In Proceedings of the 1995 IEEE Conf. on Systems, Man,
and Cybernetics, Vancouver, BC, Oct. 1995, vol. 2.

Multi-Agent Gesture Interpretation for Robotic Cable Harnessing

Richard M. Voyles, Jr.† Pradeep K. Khosla‡

†Robotics Ph.D. Program
‡Dept. of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Gesture-Based Programming is our paradigm to ease the burden of programming robots. It is an extension of the human demonstration approach that includes encapsulated expertise to guide subtask segmentation and robust real-time execution. A variety of human gestures must be recognized to provide a useful and intuitive interface for the human demonstrator. While the full gesture-based programming environment has not yet been realized, this paper describes a multi-modal gesture recognition system that embodies many of the necessary elements to achieve true gesture-based programming. It begins with recognition of the gestures of a human demonstrating a trajectory. The execution agents then try to repeat the trajectory while observing corrective gestures from the teacher. Similar multi-agent networks are used for both training and execution.

1. Gesture-Based Programming

Programming and interacting with robots can be difficult even for experts, but is downright intimidating for most casual users. Numerous attempts have been made to ease the discomfort of human/robot interaction with varying degrees of success. Behavior-based and multi-agent systems [1],[2] seek to modularize software for easier programming by experts. Visual programming environments like Chimera/ Onika [4][14] and commercial packages such as MatrixX/ SystemBuild and LabView 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. For convenience, they also provide point-and-click interaction during run-time.

Despite these advances, a fairly high level of expertise is still required to program and interact with robots. For example, Onika allows novice users to easily build simple pick-and-place applications [3]. But, it is imperative that the user understands the importance of via points, collision avoidance, grip selection, and the dynamic effects of transport in order to for the application to be successful. And all this is required for a simple pick-and-place example! Visual programming does little for the bulk of robotic applications that involve substantive contact with the environment, uncertainty, and complex sensing.

A more recent approach to human/robot interaction is the field of *learning by observation* [7],[8]. By forcing the robot to observe a human interacting with the world, rather than forcing the human to interact with a *representation* of the robot interacting with the world, a more natural, rich, "anthropocentric" environment results. But these systems are mostly kinematically-based and operate off-line; if the robot misinterprets the desired trajectory, the whole sequence must be re-taught.

The ultimate goal of our research is the creation of a gesture-based programming and interaction environment for robotic systems. This is an extension of the learning by observation paradigm that seeks to understand the *intentions* of a human demonstrator by observing physical gestures. We are primarily concerned with gestures of the hands and fingers that involve touch (tactile gestures) and free movement (motion gestures). We are attempting to develop a context-sensitive grammar of gestures that will allow the configuration and parametrization of software modules off-line (as a programming system), and also allow intuitive run-time interaction for fine tuning the program and providing on-line command input.

The software modules embody what we call *encapsulated expertise*, self-contained sensorimotor capabilities that can achieve elemental subtasks given appropriate parameters [9][11]. For example, when human trainees are learning an assembly operation involving a screw insertion, they already know how to grasp the screw and how to thread it home. What they are learning through the training process is where the screws go, what they fasten, and in what order they should be assembled. It is against this knowledge base of capabilities that subtask segmentation and information extraction is performed.

Unfortunately, we have not yet succeeded in constructing a full gesture-based environment. Many enabling capabilities are, as yet, unrealized such as identifying fundamental gestures, interpreting gestures from sensory input and abstracting from gestures to intentions. This paper describes a specific application in which tactile and motion gestures, observed with a

sensorized glove, are combined and used to program the trajectories and to command run-time operation.

2. The Cable Harnessing Task

Cable harnessing involves the bundling of individual wires into a wire "rope" with appropriate *taps*[1] to connect to various electrical accessories along its path. The wires are normally routed around a jig that maintains the appropriate spacing between taps. For our application, we use a reconfigurable jig consisting of a set of moveable pegs around which the wires are strung (Figure 1). Because the resulting harness is flexible and the robot's workspace is limited, we will assume all wires can be laid out in a single plane with the end point near the start point to keep cycle times low. We also make the more restrictive assumption that all wires in a harness follow one of a small number of paths: straight through or a tapped pattern.

Since the wires are strung point-to-point around pegs, their shapes are polygonal and the task is strictly one of tracking a trajectory. Using a DataGlove, a Polhemus sensor and a contact sensor on the index finger, the user grasps a wire and traces out the trajectory the robot must follow for each wire type. This is part of the teaching phase. To fine-tune the trajectory, simple "dictionary" hand gestures start and stop the robot's movement through the cyclic trajectory. As it's moving, tactile gestures nudge the robot into the precise trajectory. (Precise has a loose interpretation since it merely has to avoid collisions with the pegs.)

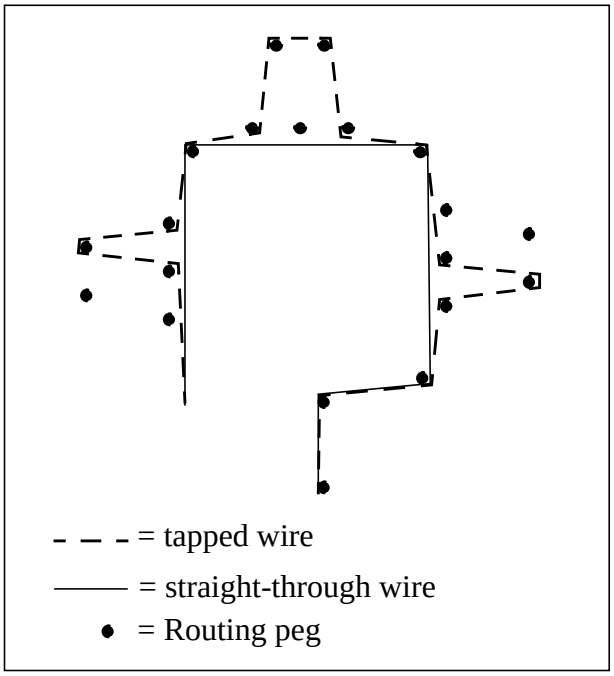

Figure 1: Wire Harness Jig

1. A "tap" is a small loop for connecting to a device.

Each harness contains a variable number of each type of wire and the robot runs the same type of wire until instructed to change. Changing from one type of wire to another is accomplished in a menu-style with symbolic hand gestures. These gestures are interpreted by a multi-agent network of "gesture intention recognizers," each of which determines the confidence with which a certain gesture matches the recognizer's internal model of the user's intentions. They continuously vote with the strength of their confidence to see if an action (i.e. "change the trajectory") is warranted.

3. Gestures

Our definition of a gesture is "an imprecise, context-dependent event that conveys the user's intentions." There are many gestural modalities including *symbolic gestures*, as in sign language, *motion gestures*, such as motioning to approach, *tactile gestures*, such as nudging to get attention, and *articulatory gestures*, which are audible sounds and words. Since the definition is rather broad, other gestural modes exist but this is the subset in which our primary research interests lie for gesture-based programming.

Gestures are context dependent largely because of their impreciseness; in isolation they convey little information. Yet, when combined with the state of the system and the environment at the time the gesture occurred, perhaps augmented with some past history of gestures, the intention becomes interpretable.

Gestural Alphabet

Using a linguistic analogy, the raw gestures and state information form a *gestural alphabet*. The gestural alphabet is made up of characters representing the different gestural modes and the different types of relevant state information. A *gestural word* is assembled from the raw gesture and its associated context by a gesture recognizer. Gesture interpretation examines these words within *gestural sentences* that are strung together by the gesturer.

The gestural alphabet is like a linguistic alphabet in the sense that characters are parametrized. For instance, in English, the letter "a" has many sounds including the long "a" as in "save," the short "a" as in "sad," and the silent "a" as in "read." This is an implicit parametrization. In contrast, the gestural characters are explicitly parametrized by either a single integer or floating-point argument.

For example, consider a "nudge" -- an end-of-arm tactile gesture. Let's call it character "**N**" for nudge. (The alphabetic labels are arbitrary.) **N** is discretely parametrized into one of three types: brief, elongated or continuation; so **N** takes an integer parameter. The reason for these three types of nudges is to allow for gestures that are drawn out in time. This notation permits the recognition of the start of a gesture in progress.

The raw gesture is just a force impulse so it has some average magnitude and direction associated with it as well as its duration. The magnitude and direction can be represented with force components so, for planar forces, we add the alphabetic characters **X** and **Y** with their floating point parametric values. For a particular gestural event the raw gesture becomes "**N***n* **X***x* **Y***y*," where *n*, *x* and *y* are the instantiated values associated with the event.

To form a complete gestural word, state information must be added to facilitate interpretation. Imagine a fictitious application that requires only the cartesian position of the manipulator. We can call the $\Re^3$ components of the robot's cartesian position **P**, **Q** and **R**, which would have floating point parametric values, also. The resulting application-specific gestural word would be "**N***n* **X***x* **Y***y* **P***p* **Q***q* **R***r*."

Gesture Recognizers

Because gestures are context dependent, state information must be associated with each gesture. This state information is generally application specific. We modularize the gesture recognition agents by separating the application-specific state specification agent from the raw gesture agent.

Gesture recognition is handled by individual gesture recognition agents for each gestural mode. Each recognition agent is made up of two subordinate agents, one general, one domain-specific. The general agent is the

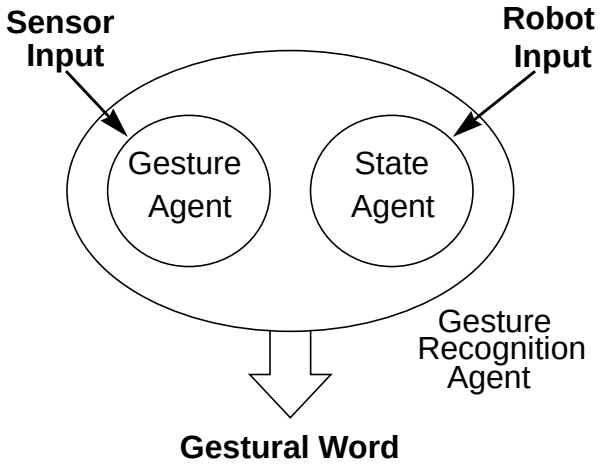

Figure 2: Structure of the Gesture Recognition Agent

gesture agent. It recognizes the instance of a gesture and the type of gesture within the gesture class (mode). The domain specific agent grabs the relevant state information associated with the gesture type for the given domain and appends it to the gestural word.

Tactile Gestures

There are two types of tactile gestures we consider for this application: end-of-arm gestures and pinching gestures. End-of-arm gestures involve the user physically interacting with the robot at the robot's end effector and are sensed by a wrist force/torque sensor. Pinching gestures are simply pinching actions of the demonstrator's fingers and are sensed by an instrumented glove.

End-of-arm tactile gestures are used to select and fine-tune the robot's polygonal trajectories. Much of the work on end-of-arm tactile gestures that is used for this research has been reported in [15]. To summarize, end-of-arm tactile gestures are force impulses that have associated with them state information that includes:

- average force components
- force magnitude
- parallelism between force and the X-axis
- parallelism between force and robot velocity
- parallelism between force and radial vector
- quadrant location of gesture with respect to centroid
- closeness to a vertex of the polygon

Except for the force magnitude variables, the state information is represented as fuzzy variables. For example, parallelism between the force and the robot's velocity has a membership function defined by the dot product. This state value is used to determine if the gesture is aiding the motion (parallel), opposing it (anti-parallel) or trying to alter its direction (perpendicular).

Pinching gestures are also tactile gestures because they involve touch, but they are quite different from end-of-arm gestures. For this application, we are only sensing pinch between the index finger and thumb to determine start and stop points for trajectories during training. Although we maintain a fuzzy membership value based on the strength of the pinch above a threshold, the pinch gesture is used as a binary flag for this application.

Symbolic Gestures

Symbolic gestures are sign language gestures. We use symbolic gestures to create a menu of trajectories because they are the most specific form of gesticulation. They result in the least amount of interpretation time and so respond much quicker for the run-time phase of operation.

Figure 3: Symbolic hand gestures

Hand Motion Gestures

Hand motion gestures are much more varied than tactile gestures and are more difficult to classify. They can be deliberative, such as tracking a curve to indicate a desired trajectory, or flippant, as in motioning a little farther along. Deliberative gestures are easier to recognize

and are better suited to training, so we focus on them for this application.

We consider two types of hand motion gestures: straight-line motions and rotations in place. These two types of gestures provide redundant information on the segments of the trajectories and their boundaries during training. Redundancy is important to achieve high interpretation accuracy because the recognition accuracy of individual gestures is low by comparison.

4. Encapsulated Expertise

For this application, the encapsulated expertise is fairly trivial. Basically, the robot can track a trajectory and it can grip a wire with a two-fingered gripper. No complex sensorimotor dexterity is required for cable harnessing so the subtask segmentation problem is limited to selecting how to generate and track the trajectories during run-time.

5. Multi-Agent Networks

Training, practice, and execution are all performed in real-time by networks of independent agents. We first describe the agent networks from a systems perspective and then describe the agents in more detail in section 6.

Training

Training the wire runs is the first step in the application. The trainer dons a DataGlove instrumented with a force sensitive resistor (FSR) on the index finger and trains each wire trajectory by grasping a Polhemus sensor between the fingers and tracing it through the route the wire takes around the jig. The act of grasping the Polhemus simulates grasping the wire (for a two-state pneumatic gripper) and signals the starting point of the trajectory. Ungrasping signals the end. Although trivial in this application, these pinching gestures are recognized by the recognition agent, pinch (see Figure 4) which monitors the FSR.

Each trajectory is assumed to be made up of a series of line segments (polygonal). Two additional recognition agents, rotate and straight, attempt to identify straight-line and rotating gross motion gestures of the hand. These agents operate independently so it is possible for straight-line and rotating gestures to be recognized simultaneously.

The interpretation of the gestures, identifying whether any given motion is part of a trajectory segment, an inter-segment reorientation, or is extraneous, is performed by the interpretation agent, segment. Segment forms the fuzzy result of `not_straight AND rotating AND grasped_wire`. If this is true, the segment is stored in a file for later recovery by the execution agents. Segment keeps track of the start and stop of each trajectory and files them separately.

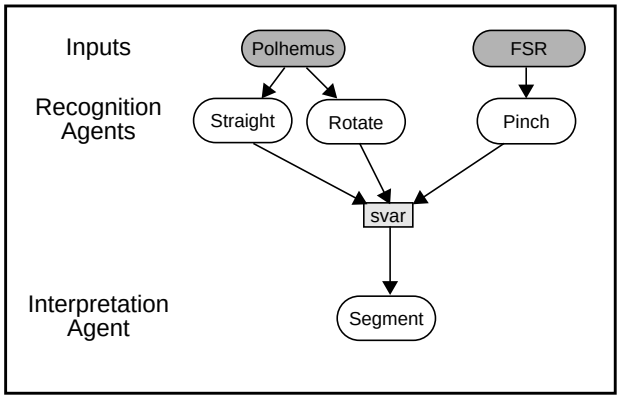

Figure 4: The training network.

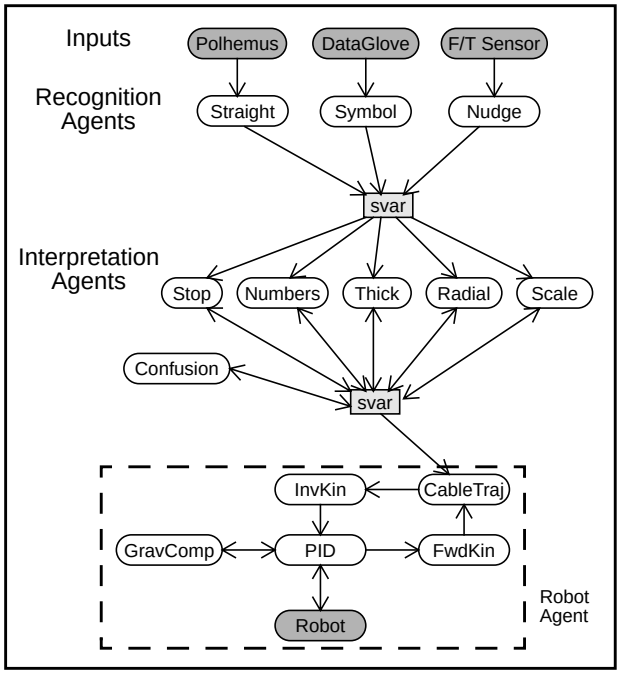

Figure 5: The practice network.

To provide a small level of immunity to the noisy output of the Polhemus, segment takes a parameter that specifies the minimum step-size of the grid of pegs. It then rounds the positions contained in the gestural word up or down if the rotation was to the left or right, respectively (assuming a counter-clockwise motion).

Practice

The practice session is when the robot moves through the various trajectories and the trainer observes the motions to see if they are as desired. End-of-arm tactile gestures similar to those we reported in our previous work [15], are used to fine-tune the trajectory while symbolic hand gestures coupled with hand motion gestures are used to switch between trajectories. These are illustrated in Figure 4.

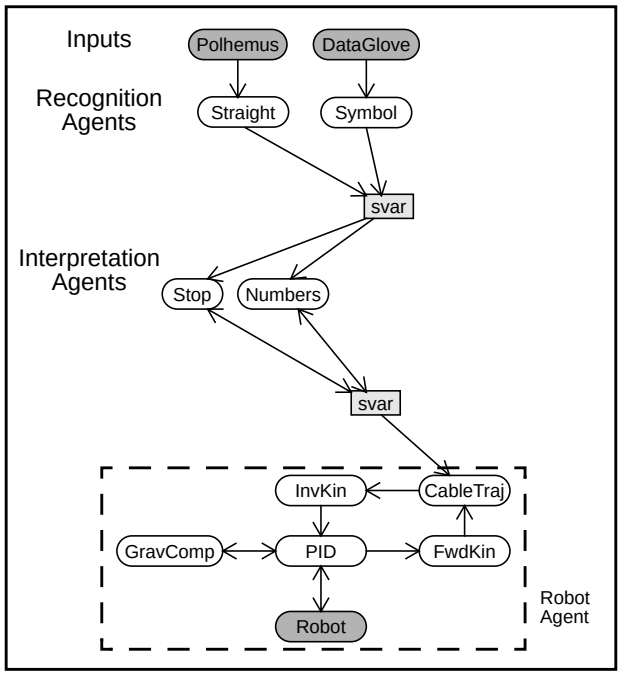

Figure 6: The application network.

Execution

The execution phase is fairly trivial because the trajectories are all prepared, we just want to provide a simple menu to select between them. This menu is implemented as a set of symbolic gestures of the raised hand indicating the trajectory number by the number of vertically extended fingers.

Interpretation Mechanism

All agents are implemented as port-based objects [14] under the Chimera reconfigurable subsystem server. They use Chimera's state variable mechanism as a blackboard for status, interagent communication, and voting. The interpretation agents receive gestural words from the recognition agents that they compare with their internal fuzzy models. These models determine whether the gesture strengthens, weakens,. or is irrelevant to the agent's hypothesis. The interpretation agents subscribe to voting pools within which they vote with the strength of their confidence. Since there are several gestural modalities under simultaneous observation, the voting pools are formed around competing gestures. This way, non-competing gesture interpretations from separate pools can all "win."

Agent Descriptions

The network topology, voting mechanism and end-of-arm tactile gesture recognizers are described in detail in [15][16].

Confusion: Confusion acts as a virtual sensor by determining the "confusion" within the pools of interpretation agents. It examines the value and rate of change of the most confident agents to estimate the indecision of the voting pool as a whole.

In the ultimate gesture-based programming environment, we want the networks to be self-modifying (evolutionary). For this type of programming paradigm, it will be necessary for the network to have a proprioceptive sense of its competence so it can modify its own topology to better achieve its goals. The confusion agent provides one candidate measure of competence.

Numbers: This is actually a wild-card in the topology that symbolizes a group of agents for recognizing numeric sign language gestures. Each number agent stores a template of a symbolic gesture and computes a closeness of fit. Since static pattern matching for recognizing sign language is non-robust, the gesture recognizer is fairly liberal while the interpreter is biased to look closely at negative stimuli such as hand motion gestures and end-of-arm tactile gestures. (It is assumed that symbolic gestures are usually made close to the body, in a roughly vertical position, with the hand approximately still.)

6. Experiments

The most important step in operating the system is training the trajectories. Figures 6 and 7 indicate two

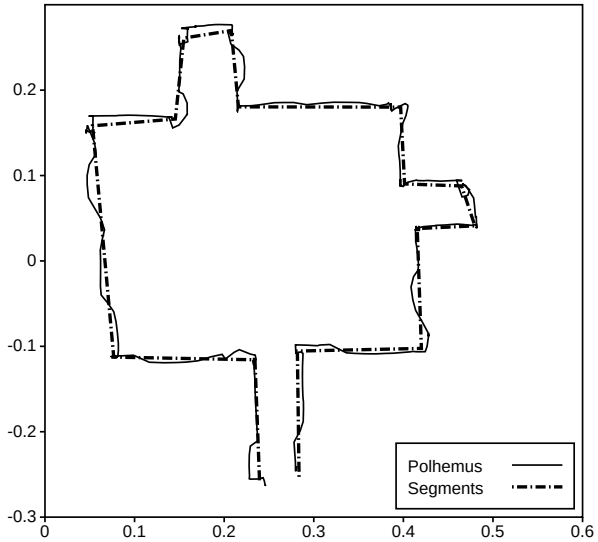

Figure 7: Segmented and raw trajectories from training.

segmented trajectories and the raw Polhemus data that resulted from the training phase. Noise in the Polhemus data can be significant and it is particularly disturbed by ferromagnetic materials. After the practice session (Figure 9), however, it squares up fairly well. Of course, the only criteria for a good trajectory is that it doesn't hit any of the pegs of the jig.

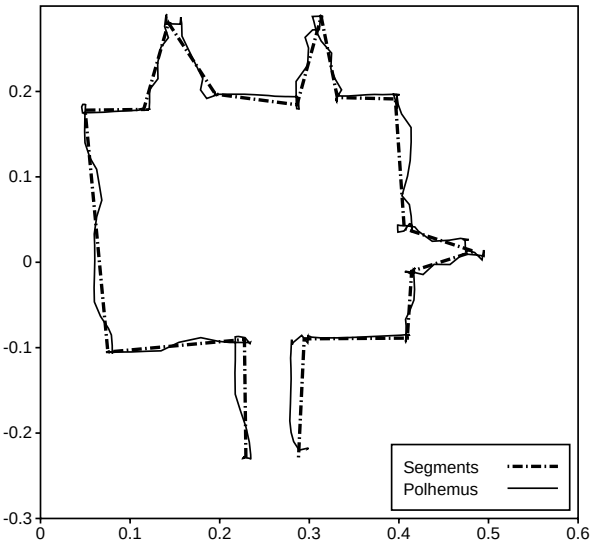

Figure 8: Another segmented trajectory and raw trajectory from training.

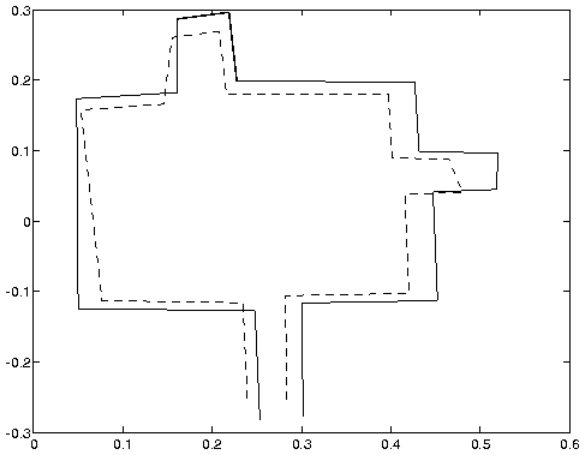

Figure 9: Trained trajectory (dotted) and practice trajectory (solid)

7. Conclusions

Although gestural input may not be the most efficient method of commanding a cable harnessing robot, it seems to work satisfactorily. For this particular task, a teach pendant would work at least as well and probably much better. However, teach pendants have been around for a long time and they have little hope of making robots any more intuitive to program for complex tasks. To the contrary, our gestural framework has the potential for tremendous impact if we can achieve all our goals.

In the near term, the biggest problem we have is in recognizing symbolic gestures. Most other gestural modes have redundancy built-in, while sign language gestures tend to be one-shot events. The other gestures, by design, require only 60 - 80% accuracy in recognition because interpretation waits for corroborating gestures. If one or two get lost, the impact is minimal. With symbolic gestures, that level of recognition is irritating to the user.

8. References

[1] 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.

[2] Connell, J.H., "A Colony Architecture for an Artificial Creature," MIT AI Technical Report no. 1151, 1989.

[3] Gertz, M.W, Ph.D. Thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1994.

[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] Hirai, S. and T. Sato, "Motion Understanding for World Model Management of Telerobot," in *Proc. of the 5th International Symp. on Robotics Research*, 1989, pp. 5-12.

[6] Hoffman, D.D. and W.A. Richards, "Parts of Recognition," *Cognition*, v. 18, pp. 65-96, 1984.

[7] Kang, S.B. and K. Ikeuchi, "Grasp Recognition and Manipulative Motion Characterization from Human Hand Motion Sequences," in *Proc. of the 1994 IEEE International Conf. on Robotics and Automation*, v. 2, May, 1994, pp. 1759-1764.

[8] Kuniyoshi, T., M. Inaba, and H. Inoue, "Teaching by Showing: Generating Robot Programs by Visual Observation of Human Performance," in *Proc. of the 20th International Symp. on Industrial Robots*, 1989, pp. 119-126.

[9] Morrow, J.D. and P.K. Khosla, "Sensorimotor Primitives for Robotic Assembly Skills," to appear in *Proc. of the 1995 IEEE Int. Conf. on Robotics and Automation*, May, 1995.

[10] Rubin, J.M. and W.A. Richards, "Boundaries of Visual Motion," MIT AI Lab tech. report 835, April, 1985.

[11] Schneider, J.G. and C.M. Brown, "Robot Skill Learning, Basis Functions, and Control Regimes," in *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, v. 1, May, 1993, pp. 403-410.

[12] Shepp, B.E. and S. Ballesteros, editors, *Object Perception: Structure and Process*, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1989, chapter 5.

[13] Stewart, D.B., D.E. Schmitz and P.K. Khosla, "The Chimera II Real-Time Operating System for Advanced Sensor-Based Robotic Applications," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1282-1295, Nov./Dec. 1992.

[14] Stewart, D.B., R.A. Volpe, and P.K. Khosla, "Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects," CMU Robotics Institute tech. report, CMU-RI-TR-93-11, July, 1993.

[15] Voyles, R.M. and P.K. Khosla, "Tactile Gestures for Human/Robot Interaction," to appear in *Proceedings of the IEEE Intelligent Robots and Systems Conf.*, Aug. 5-9, 1995, Pittsburgh, PA.

[16] Voyles, R.M. and P.K. Khosla, "Multi-Agent Perception for Human/Robot Interaction: A Framework for Intuitive Trajectory Modification," CMU Robotics Institute tech. report, CMU-RI-TR-94-33, September, 1994.