Technical Report

# Multi-Agent Perception for Human/Robot Interaction: A Framework for Intuitive Trajectory Modification


Richard M. Voyles, Jr.
Pradeep K. Khosla



CMU-RI-TR-94-33



Advanced Manipulators Laboratory, The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3891



September 15, 1994





Mr. Voyles was supported in part by a National Defense Science and Engineering Graduate Fellowship and by Sandia National Laboratories through contract number NAG1-1075

# Abstract

An application of distributed perception to a novel human/computer interface is presented. A multi-agent network has been applied to the task of modifying a robotic trajectory based on very sparse physical inputs from the user. The user conveys intentions by nudging the end effector, instrumented with a wrist force/torque sensor, in an intuitive manner. In response, each agent interprets these sparse inputs with the aid of a local, fuzzified, heuristic model of a particular parameter or trajectory shape. The agents then independently determine the confidence of their respective findings and distributed arbitration resolves the interpretation through voting.

# Table of Contents

# List of Figures

# 1 Introduction

Distributed problem solving (DPS) has had a long history in computer science [1][2], but, historically, little of this work was applied to hard real-time systems. Behavior-based frameworks, a relative new-comer to the DPS field, have made strong contributions to the real-time world of robotic control as evidenced by the subsumption architecture [3] and others [4][5]. The success of these frameworks has been due to the inherent decomposition of difficult tasks into autonomous, simple behaviors and the resultant reconfigurability and reusability of the constituent modules.

Distributed systems rely on the decomposition of the problem to make things tractable. This results when difficult problems can be attacked by focussing on and refining constituent parts in a building-block manner. These building blocks, once tested and confirmed, are used to create increasingly complex algorithms with ever more powerful features [6]. These reusable components, be they hardware or software, form a set of reconfigurable modules from which various and much more complex systems can be constructed [7]. As the structure becomes more complex, minor adjustments to the foundation may be necessary, but these adjustments are much easier to conceive and achieve as a result of the decomposition. The compartmentalization results in *fewer* interactions within the decomposed system than an equivalent monolithic one.

These advantages are now being heavily exploited in the domains of planning and control but, except for the field of speech recognition [8], high-level perception, in the classical sense, has largely remained encapsulated in monolithic systems or subsystems.

This paper describes an experiment that applies a multi-agent network to a high-level perception task to produce a novel man-machine interface. It focuses on the implementation of a system for modifying a robot's trajectory based on sparse force inputs applied to the manipulator's end effector. The goal is to allow the operator to physically "nudge" the robot into the desired trajectory shape with his or her hands. The sparseness of the inputs, both temporally and spatially, make the perception task one of *inferring intention* - that is, determining what the user *intends* to happen based on incomplete information and a heuristic "model" of the world - rather than the data-rich assumptions of traditional perception.

We achieve this with a moderately-sized collection of peer 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 user inputs 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.

This project arose as just one thrust of a multi-faceted study in non-traditional control systems. Applications of multi-agent architectures, evolutionary systems, and primitive-based control are being examined to provide a framework for intelligent robotic systems. This particular project was conceived to explore multi-agent networks for perception and to investigate minimizing information flow between agents. The application was designed to allow extension of the framework into the domain of human/computer interaction.

While the immediate domain of this project 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 Task

The goal is to develop a framework allowing intuitive interaction with a robot for modifying its periodic trajectory. The trajectory shape is not free-form, but one of a small number of trajectory families. For this implementation, each family of trajectories is constrained to the same vertical plane and the set of trajectory shapes initially included three: a cross, a rectangle, and a right triangle (see Figure 1.) These “toy” shapes were chosen just for the sake of visualization. Subsequently, a simulated pick-and-place trajectory was added which will be discussed later.

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).

The user can only communicate with the system by exerting forces on the 6-DOF force/torque sensor at the end effector (or a similar trackball on the table). Since the implemented trajectories are themselves planar, only force components in the plane are used at this time.

## 3 Distinguishing Force Features

The basic intent of the application is to provide a user interface for modifying polygonal robot trajectories. The key to its success is finding intuitive features for tugging the existing trajectory into

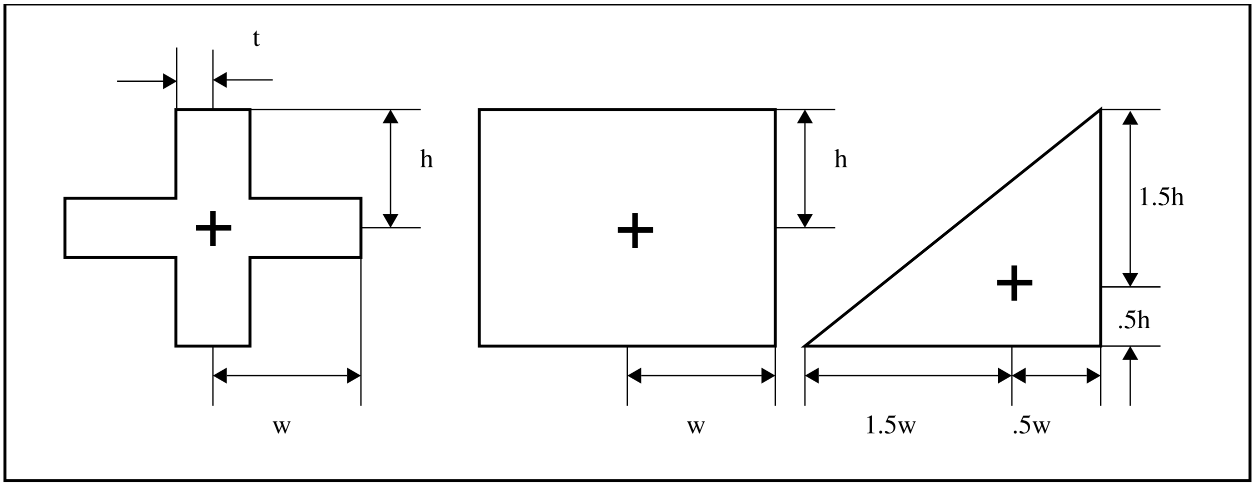


**Figure 1: the Shapes of the Trajectory Families**

the desired trajectory. (A different paradigm could be developed for non-polygonal trajectories, but the underlying multi-agent framework would remain the same.)

A subset of the psychology literature [9],[10],[11],[13] and our own observations suggest the dominant features distinguishing different polygons are the corners. It seems natural that if one wants to "massage" one polygon into another polygon, he/she would tug at the corners. Conversely, if one wanted merely to change a parameter of the given polygon - such as elongating it - he/she would tug on the sides rather than the corners (also suggested by the literature). Furthermore, changing the shape of the trajectory requires the corroboration of several of these "force features" (as perceived by the agents), hence, response is delayed. There are other modifications - elongation, for example - for which the user's intentions can be deduced immediately. Response to these input features can be instantaneous.

These observations provide the basis for all of the perceiving agents. Forces applied at the corners are the dominant features for the shape-determining agents while forces applied along the sides are the dominant features for the parameter-determining agents. Of course, there are some exceptions and special cases that particular agents employ, but in general, these two basic rules dominate the agents' decisions. Each agent has an informal model of which of these features strongly support or discredit that agent's particular hypothesis on the user's intentions.

## 4 Agents

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, in this context, 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. As such, agents can be recursive in that a *collection* of agents can be considered an agent if they process information together. This implies a natural hierarchy but also dictates that care must be taken to explain peer relationships to avoid confusion between levels.

For example a robot arm is a hardware agent that "processes" torque information to act on the physical world. A PID controller is a software agent that processes desired and current position information to produce a new piece of information: torque commands. Without a robotic agent, this piece of information (the torque commands) may be worthless, but that is not the concern of the controller agent. It is merely concerned with producing the new piece of information, regardless of how it is used (i.e. a robot, a simulator, a GUI, etc.) On the other hand, a robot, a controller, and a trajectory generator taken together can be considered an agent for producing desired motions in the physical world. Individually, the agents work autonomously to achieve independent goals, but taken as a group, the goals and actions combine to achieve a new and broader goal. At the extremes, a suite of programs (TCP/IP, for example) can be an agent, while normal subroutines can be *piecewise autonomous agents* because they process information (arguments and global variables) but are autonomous only for the duration of the call. (Their operation is strictly gated by the calling program.)

This definition places no artificial constraints on the design of the agents such as prohibiting the saving of state [3]. Quite to the contrary, we feel that saving state is important. In the grand scheme, we envision learning and adaptive agents serving the role of teacher - helping the execution agents improve their performance. But some philosophical beliefs on the usefulness and benefits of multi-agent architectures guided our implementation. First of all, the agents were designed to be as simple as possible - concerning themselves with generally only one task. The philosophical justification for this is modularity and flexibility of combination. Secondly, intercommunication was minimized. This is a natural outgrowth of the first philosophical point. If the agents are simple, a large variety of them are required to perform complex tasks. If there are a lot of agents, there is a practical need to reduce communication bandwidth. (In the microfabrication/MEMS domain, with potentially thousands of agents, this becomes particularly important.)

Since the distinguishing force features of the trajectory families are common to all perception tasks, and they are not native outputs of the sensors, we decided to perform some centralized preprocessing. Although one agent performs all preprocessing, it creates several virtual sensors with which the agents communicate as needed. Likewise, joint-level control of the robot is not distributed among the perception tasks. Joint-level control is segregated in a separate multi-agent network (an agent) that performs operational space control. (This is described in-depth in section 4.2.) This is a deliberate attempt to guarantee that only the specified trajectories result, not free-form motion.

As a result of these consolidations, there are two basic types of agents: *perceiving agents* and *utility agents*. The utility agents provide the common utilities and housekeeping functions that the perceiving agents need to carry out their tasks. The utility agents consist of:

- *preft* - signal preprocessing for force/torque sensor
- *pretb* - signal preprocessing for trackball
- *the robot* - an agent that consists of the following agents:
  - *osppos* - operational space robot controller
  - *velos* - calculates filtered cartesian velocity
  - *osslow* - calculates jacobian-related terms of operational space controller
  - *carttraj* - cartesian trajectory generator

In keeping with the simplicity directive, each perceiving agent was made responsible for only one parameter or trajectory shape. This provides the greatest modularity and flexibility for combining agents and minimizes their interaction. Since there are three parameters and three shapes, the perception tasks are performed by six agents:

- *width* - modifies the width parameter
- *height* - modifies the height parameter
- *thick* - modifies the thickness parameter
- *rectangle* - votes for the rectangle trajectory family
- *triangle* - votes for the triangle trajectory family
- *cross* - votes for the cross trajectory family

### 4.1 Agent Topology and Voting Mechanism

A rough model for the network architecture in which the above agents were connected is the Hearsay-II architecture [8]. There are several significant differences, however. Most important is the blackboard. In our network, the single-level blackboard serves only as a simple communication mechanism, not as a "data structure," per se. The agents are also simpler, acting on only one known hypothesis rather than generating new ones. Finally, scheduling is irrelevant in our context. We specifically want to minimize communication and low-level synchronization would consume unnecessary bandwidth. All agents are robustly designed to just "do their thing" irrespective of relative timing. (Collisions have very low probability and minimal impact.)

The internal structure of all perception agents is similar and based on a simplified hypothesize-and-test model for casting their votes. Each 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. A simple internal model, represented by a set of fuzzified heuristics that encode the types of force features that reinforce and weaken the hypothesis, is used to increase or decrease the confidence value (the "test" of the hypothesis). Fuzzification is employed to group the direction and point of application of the applied force into finite sets. Finally, the confidence value continuously decays with time to place greater weight on recent stimuli.

Accepting a hypothesis as true is generally a three-step process involving distributed conflict resolution. First, each agent has an internal threshold below which the hypothesis is false. (Essentially, it does not get to vote.) 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) and awaits a final decision. The agent holding the arbitration token makes the final confidence comparison and either keeps the token or passes it on to the new "winner." Note that this arbitration scheme proceeds regardless of the particular agents involved.

As mentioned previously, the internal models of some agents contain directives to circumvent this process if the user's intentions seem obvious. In these cases, the confidence is assumed so high that voting is bypassed and the parameter is modified immediately. These are special cases of the heuristic models and are akin to reflex actions in biological systems.

Because all the perception agents are peers, there is no specific hierarchy; they can be freely combined in any arrangement. However, certain tasks are centralized within utility agents, and this imposes a specific hierarchy for the overall system. This hierarchy, in which "svar" represents parts of the blackboard communication mechanism, is shown in Figure 2. (*Pick* and *confusion* will be introduced later. The DataGlove elements have not yet been implemented.)

### 4.2 Operational Space Control Agents

The operational space controller is broken into several agents that are described in subsequent subsections. The agents used for this application implement a reduced subset of the full operational space controller to reduce CPU burden and to demonstrate the reconfigurability of the multi-agent implementation. The rest of this section detours momentarily to describe the operational space formulation and can be skipped without loss of continuity.

The important concept of the operational space formulation [12] is the selection of an appropriate *space* that is a natural complement to the task, or *operation*. In most cases, cartesian space is selected because it is an intuitive basis for human interaction. Within this operational space is an *operational point* with an attached frame. The selection of this point and frame is also suggested naturally by the task but is dictated solely for the programmer's convenience. It is from this point

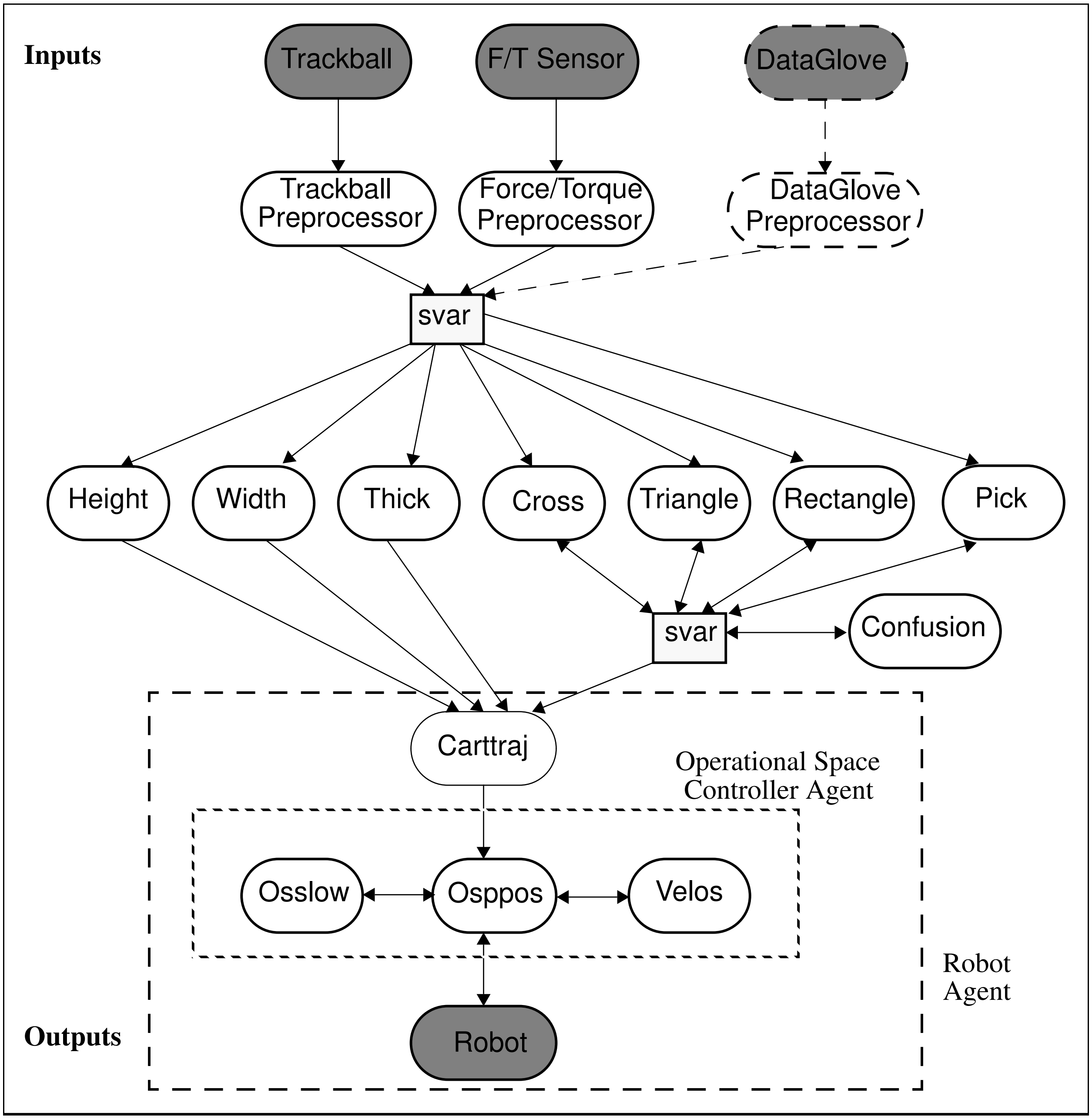


**Figure 2: The Agent Hierarchy**

and frame, chosen in such a way as to ease the task specification, that the operational space formulation derives its elegance.

Accomplishing the task simply reduces to applying appropriate forces to the operational point so that it moves toward the goal in operational space. These forces can be calculated by any control algorithm of choice and applied to unit inertias which are assumed along the axes of the operational frame. The calculated forces must be scaled by the effective inertias of the object/manipulator combination in order to produce the desired motions. The final step is just the mapping of the generalized forces in operational space to the motor torques in the manipulator joint space.

This formulation is most elegant when applied to complex object/manipulator combinations, but it is also rather compute-intensive under those circumstances. For this project, we begin with a single manipulator and assume we want to track a position/velocity trajectory in space. The forces of motion on the unit inertia at the operational point (attached to the end effector) can be calculated with a simple proportional-plus-derivative (PD) controller:

$$\mathbf{F}'_m = -k_p(\mathbf{x} - \mathbf{x}_d) - k_v(\dot{\mathbf{x}} - \dot{\mathbf{x}}_d) \tag{1}$$

$k_p$ and $k_v$ are the controller gains while $\mathbf{x}$ and $\mathbf{x}_d$ are the actual and desired position vectors, respectively, in operational space. To find the actual forces that must be applied to the object/manipulator combination, dynamic factors must be considered:

$$\mathbf{F}_m = \Lambda(\mathbf{x})\mathbf{F}'_m + \mu(\mathbf{x}, \dot{\mathbf{x}}) + \mathbf{p}(\mathbf{x}) \tag{2}$$

where $\Lambda(\mathbf{x})$ represents the inertia matrix or kinetic energy matrix, $\mu(x, \dot{x})$ is the coriolis and centrifugal force vector, and $\mathbf{p}(\mathbf{x})$ is the gravity vector. It is important to remember that these terms are referenced to operational space, not joint space. For this particular project, we ignored the coriolis and centrifugal terms because low speeds make them irrelevant in this context. $\Lambda(\mathbf{x})$ is determined from geometric and inertial knowledge of the manipulator:

$$\Lambda(\mathbf{x}) = \mathbf{J}^{-T}(\mathbf{q})\mathbf{A}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q}) \tag{3}$$

where $\mathbf{J}(\mathbf{q})$ is the jacobian and $\mathbf{A}(\mathbf{q})$ is the inertia matrix of the links in joint space.

With the forces required to produce the desired motion known, it is easy to transform them into torques in joint space by simply pre-multiplying by the jacobian transpose:

$$\Gamma = \mathbf{J}^{T}\mathbf{F}_m \tag{4}$$

where $\Gamma$ is the vector of joint torques.

The operational space formulation provides an ideal environment for force control. The fastest control loop contains the force calculations, resulting in maximum bandwidth for the force servo. It also provides a consistent framework for thinking about force control because the commands are never translated into artificial position errors. The formulation also suggest a natural separation between compute-intensive constructs and the high-speed servo. The jacobian and inertia matrices change slowly and can be considered quasi-static. As a result, they are computed at rates much below that of the force servo. With this in mind, the following sections detail the sub-agents constructed, as a result of this natural decomposition, to collectively form the operational space control agent.

#### 4.2.1 OSPPOS

Osppos uses a PD cartesian position loop and data generated by osslow and velos to generate appropriate command forces in cartesian space as in equations (1) and (2). The agent that calculates centrifugal and coriolis forces is not required for this application so $\mu$ terms are zero. Equation (4) converts the cartesian forces to torques which are written to the robot. All velocity and jacobian references are treated as virtual sensor data provided by other agents.

#### 4.2.2 VELOS

This agent generates filtered cartesian velocity data. Because the data is filtered, it can be considered quasi-static as long as the velos agent executes at approximately the same speed as osppos. Under this assumption, there is no need to explicitly synchronize the agents.

#### 4.2.3 OSSLOW

Osslow generates the data that changes slowly such as $\Lambda$, $\mathbf{J}$, and $\mathbf{J}^{-1}$. Since these matrices change comparatively slowly, there is, again, no need to synchronize with osppos.

### 4.3 CARTTRAJ

Carttraj generates all the cartesian trajectories. It receives messages from all the other agents through the blackboard mechanism and generates the specified path. It is necessary to have an agent directly responsible for the cartesian trajectory because the shape of each possible trajectory is rigidly pre-determined. If the perception agents were influencing the motions of the robot directly, there would be complex interactions among their conflicting desires and the shape of the trajectory would be difficult to predict. Instead, they communicate their *desires* for the trajectory and, through distributed conflict resolution, mutually agree on what it should be.

### 4.4 Preprocessing

*Pretb* and *Preft* condition the trackball and force/torque sensor inputs to provide a consistent interface that allows the substitution of different input devices. They also perform mathematical operations on the raw data. The operations they perform average the force inputs over the duration of the impulse to determine average magnitude and direction of the force and the degree of parallelism to end effector velocity, to end effector displacement from center point, and to the vertical. They also determine if the force was applied at a corner or on a side. To support both immediate and delayed actions of the agents, the preprocessors maintain information on the current state of the force input and the previous impulse applied.

These agents can run concurrently, allowing inputs to be provided interchangeably from either trackball or F/T sensor (although, not both inputs simultaneously).

### 4.5 WIDTH and HEIGHT

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. These actions are determined by a very simple internal model that encodes a small number of distinct features which meet three criteria: the forces must be parallel or perpendicular to the parameter; the forces must be parallel or perpendicular to the direction of motion; and the forces must act on a side of the polygon (rather than a corner).

If the force feature is applied parallel to the direction of motion along a side of the polygon that is parallel to the parameter of interest, an immediate change in the parameter is effected. (See force A, Figure 3.) Whether the parameter is increased or decreased depends on whether the force is parallel or anti-parallel to the direction of motion.

If the feature is perpendicular to the direction of motion and parallel to the parameter of interest, the confidence is increased. (Forces B and C, Figure 3.) If the feature is perpendicular to the direction of motion and perpendicular to the parameter of interest, the confidence is decreased. Internal arbitration determines when to effect a change.

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, I must want to elongate that side. 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.

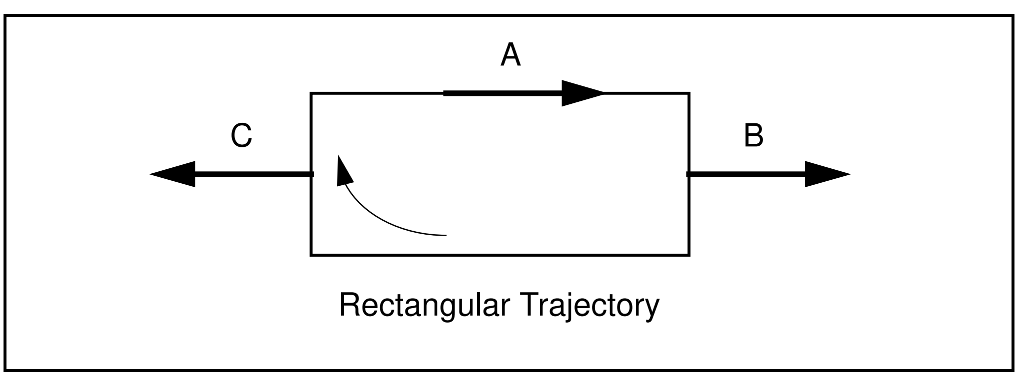


**Figure 3: To increase the width, either apply force A or alternately apply forces B and C.**

## 4.6 THICK

Not all the parameters defined by the multi-agent system apply to all trajectories. Thickness is an example of this. It applies to the cross trajectory but not to the triangle or rectangle. However, there is no central intelligence that disables the thickness agent. All agents are peers with no explicit knowledge of the existence of other agents. Agents only have knowledge of the existence of agent by-products which are virtual sensors. (If no agent is maintaining a virtual sensor, the sensor appears to be disconnected, but consumer agents are not explicitly aware of this.)

Nonetheless, the thickness agent is very much like the width and height agents in principle, except there is no immediate update of the parameter. The implementation, on the other hand, includes a key difference from *height* and *width* in that the confidence is a signed value instead of a positive quantity. This better accommodates the fact that multiple forces applied in different directions best disambiguate thickness intentions. Forces consistent with increasing the thickness make the confidence more positive while forces consistent with decreasing the thickness make the confidence more negative. Time still decays the confidence to zero, as in *height* and *width*.

The internal model relies on forces that are perpendicular to the direction of motion of the end effector and favors forces that alternate between the horizontal and vertical directions. (This saving of state is implicit in the signed representation of confidence and does not require additional state information.) All other impulses are mildly inhibitory.

## 4.7 RECTANGLE

As in all the shape agents, the confidence mechanism is used in the rectangle agent very much as it is used in the parameter agents described above. The key to understanding its operation is in understanding the internal model and the arbitration mechanism used for changing from shape to shape.

The model is based on tugging out at the corners - or what would be the corners if the current shape was a rectangle. Therefore, outward forces at any corner are excitatory. Inward forces anywhere are strongly inhibitory. If the force 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. Other impulses are ignored. This model is represented graphically in Figure 4.

To assist in arbitrating the “winning” shape, the agent compares its confidence to the “best runner-up” which is posted on the blackboard. If the rectangle’s confidence value exceeds that of the current best runner-up as posted in the table, the old value is overwritten with the new value and it is tagged as belonging to the rectangle family. Final arbitration of the winning shape is done by the currently executing shape. The current shape (which holds the “token”) compares the best runner-

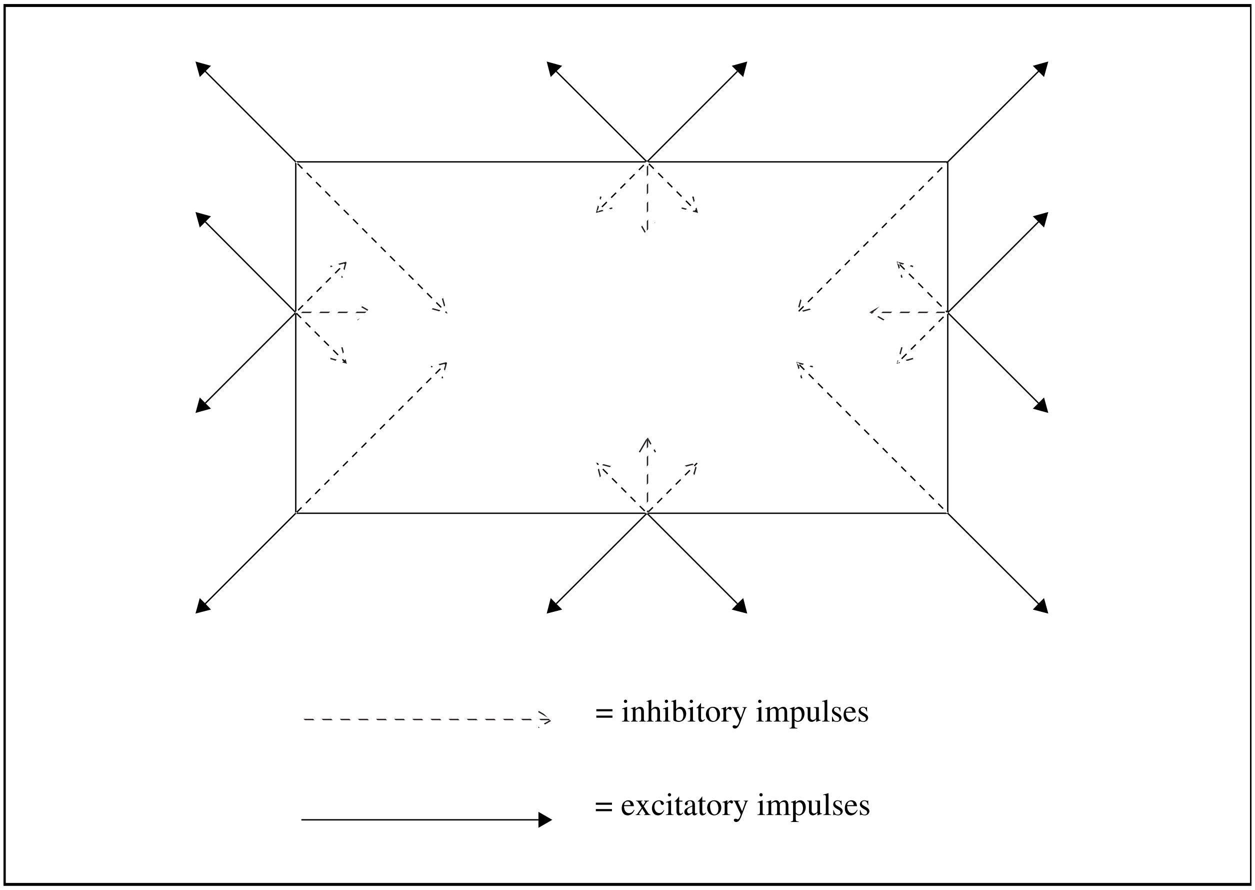


**Figure 4: Graphical representation of the internal model of the rectangle agent.**

up confidence to its own internal confidence and passes the token if necessary. To prevent spurious changes in trajectory shape, the confidence value of the shape holding the token is not allowed to fall below a certain threshold. This tends to maintain the status quo.

### 4.8 TRIANGLE

Since the triangle is asymmetric, the model for inhibitory and excitatory impulses is indexed by the quadrant in which the impulse was applied, relative to the center. Again, forces applied at the corners are most significant, plus forces along the hypotenuse.

There are four cases for forces applied at the corners. If the corner corresponds to the hypotenuse and the force points in, it is strongly excitatory. If it points out, it is strongly inhibitory. If the force points out from any other quadrant, it is slightly excitatory. Other corner forces are ignored.

The only other model features deal with the hypotenuse. If the force is roughly parallel to the hypotenuse, it is moderately excitatory. If the force is perpendicular, it is strongly inhibitory. See figure 5 for a graphical representation.

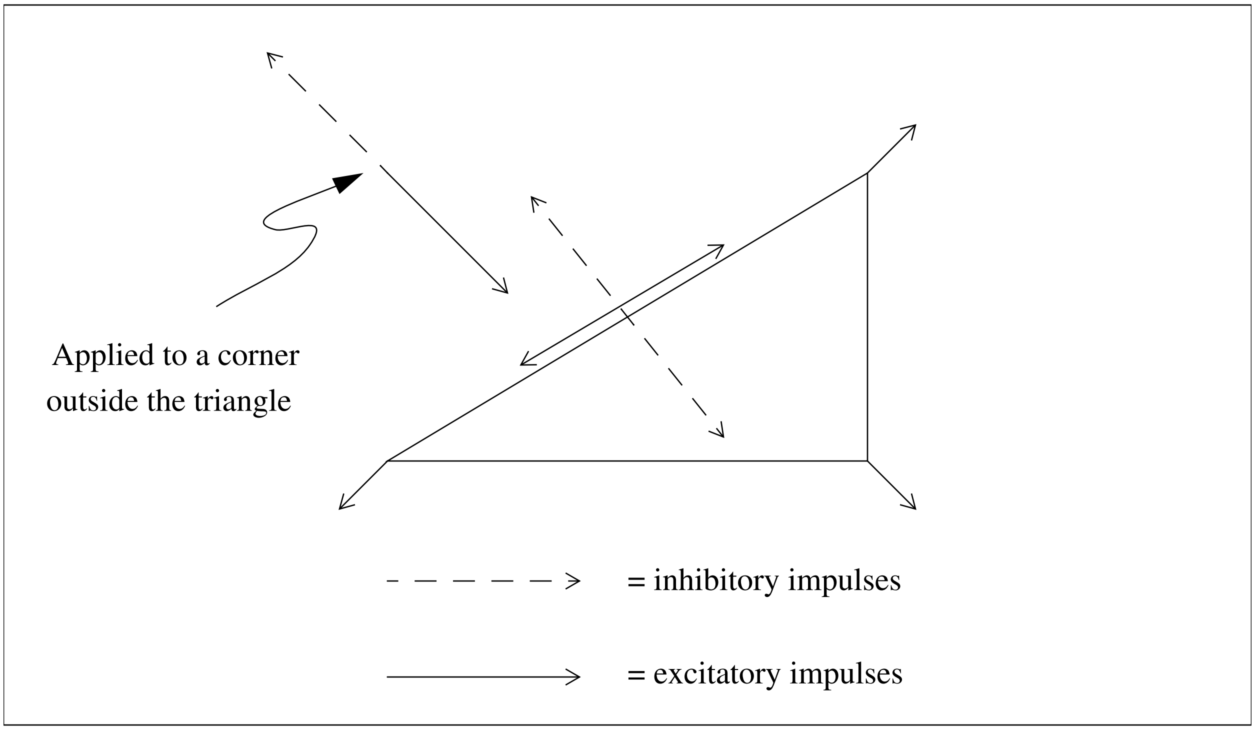


**Figure 5: Graphical representation of the internal model of the triangle agent.**

### 4.9 CROSS

Finally, the cross model is nearly the opposite of the rectangle. Forces pointing in at the corners are excitatory while forces pointing out at the corners are inhibitory. Forces along the sides of the polygon have similar effects. If they point in at an angle, they are excitatory. If they point out at an angle, they are inhibitory.

## 5 Implementation

This entire application is implemented as reconfigurable software with reusable modules using multi-agent simulation tools running under the Chimera real-time operating system developed at Carnegie Mellon [14]. Chimera has low-level control over three PUMA robots that have been modified with PUMA Interface Boards from Trident Robotics 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. The multi-agent simulation tools provide a number of different communication, arbitration, and synchronization mechanisms to allow simulation of a wide variety of network topologies. However, we feel the particular topology of a network, although important for a particular application, is inconsequential to the multi-agent concept. In fact, a truly general multi-agent framework should be able to embody even normal subroutines as *piecewise autonomous agents*.

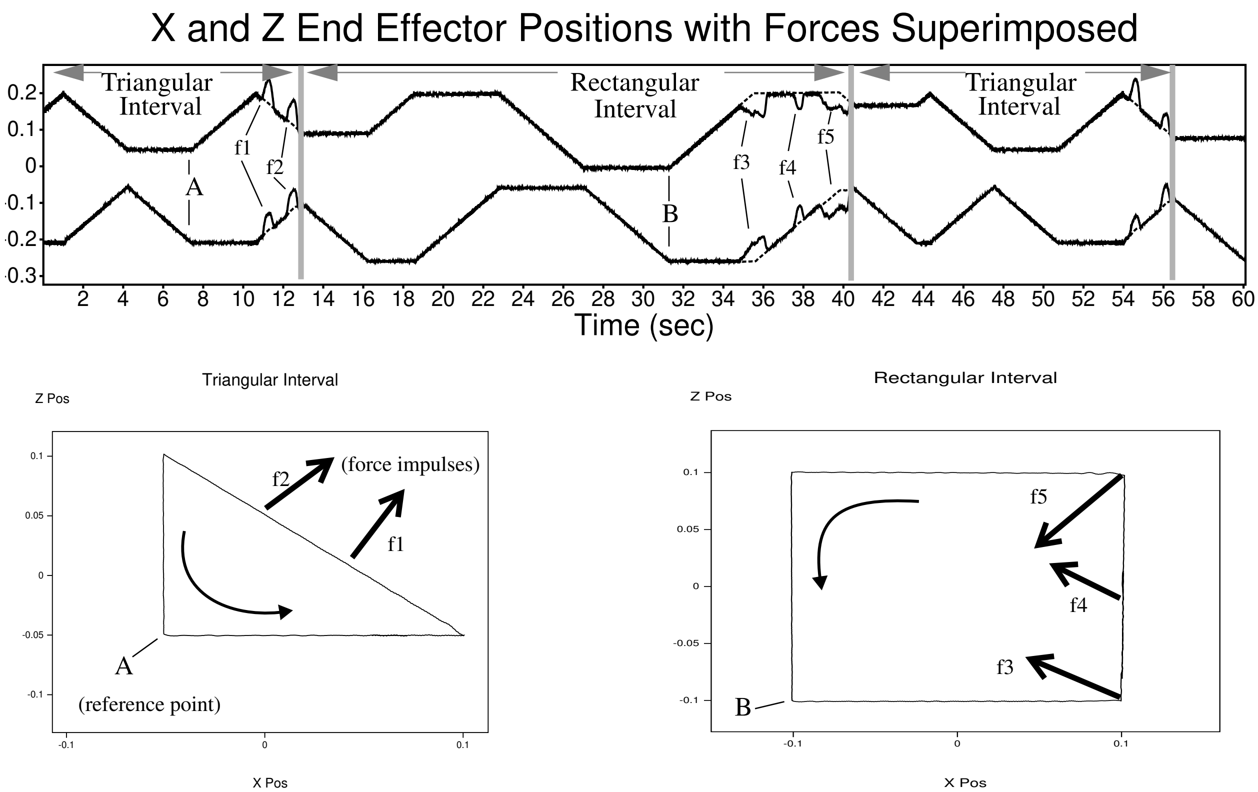


**Figure 6: Annotated temporal plot (top) of the robot's position in the plane superimposed with force impulses. Spatial plots (bottom) show trajectory intervals in the plane with corresponding annotations.**

For this application, 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 [15], also developed at Carnegie Mellon. (A command-line interface is also provided for "purists.") This allows the easy testing of agents alone and in variable configurations. Most of the communication between agents (minimal as it is) 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. Several types of point-to-point messages can be simulated through either private state variables or Chimera's built-in message passing capability.

With the initial set of agents described in section 4, the system works quite well. Figure 6 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.

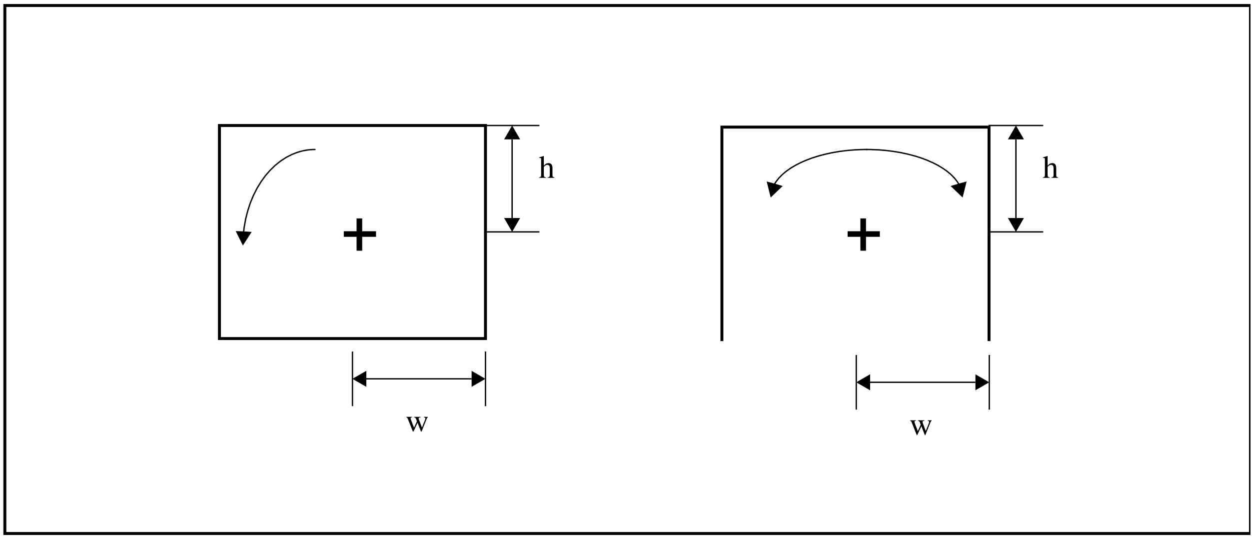


**Figure 7: Similarity of the rectangle and pick-and-place trajectories.**

In the first phase of the strip chart in Figure 6 (0 to 13 seconds), the robot is executing a triangular trajectory which is depicted spatially in the bottom left box. Impulses 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 impulses 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 force impulses, respectively. On average, it takes about four impulses 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 impulses to induce.

## 6 Extensions

To make the system more "practical" and to further investigate the interactions of agents, we added a pick-and-place trajectory. The *pick* trajectory was selected for its resemblance to an industrial operation and because it is nearly identical to the rectangular trajectory. (See Figure 7.) This close match increases the chance for adverse interactions and provides a "stress test" for the agent formulation.

The implementation of *pick* is very similar to *rectangle* so the details will not be described. Suffice it to say, a similar model with appropriate modifications allowed pick to work fine in relative isolation. Predictably, there were adverse interactions with the rectangle agent, due to its simplistic internal model, but this was cured by extending the rectangle model to include additional inhibitory features.

The problem, here, is the added inhibitory features were guided by the known adverse interactions with *pick*, so it would seem that knowledge of the pick-and-place trajectory was coded into the model of the rectangle agent. In fact, this is not true, but it points out the classic problem with multi-agent systems. The interactions with *pick* only guided the model development, suggesting which elements of the world should be included in the simple model. But multi-agent research has not yet produced an effective method of predicting or even analyzing agent interactions. Thus far, the greatest progress along these lines seems to be in the field of neural networks [16].

Surprisingly, the greatest interactions came with the enabling of the height and width agents. The interactions were more prominent because these agents include heuristics that bypass the normal arbitration channels and modify parameters immediately. To combat this problem an entirely different agent was developed to act as another virtual sensor. What this agent senses is the "confusion" of the shape agents. It calculates a measure of their indecision based only on the value and rate of change of the runner-up confidence. As such, it requires no knowledge of the particular agents that are executing, nor their number. Also, the failure of this agent or the elimination of it from the network results in graceful degradation.

The width and height agents were modified to use the complement of the confusion measure as an inhibitory stimulus. The lower the confusion in the trajectory, the less likely the width or height will change. This cured the interaction problem with pick (and improved performance with cross) by proprioceptive sense of the network's own stability.

## 7 Related Approaches

There is no other work that we are aware of that directly addresses this application 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 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. 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 joyball 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 [17] 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.

The work of Hirai and Sato [18] 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” [19]. 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 for modifying and fine-tuning robot trajectories was successfully implemented. With a trained operator, the perception system 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 impulses. Simple changes are generally possible to achieve with only two or three impulses as evidenced in figure 6. This demonstrates, at least for some subset of tasks, that multi-agent networks can be successful high-level perceptual systems as well as low-level behavioral systems.

In fact, due to the ease with which new features can be added (in the form of new agents) and the flexibility in configuration that the multi-agent system allows, we feel it's the best approach for this particular task and hardware configuration. We tried applying several more-traditional vision-based algorithms to the force inputs such as snakes and even discretizing the trajectory into a linear or planar "image," but the multi-agent approach proved the most satisfactory.

This area of perceiving intentions is rather interesting in itself. It also seems to lend itself well to multi-agent or "behavioral" solutions. We think the reason for this is the qualitative nature of the models. Qualitative models and heuristics seem easier to decompose into "behavioral-type" agents than quantitative models (unless the quantitative models are strongly orthogonal by some definition).

Finally, some mention should be made of the applicability of this approach to the real world. Few large, "OSHA-sensitive" companies allow their operators to physically interact with robots due to potential liability. Although our system currently allows the operator to interact remotely via a 6-DOF trackball and we are working on an interface for a DataGlove with Polhemus, it is not intended for such high-volume operations as a means of user input. Instead, it is aimed at small companies with job-shop missions whose direct employees do not have rigid job descriptions. The ideal trainee is one whose job is "to get the job done." These types of environments are more receptive to such hands-on approaches with appropriate safeguards.

For more restrictive corporate environments, there is another mode of operation we are beginning to explore: self-calibration. We think we will be able to use the user interface without modification for a task unrelated to the user; the task of self-calibration. If the workspace of the robot is populated with a series of "soft find-posts" - spongy calibration posts that provide force impulses 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 Steve Shafer for the initial project suggestion and his input along the way to completion.

## 10 References

[1] Decker, K.S., "Distributed Problem-Solving Techniques: A Survey," *IEEE Transactions on Systems, Man, and Cybernetics*, v.SMC-17, n.5, Sept./Oct. 1987, pp729-740.

[2] Durfee, E.H., V.R. Lesser and D.D. Corkill, "Trends in Cooperative Distributed Problem Solving," *IEEE Transactions on Knowledge and Data Engineering*, v.1, n.1, March 1989, pp. 63-83.

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

[5] Matsui, T., T. Omata, and Y. Kuniyoshi, "Multi-Agent Architecture for Controlling a Multi-Fingered Robot," in *Proceedings of the 1992 IEEE International Conference on Intelligent Robots and Systems*, pp. 182-186, July, 1992.

[6] Raibert, M.H., *Legged Robots that Balance*, MIT Press, Cambridge, MA, 1986.

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

[8] Lesser, V.R. and L.D. Erman, "A Retrospective View of the Hearsay-II Architecture," in *Proceedings of the International Joint Conference on Artificial Intelligence*, Cambridge, MA, pp. 790-800, 1977.

[9] Hebb, D.O., "The Organization of Behavior; a Neuropsychological Theory," New York, Wiley, 1949.

[10]Pearson, R.G., "Judgment of Volume from Two-Dimensional Representations of Complex, Irregular Shapes," Ph.D. Thesis, Department of Psychology, Carnegie Mellon University, Oct. 1961.

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

[12]Khatib, O., "The Operational Space Formulation in Robot Manipulator Control," in *Proceedings of the 15th International Symposium on Industrial Robots*, vol. 1, pp. 165-172, September 1985.

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

[14]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, November/December 1992.

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

[16]Pomerleau, D.A., "Neural Network Perception for Mobile Robot Guidance," Ph.D. Thesis, Computer Science Department, Carnegie Mellon University, CMU-CS-92-115, February 1992.

[17]Pook, P.K. and D.H. Ballard, "Recognizing Teleoperated Manipulations," in *Proceedings of the 1993 IEEE International Conference on Robotics and Automation, v. 2*, May, 1993, pp. 578-585.

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

[19]Kass, M., A. Witkin, and D. Terzopolous, "Snakes: Active Contour Models," *International Journal of Computer Vision*, v.1, n.4, 1987, pp. 321-331.