In Proceedings of IEEE/RSJ Intelligent Robots and
Systems Conference, August 5-9, 1995, Pittsburgh,
PA, Volume 3

Tactile Gestures for Human/Robot Interaction

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 a new paradigm to ease the burden of programming robots. By tapping in to the user's wealth of experience with contact transitions, compliance, uncertainty and operations sequencing, we hope to provide a more intuitive programming environment for complex, real-world tasks based on the expressiveness of non-verbal communication. A requirement for this to be accomplished is the ability to interpret gestures to infer the intentions behind them. As a first step toward this goal, this paper presents an application of distributed perception for inferring a user's intentions by observing tactile gestures. These gestures consist of sparse, inexact, physical "nudges" applied to the robot's end effector for the purpose of modifying its trajectory in free space. A set of independent agents - each with its own local, fuzzified, heuristic model of a particular trajectory parameter - observes data from a wrist force/torque sensor to evaluate the gestures. The agents then independently determine the confidence of their respective findings and distributed arbitration resolves the interpretation through voting.

1 Gesture-based programming

The three most common industrial applications of robots are spray painting, welding, and material handling, which together account for the vast majority of installations [15]. These applications have been successful for two reasons: they're well-suited to robotic capability and they are easily programmed. They're well-suited because they are all essentially kinematic tasks and programming is easy because they are "taught" what to do.

From a robocentric perspective, spray painting is the most challenging task because the quality level of the job is impacted by the robot's execution. (Spot welding and loading, by comparison, have "nearly" binary quality functions with respect to robot execution: good or bad.) As such, spray painting requires the most *skill transfer* from the teacher (i.e., programmer) to the robot. To accomplish this, a painting expert (not a programming expert) employs "lead-through teaching" [15] during which the robot is intuitively and expertly guided through the proper motions of the task by the expert user.

Gesture-Based Programming is an extension of the successful aspects of lead-through teaching and also of the emerging "learning by observation" paradigms [6][9]. It extends these ideas by combining multi-agent [1] and skill-based [11] concepts to form *encapsulated expertise* which guides the recognition and segmentation of gestures during teaching and guides error recovery and exception handling of the robot during execution.

In a nutshell, gesture-based programming takes advantage of the wealth of human experience we all gain through our day-to-day lives to specify complex tasks. In a loose sense, gesturing involves "acting out" a scenario, as in learning by observation. 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.

The subject of this paper is not the entire system, which is largely incomplete. Instead, it focuses on our first step toward recognizing and interpreting *tactile gestures* and the framework we developed for doing so. The application interprets tactile gestures for commanding the modification of a robot's trajectory. The goal is to allow the operator to physically "nudge" the robot into the desired trajectory shape with his or her hands (Figure 1) using only sparse, but intuitive gestures. The gestures are observed by a wrist force/torque sensor attached to the robot or a remote trackball outside the workspace.

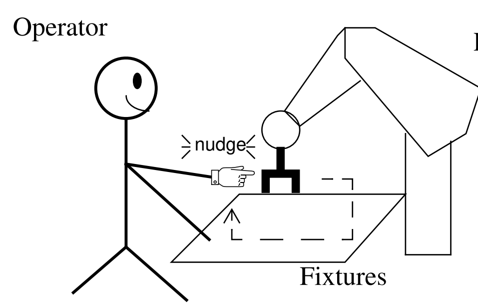

Figure 1: The operator can fine tune the robot's trajectory while changing the fixture setup.

Perception, interpretation and abstraction of these gestures is achieved with a moderately-sized collection of agents (10-15, depending on configuration), each one of which is explicitly programmed to evaluate the user's intentions with respect to a particular attribute of the trajectory. An internal, fuzzified, heuristic model specific to each agent maps gestures to a particular hypothesis relevant to the attribute of the agent. The agents form confidence measures for their respective hypotheses and vote on actions with the strength of the confidence.

While the immediate domain of this application seems somewhat contrived, the broader domain encompasses self-calibration of robotic systems such as a flexible manufacturing system (FMS). Imagine an automated, small-batch assembly process that performs simple pick-and-place operations. Periodically, throughout the day, the widget produced by this FMS changes from among a small set of possible widgets under the direction of a small, semi-skilled staff. Each setup will vary slightly from previous incarnations, so fine-tuning is required. Rather than switching assembly programs and reprogramming parameters on a computer screen, this system allows a moderately-trained operator, or even several soft calibration posts in lieu of the operator, to "nudge" the robot into the correct trajectory and fine tune its parameters using sparse inputs in a natural manner.

2 The trajectory task

As mentioned, the application involves providing intuitive interaction with a robot for modifying its periodic trajectory. The trajectory shape is not free-form, but one of a small number of randomly-chosen trajectory families. For this implementation, each family of trajectories is constrained to the same vertical plane and the set of trajectory shapes includes four: a cross, a rectangle, a right triangle, and a pick-and-place path (Figure 2). There is also a "null" trajectory that allows the operator to pause the robot. Note that the rectangle and pick-and-place trajectories were deliberately chosen to be very similar.

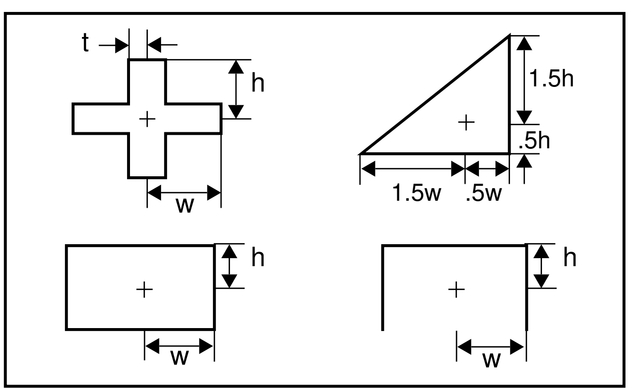

Figure 2: The shapes of the trajectory families.

All shapes have variable width (w) and height (h) and the cross can also vary in thickness (t). These parameters all vary independently. Nothing else but these parameters is allowed to vary, including the orientation and center point (although appropriate agents could be developed). No provision has been made at this time for non-polygonal trajectories.

3 Tactile gestures

Our definition of a gesture is "an imprecise, context-dependent event that conveys the user's intentions." In this case, the gestures are force impulses - *nudges* - on the end-effector. Because gestures are context dependent, state information must be associated with each gesture. This state information is generally application specific. Using a linguistic analogy, the raw gestures form a *gestural alphabet* along with the state information. *Gestural words* are assembled from the raw gesture and its associated context by the gesture recognizers (preprocessors in Figure 5). Gesture interpretation examines these words within *gestural sentences* that are strung together by the user. This analogy of a gestural alphabet is the subject of a subsequent paper and is for illustrative purposes only. Further elaboration is not required to understand this paper.

For the success of this particular application, it is necessary to find a basis for the heuristic models that will guide gesture interpretation. The psychology literature [3],[5],[14] and our own observations suggest the dominant features distinguishing different polygons are the corners. Looking at Figure 3, it is obvious that those are the corners

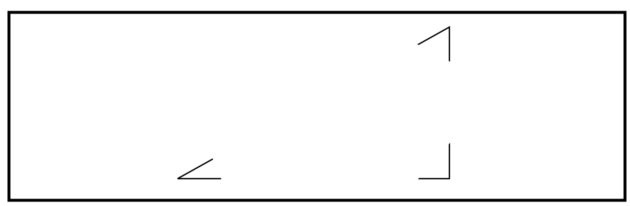

Figure 3: Hints of a triangle.

of a triangle. Even though the sides are substantially missing, the shape is still apparent. It is natural that if one wants to "massage" one polygon into another polygon, he/she would gesture at the corners. Conversely, if one wanted merely to change a parameter of the given polygon - such as elongating it - he/she would gesture on the sides rather than the corners (also suggested by the literature). The lines in Figure 4 do not suggest "rectangle" so much as they suggest some *relationship* such as separation, width, or height. It is interesting to note this division in *influencing* motion runs parallel to work in *perceiving* motion [13]. Furthermore, changing the shape of the trajectory requires the corroboration of several gestures in much the same way as several corners are required to visually suggest a shape.

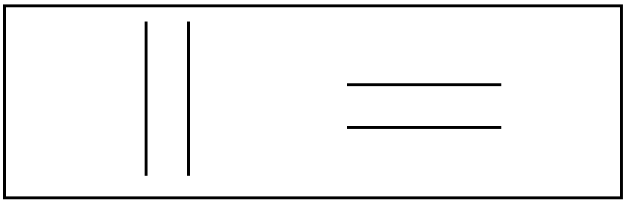

Figure 4: Sides of polygons suggest relationships more than shape.

These observations provide the basis for all of the selection agents. Forces applied at the corners are the dominant gestures for the shape-determining agents while forces applied along the sides are the dominant gestures for the parameter-determining agents.

4 The multi-agent network

We hold a loose definition of agents as *information processors*. In our framework, an *agent* is an entity, either hardware or software, that performs autonomous actions based on information. Autonomous does not imply they must function in isolation; agents may rely on other agents to acquire information or to pre- and post-process information to achieve their goals. Agents can also be hierarchical in that a *collection* of agents can be considered an agent if they process information together.

The important factor in constructing the agents used for this application is true autonomy. We strove to design each agent independently of all other agents, so no agent required knowledge of the other agents in its peer group. This had a strong impact on the implementation of agents but it maintains a high degree of modularity and minimizes inter-agent communication.

4.1 Agent topology

A rough model of the network is a simplified Hearsay-II architecture [10]. Basically, it's a single-level blackboard for inter-agent communication and maintenance of a flat data structure. The data structure keeps track of the most confident agents. (More on this in section 4.3.) Unlike the classical Hearsay-II architecture, scheduling is irrelevant in our context. We specifically want to minimize inter-agent communication and low-level synchronization would consume unnecessary bandwidth. All agents are robustly designed to operate irrespective of the particular collection of peers in the environment or their relative timing. (Collisions have very low probability and minimal impact.)

The agent topology is illustrated graphically in Figure 5. Dark gray ovals are hardware agents and light gray rectangles represent blackboard elements. The preprocessors at the top are the gesture recognizers. (The DataGlove is not yet implemented.) They detect elemental gestures and attach application-specific state information (creating "gestural words"). Peer interpretation agents (height, width, etc.) interpret the gestures, collectively arbitrate the most probable interpretation, and modify the execution parameters of the robot agent and its operational space controller [8].

4.2 Interpretation agents

The interpretation agents consist of a two-stage combination of characteristics from both fuzzy and neural systems. State information associated with each gesture is represented by fuzzy variables but the final confidence value is computed by a biological neuron-like accumulator.

All interpretation agents are independent from one another and are responsible for evaluating the user's intentions with respect to one particular parameter. (i.e. *rect* is responsible for determining if the user is trying to switch to the rectangle trajectory.) These agents utilize the application-specific state information appended to the gestural word by the gesture recognizers which includes:

- average force components
- force magnitude
- perpendicularity of force and vertical (*perp-vert*)
- perpendicularity of force and robot velocity (*perp-vel*)
- parallelism of force and radial vector (*points-out*)
- quadrant location of gesture with respect to center
- closeness to a corner (*corner*)

Most of these are fuzzy variables. The interpretation agents have internal models in the form of fuzzy logic equations that create relevant features specific to the agent's parameter of interest. For example, *rect* uses the feature {*corner* AND *points-out*}. The outputs of the these features, which are themselves fuzzy membership values,

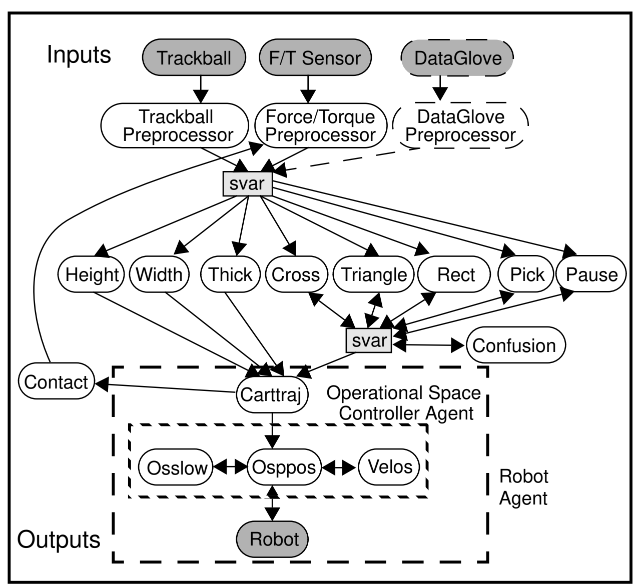

Figure 5: The agent hierarchy.

are combined by the neural element. This "neuron" weights the features positive or negative (reinforce or discredit the hypothesis) and sums them over time. As features are accumulated, there is a constant exponential decay that weights recent stimuli more heavily than past stimuli. Finally, the neuron fires only after the accumulator reaches a threshold. This accumulated value is considered to be the agent's confidence in the user's intentions. Currently, the relevant features and their associated weights are set and fine-tuned by the programmer during agent design.

4.3 Voting mechanism

Each interpretation agent holds the single hypothesis that the user intends to modify the agent's own parameter. It also maintains its own confidence value, from zero to one, that this hypothesis is true based on its fuzzy internal model described above. Accepting a hypothesis as true is generally a distributed three-step process. First, each agent has an internal threshold below which the hypothesis is false. (The "neuron" doesn't fire.) If the confidence exceeds the internal threshold, it is possibly true and is compared to the leading contender on the blackboard. If it exceeds the current leading contender's confidence, it posts itself as the leading contender (its vote overwhelms the opponent's, "Electoral College" style) and awaits a final decision. The agent holding the arbitration token (the most recent previous winner) makes the final confidence comparison and either keeps the token or passes it on to the new "winner." This final step provides hysteresis and some level of noise immunity. Note that this arbitration scheme proceeds regardless of the collection of agents involved.

4.4 Detailed agent descriptions

To illustrate the text above, several representative agent descriptions appear below. All other agents that are not described operate similarly and are described in [17].

4.4.1 Contact. *Contact* is a special purpose agent that allows for predetermined physical interaction with the world. If the robot makes contact with the world, the agents could confuse that information for some type of gesture. Instead of programming agents to beware of such "sensor noise," the contact agent instructs the recognizers to filter out specific impulse vectors that are task-related.

4.4.2 Confusion. This agent serves two very important purposes; it helps us maintain agent independence and it allows for future development. Before explaining how it accomplishes these two things, we'll first describe it.

Confusion acts as a virtual sensor by determining the "confusion" of all the shape agents and calculating a measure of their indecision based only on the value and rate of change of the runner-up confidence. This value is available to all other agents' input ports if their internal models dictate its use. This is useful to any particular agent because it provides information on the "confidence" of the network as a whole, rather than an individual peer.

By examining the confusion measure, any agent can determine a rough upper bound for the confidence of all other agents without being aware of their individual goals or identities. This helps to ensure agent independence and modularity. Also, the confusion agent does not violate modularity since its calculation is based only on the runner-up confidence value without dependence on the particular agents that are posting. Finally, the failure or removal of this agent merely results in a defective virtual sensor and the other agents continue operating.

In future work, we want the network to be self-modifying to effect gesture-based programming of multi-agent systems. 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.

4.4.3 Width. The width and height agents are identical in operation. Only the direction of the force features and the parameter on which they act are different. The actions of *width* are determined by a very simple internal model that encodes a small number of distinct features. Positive (reinforcing) gestures are: {{{*perp-vert* AND *not-perp-vel*} OR {*perp-vert* AND *perp-vel*}} AND *not-corner* AND *confusion*} while negative gestures are {*not-perp-vert*}. (Width is horizontal, hence the importance of *perp-vert*.)

From an intuitive standpoint, the model works like this: If the end effector is moving along a side of the polygon and I push it further along the side (force A, Figure 6), I must want to elongate that side *if the shape confusion is high*. Likewise for shortening. On the other hand, if I pull out on a side of the polygon, it's not clear what I'm trying to do. I could be trying to change the thickness, the width, or even the shape. But if I repeatedly pull out on opposite sides of the polygon, I must be trying to increase the width (forces B and C, Figure 6)

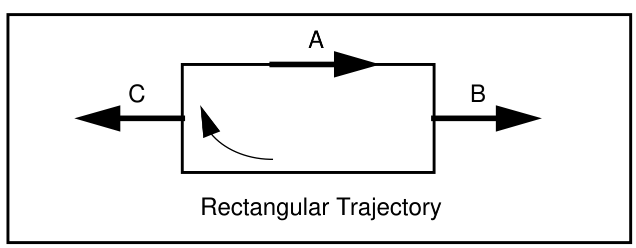

Figure 6: To increase the width, either apply force A or alternately apply forces B and C.

4.4.4 Rectangle. The rectangle model is based on gestures that tug out at the corners - or what would be the corners if the current shape was a rectangle. Therefore, outward forces at any corner have high rectangle membership (“excitatory” in the graphical representation of Figure 7). Inward forces anywhere have high membership in “not rectangle” (“inhibitory”). If the gesture is applied along a side, it is excitatory if it is outward in the direction of the corners of a very large square. If the force is oblique with respect to the velocity vector, it is slightly inhibitory as are attempts at reversals in direction.

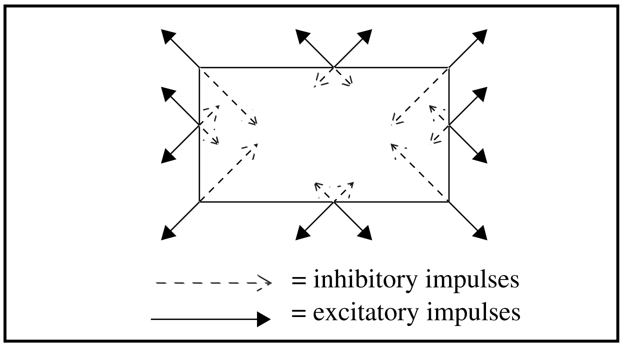

Figure 7: Graphical representation of the internal model of the rectangle agent.

5 Implementation

This application is implemented as reconfigurable software modules under the Chimera real-time operating system developed at Carnegie Mellon. Chimera has low-level control over three PUMA robots that have been modified with PUMA Interface Boards from Trident Robotics [16] to eliminate VAL. Because of the flexibility of Chimera’s *reconfigurable software module library*, the selection of the physical robot and the configuration of agents is dynamically alterable at run-time. Multi-agent simulation tools provide a number of different arbitration, communication, and synchronization mechanisms to allow simulation of a wide variety of network topologies.

For gesture recognition, all agents execute as periodic tasks in a multiprocessor environment, some actually sharing the same processor. Dynamic reconfiguration of the agents during run-time is accomplished via a high-level GUI called Onika [2], also developed at Carnegie Mellon. This allows the easy testing of agents alone and in variable configurations. We plan to implement dynamic self-reconfiguration and full gesture-based programming because we have found visual programming using Onika to be rather awkward for very complex, sensor-based tasks.

Most of the communication between agents is simulated broadcast messages. Broadcasting is simulated blackboard-style through Chimera’s state variable facility. State variables are equivalent to shared memory with the addition of automatic address resolution and several other desirable features.

6 Experimental trials

Several trials were executed with a trained operator using both a desk-mounted trackball and a gripper-mounted force/torque sensor as input devices.The agents were tested in approximately ten “significant” configurations where significant is defined as consisting of a minimum of ten of the roughly fifteen available agents.

With all agents operating and a trained operator, Figure 8 illustrates a trial of the system switching between the triangle and rectangle trajectories. The top strip chart shows time histories of the X and Z positions of the robot (top line is X-position, bottom line is Z-position) with the corresponding end effector forces superimposed. The bottom two boxes are spatial (X-Z) representations of one actual cycle of each trajectory family with force vectors showing points of application of the impulses. The top strip chart and bottom two plots are different representations of the same data; the former temporal, the latter spatial.

In the first phase of the strip chart in Figure 8 (0 to 13 seconds), the robot is executing a triangular trajectory which is depicted spatially in the bottom left box. Gestures f1 and f2 (visible as positive blips in the strip chart data) are applied to the hypotenuse, pulling out on the side to create another corner. This causes the confidence of the rectangle agent to dominate and the trajectory switches. Phase 2 (13 to 40 seconds) is rectangular, but gestures that push the end effector along the hypotenuse of the intended triangle (f3 and f4) and one that collapses the corner of the rectangle cause a switch back to the triangular trajectory.

These two transitions occurred after only two and three gestures, respectively. On average, it takes about four gestures to properly discern the user’s intentions. This particular trial violated an assumption on which the system is based: changes to the trajectory are rather few and far between. As a result, the mode changes happen somewhat more rapidly than they otherwise would due to the decay time of the confidence value. In normal operation, a trajectory change typically takes about four individual gestures to induce.

7 Related modification approaches

There is no other work that we are aware of that directly addresses this trajectory modification in a manner even remotely similar. The standard techniques for dealing with this type of variation are menus, keyboards, and teach pendants. A menu could certainly be used for selecting the trajectory shape, but our goal is to provide a user interface that is close to the robot’s workspace. During setup, the

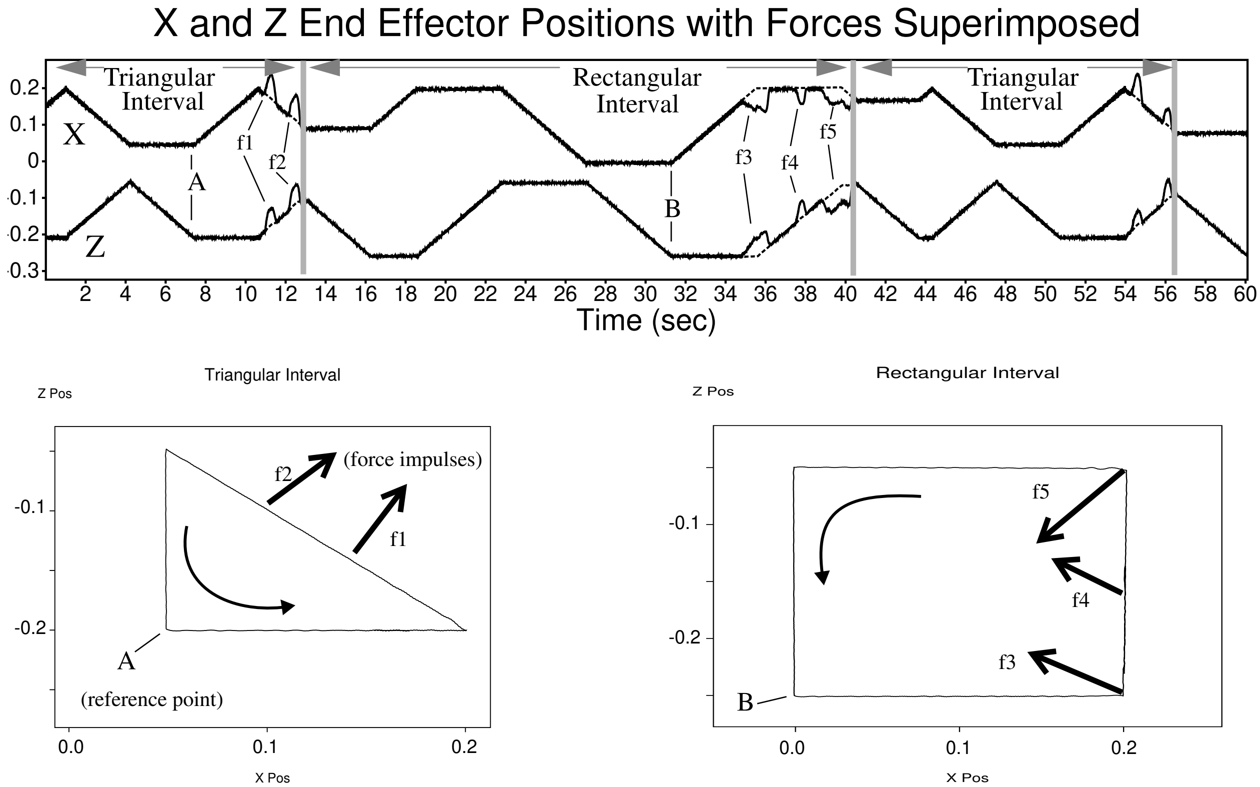

Figure 8: Annotated temporal plot (top) of the robot's position in the plane superimposed with force impulses. Spatial plots (bottom) show planar trajectory intervals with corresponding annotations.

operator is in the workspace anyway, adjusting the fixtures for the new parts. We feel it is beneficial to keep the operator's attention focused on the workspace, rather than wandering off to the computer screen.

Entering, on a keyboard, tuned parameters measured relative to the particular setup is not practical because accurate measurements take too much time for a small-batch process. Using a teach pendant or joystick, which can be moved closer to the action to make differential changes (rather than accurate measurements), is possible but not very intuitive. This type of interface is similar to our trackball implementation, which suffered in comparison to the F/T sensor due to the required mental frame transformation. Although a teach pendant could be used with dedicated buttons to implement a "hardware menu," this is less flexible and still suffers from the non-intuitive, discrete transformation from motion-direction to pendant-button. Re-executing a "canned" calibration sequence is the most common approach to tuning trajectory parameters but, again, the state changes (into and out of calibration mode) and execution consume valuable time.

There are some applications that have similarities to ours, despite a different task domain. Pook and Ballard [12] implemented a learning strategy so their Utah/MIT hand could flip eggs in a frying pan. Their internal representations for *primitives,* such as pressing and grasping, are based on particular force signatures much like our heuristic models. Key differences are their models are learned over repeated trials while ours are explicitly programmed and their system is monolithic versus our multi-agent approach.

The work of Hirai and Sato [4] also carries some similarities in their *symbolizers*. Symbolizers are agents that help maintain an internal world model during telerobotic manipulation. They are a cross between our preprocessors and perceiving agents: they process and produce virtual sensor information. The symbolizers are significantly more limited, however, because they have minimal decision capability.

We attempted two other monolithic approaches on a subset of the task, but neither was successful. Our first alternative was "snake-based" [7]. We assumed a "snake" in the shape of the current trajectory and tried to apply the measured force impulses to deform it. The deformed snake was continuously compared to all candidate shapes to see which it matched (in a minimum potential energy sense). We had no way to vary parameters, though.

Our second alternative was the creation of a virtual "force retina" from which we hoped to extract features in a classical vision sense. But to capture all the information we thought we needed, the dimensionality of the retina became too great and we abandoned the approach. Interestingly, this pointed us back to the multi-agent approach as a way to handle the high dimensionality of the sensed data.

8 Conclusions

A multi-agent perception system capable of inferring intentions from tactile gestures for modifying and fine-tuning robot trajectories was successfully implemented. With a trained operator, the gesture abstraction works quite well and provides a novel framework for human/computer interaction. By "quite well," we mean a trained operator can generally change the trajectory and fine tune it with fewer than five applied gestures. Simple changes are generally possible to achieve with only two or three gestures as evidenced in figure 8.

The multi-agent approach to tactile gesture understanding based on our gesture basis parameters was quite satisfying. Not only does it provide a modular way to build interpretation/abstraction systems, it also provides natural hooks, in the form of knowledge encapsulation agents, for future development of a programming environment. Programming not only involves configuring the agents statically, but also involves dynamic self-reconfiguration based on proprioceptive sensing. The *confusion* agent is a proprioceptive virtual sensor that we intend to use in this manner.

Finally, some mention should be made of the applicability of this application to the real world. Although it was conceived primarily as a vehicle to explore tactile gestures (and we feel the big contribution will be in gesture-based programming), we feel the application has some relevance to the real world. Tactile gesturing is not aimed at large, high-volume operations. But, the concept could be implemented at small companies with job-shop missions whose direct employees do not have rigid job descriptions. The ideal operator is one whose job is "to get the job done." These types of small-company environments are more receptive to such hands-on approaches with appropriate safeguards. Of course, we have also demonstrated tactile gesturing on a remote trackball and we are working on non-tactile gesturing with a DataGlove.

There is another mode of operation for tactile gestures we are beginning to explore: self-calibration. If the workspace of the robot is populated with a series of "soft find-posts" - spongy calibration posts that provide the equivalent of tactile gestures for fine-tuning the trajectory - automatic calibration could be performed for each new setup without running a separate calibration routine and without the need of the user's input. A secondary benefit of the softness of the findposts is that they can be placed more intrusively into the workspace, providing more accurate references for the task at hand. This is possible because the system designer does not need to be as concerned about accidental collisions with a soft object.

9 Acknowledgments

We would like to thank Prof. Steve Shafer for his input. Mr. Voyles was supported, in part, by a National Defense Science and Engineering Graduate Fellowship.

10 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] 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.
[3] Hebb, D.O., "The Organization of Behavior; a Neuropsychological Theory," New York, Wiley, 1949.
[4] 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.
[5] Hoffman, D.D. and W.A. Richards, "Parts of Recognition," *Cognition*, v. 18, pp. 65-96, 1984.
[6] 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.
[7] Kass, M., A. Witkin, and D. Terzopolous, "Snakes: Active Contour Models," *International Journal of Computer Vision*, v.1, n.4, 1987, pp. 321-331.
[8] Khatib, O., "The Operational Space Formulation in Robot Manipulator Control," in *Proc. of the 15th International Symp. on Industrial Robots*, v. 1, pp. 165-172, Sept. 1985.
[9] 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.
[10] Lesser, V.R. and L.D. Erman, "A Retrospective View of the Hearsay-II Architecture," in *Proc. of the Int. Joint Conf. on Artificial Intelligence*, Cambridge, MA, pp. 790-800, 1977.
[11] 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.
[12] Pook, P.K. and D.H. Ballard, "Recognizing Teleoperated Manipulations," in *Proc. of the 1993 IEEE Int. Conf. on Robotics and Automation,* v. 2, May, 1993, pp. 578-585.
[13] Rubin, J.M. and W.A. Richards, "Boundaries of Visual Motion," MIT AI Lab tech. report 835, April, 1985.
[14] Shepp, B.E. and S. Ballesteros, editors, *Object Perception: Structure and Process*, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1989, chapter 5.
[15] Todd, D.J., Fundamentals of Robot Technology, John Wiley and Sons, 1986, chapters 3, 7.
[16] *TRC004 PUMA Interface Board User's Manual*, Trident Robotics and Research, Inc., Wexford, PA, April, 1994.
[17] 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.