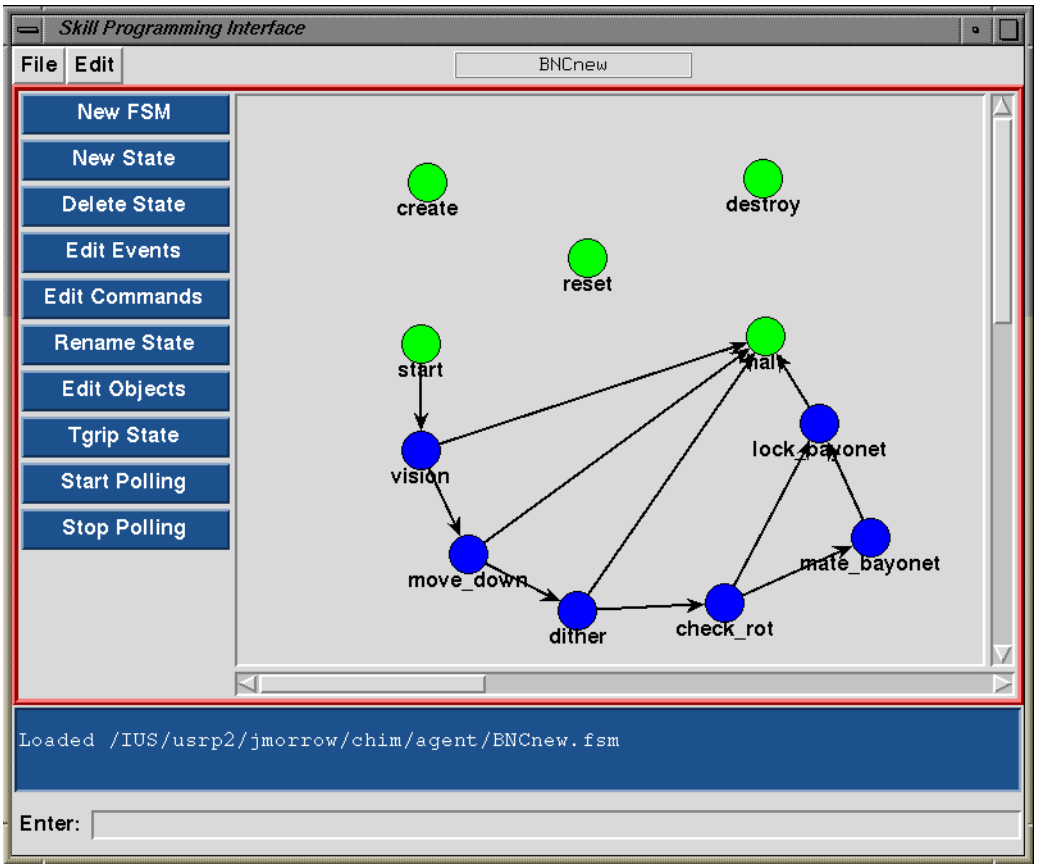


FIGURE 10: SPI FINITE STATE MACHINE EDITOR BUILDING A BNC CONNECTOR INSERTION SKILL

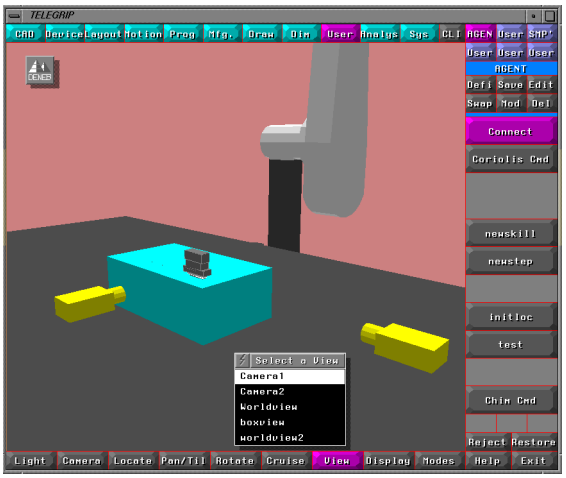


FIGURE 11: TELEGRIP WINDOW OF VIRTUAL SKILL DEMONSTRATION ENVIRONMENT WITH D-CONNECTOR

FIGURE 12: SENSORIMOTOR PRIMITIVE SELECTION DURING ONE STEP IN THE VIRTUAL DEMONSTRATION OF A D-SHELL CONNECTOR INSERTION