Robot Active Touch Exploration: Constraints and Strategies

Kenneth S. Roberts

Department of Computer Science
Columbia University
New York, New York 10027
roberts@cs.columbia.edu

Technical Report CUCS-480-89

Robot Active Touch Exploration:

Constraints and Strategies

Kenneth S. Roberts[1]

Department of Computer Science
450 Computer Science Building
Columbia University
New York, New York 10027
roberts@cs.columbia.edu

Technical Report CUCS-480-89
June 1989

Abstract

We investigate the problem of using active touch ("haptic") exploration to recognize a 3D object taken from a known set of models. What is new is that we combine two approaches: (1) using geometric constraints between components to eliminate interpretations, and interpretation tree methods for choosing the best active sensing move; (2) exploratory moves made by tracing continually along the surface of the object (and not through free space). We restrict ourselves to polyhedra, and give a set of geometric constraints tailored for matching components acquired from haptic exploration against components in the models. We present a new constraint using pairs of line segments. We then give a set of active sensing moves, each with an associated cost measure, and our strategies for choosing the next move.

1 Introduction

People usually find it easy to explore an object with their fingers and then identify it, even if they cannot see it. Robot tactile probes and multi-finger mechanical hands have also been applied to recognize objects. Such active touch exploration, using both external tactile sensors and internal position and force sensors, is called "haptic perception". The problem is to choose the active exploratory moves, and

[1]The author is an AT&T Bell Laboratories PhD Scholar. This work was also supported in part by DARPA contract N00039-84-C-0165, NSF grants DMC-86-05065, DCI-86-08845, CCR-86-12709, IRI-86-57151, North American Philips Laboratories, and the AT&T Foundation.

utilize the resulting sensory data in the best way to recognize and localize the object quickly and reliably.

The object is drawn at random from a set of fully known models of rigid, non-articulated objects. It is placed in a random pose (where "pose" refers to both position and orientation) in the workspace. In robot experiments, the object is usually held fixed, while in human experiments the explorer may move it. The two usual performance measures are percentage of successful identification, and quickness of identification.

We are working to construct an autonomous robot system to perform this task. We are using a Utah/MIT hand, with 4 fingers of 4 joints each, attached to a PUMA 560 6 degree of freedom robot arm [Allen *et al.*, 1989]. We have attached Bell Labs / Interlink tactile sensors to the fingertips, and have performed shape recovery [Allen and Roberts, 1989] and haptic exploratory procedures [Allen and Michelman, 1989]. This paper describes work on another aspect of active touch perception system: matching against a database of object models, and planning the next exploratory move.

1.1 Previous work

Previous research has been along several lines: First, psychologists have studied human haptic perception, with these main concerns: level of performance (especially haptic vs. vision, and active vs. passive touch), and choice of exploratory procedure [Gibson, 1966] [Klatzky *et al.*, 1985] [Klatzky and Lederman, 1986].

Some roboticists have built systems based on an awareness of the human haptic approach: tracing along the surface of the object, choosing an exploratory procedure tailored to the type of feature or property being acquired [Bajcsy, 1985] [Allen, 1987] [Stansfield, 1986] [Stansfield, 1987][Bay, 1989]. Other work in robot haptic recognition has used a variety of approaches [Fearing, 1988] [Dixon *et al.*, 1979] [Ivancevic, 1974] [Briot *et al.*, 1978] [Marik, 1981] [Okada and Tsuchiya, 1977] [Stojilkovic and Saletic, 1975] [Kinoshita *et al.*, 1975].

Several researchers have considered the question of how to apply geometric constraints between pairs of components to find a unique interpretation, and also to choose the optimal sensing move [Gaston, 1983][Gaston and Lozano-Perez, 1984] [Grimson and Lozano-Perez, 1984] [Schneiter, 1986] [Grimson, 1986] [Ellis, 1987] The paradigm here is that the sensor moves through free space until it contacts the polyhedral object, where it acquires the position and surface normal within some error limits. It is then removed from the object and can make another move-through-free-space-until-contact move. The cost of sensing is measured as the number of such moves it takes in order to recognize and localize the object. Geometric constraints among planar surfaces are used to eliminate candidate interpretations. The problem

is to find the free-space path for the next move to maximize the expected number of eliminated interpretations.

Our scheme complements these two previous approaches. To the systems using exploratory procedures to trace along the surfaces, it adds a scheme of geometric constraints, and a method to find the optimal next move. To the work on optimal free-space moves, it adds optimal surface-tracing moves. The goal is a recognition system that will use both free-space and surface-tracing moves, and choose optimally among them all.

2 Matching against models

2.1 Problem Statement

We are given a set of polyhedral model objects, each with vertexes $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \ldots$, edges $E_{0,1}, E_{1,3}, \ldots$, and faces $F_{(2,5,7)}, F_{(7,5,3,4)}, \ldots$. The vertexes, faces, and edges are together called the "components" of the object. A model object need not be convex, or even of genus 0, but the set of faces is assumed to form a single closed, connected set in 3-space.

One of the models is randomly chosen as the object which is to be recognized and localized. It is transformed by a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$ which are chosen randomly, and placed in the global coordinate frame of the workspace. Any vector $\mathbf{w}_d$ in the data object in the workspace is related to the corresponding model vector $\mathbf{w}_m$ by $\mathbf{w}_d = \mathbf{R}\mathbf{w}_m + \mathbf{t}$.

The problem is to find a pairwise correspondence between the components of the data object, and a set of model components which are all from the same object; and also to recover the pose given by $\mathbf{R}$ and $\mathbf{t}$.

2.2 Representation

Our method for representing a 3D object can be described as a "winged-edge" scheme [Baumgart, 1972], i.e. a graph whose nodes are face, edge, and vertex components, and whose arcs are adjacency relations. There could also be a hierarchy of object parts built on top of it, though we have not yet given any of our objects any hierarchical structure.

A face is a plane segment, which we represent by $(\mathbf{n}, d)$, where $\mathbf{n}$ is the outward-pointing surface normal, and d is the signed distance from the origin, so that the plane is given by $\{\mathbf{w} \in \mathsf{R}^3 \,|\, \mathbf{n} \cdot \mathbf{w} - d = 0\}$. There are also lists of pointers to adjacent edges and vertexes.

An edge is a line segment, represented by (b, p, l), where p is the position of the source vertex $\mathbf{v}_A$, and b is the direction, pointing toward the destination vertex $\mathbf{v}_B$, and l is the length. So the edge is given by $\{\mathbf{w} \in \mathsf{R}^3 \mid \mathbf{w} = \mathbf{p}+s\mathbf{b}, \quad 0 \leq s \leq l\}$. There are also pointers to the two endpoint vertexes, and pointers to the two adjacent faces, which are distinguished as left and right, relative to the sense of b, as viewed from outside the object.

A vertex is simply a position v, together with pointers to adjacent edges.

2.3 Procedure for matching

We assume that our module for executing sensor moves can extract face, edge, and vertex components from the data, and report them to our "matcher" module. For each newly recovered data component, the recovered parameters are reported together with error bounds, and also any known adjacency relations between this and previously reported data components. For an edge, the reported length l is that of the interval actually explored, and the position p is an endpoint of that interval, and not in general the position of a vertex.

The matcher maintains a set of interpretations which are consistent with the data so far. An *interpretation* is a correspondence between the data components acquired so far, and components which are all from the the same model. We can represent it as a sequence of pairs: $\{(D_1 : E^{[7]}_{2,3}), (D_2 : \mathbf{v}^{[7]}_2)(D_3 : E^{[7]}_{1,2})\}$, a matching between model 7 and the data components D_1, D_2, D_3.

All the interpretations may be arranged into a "tree" data structure [Grimson and Lozano-Perez, 1984]. The root of the tree is the null sequence. The children of the root are the model components which can be paired with D_1, the first object component acquired. As each new object component is acquired, a new level of children can be added to the interpretation tree for that model. The new child inherits the sequence of pairs from its parent, and augments that with a pairing of the next new data component with a model component. The new child must satisfy the constraints on matching given below. If an interpretation fails to generate any valid children (or has all its children pruned away), then it is pruned from the tree. The recognition process succeeds when only one interpretation remains (or all the interpretations are from one model object, and are known to be indistinguishable).

The matching procedure is to visit all the leaf nodes on the interpretation tree, and see which child data-model pairings should be generated. At each leaf node, the procedure is:

1. Use *adjacency* relations to generate candidate model components. Go through the list of data components reported as adjacent to the new one. For each one, find the corresponding model component in this interpretation (i.e. among

the ancestors of this leaf node). Each such model component then checks through the adjacency relations in its model object and gets the set of adjacent components whose type (face, edge, vertex) matches that of the new data component. The intersection of those sets is the initial candidate set of model components.

2. Test the candidate model components for matching of *intrinsic parameters* and properties with the new data component. Currently the only intrinsic parameter is the length of an edge, and we require $l_d < l_m + \epsilon$, where l_d is the length of the data edge as known so far. l_m is the length of the model edge, and ϵ is derived from data error bounds. In further work, intrinsic parameters might include convex vs. concave vs. planar surface, texture quality, etc.

3. Test *geometric relations* between components that are adjacent, or are adjacent to the same face. These tests are described in the next section. For a face, we take each pairing with a data face known to be adjacent to this one (sharing a common edge), and apply the face pair test. For an edge, we take each pairing with a data edge such that both edges are adjacent to a common data face, and do the edge pair tests. There is no special test for a vertex, but detection of a vertex tightens up the edge pair tests by constraining the position of the edge.

Each model component which passes all those tests is paired with the new data component to form a new node in the interpretation tree, and made a child of the current leaf node.

2.4 Geometric constraints

For a pair of adjacent faces, we use only one constraint, the angle between the surface normals [Grimson and Lozano-Perez, 1984]:

$$|\arccos(\mathbf{n}_{d1} \cdot \mathbf{n}_{d2}) - \arccos(\mathbf{n}_{m1} \cdot \mathbf{n}_{m2})| < \epsilon$$

where $\mathbf{n}_{d1}$ and $\mathbf{n}_{d2}$ are data normals, and $\mathbf{n}_{m1}$ and $\mathbf{n}_{m2}$ are the normals from the corresponding model components, and ϵ is derived from data error bounds.

For a pair of edges which are adjacent to the same face, we have several tests. But first we temporarily modify the sign of some of the edge directions so that they are all consistent with each other. We require that each edge direction $\mathbf{b}$ point clockwise around the common face, as viewed from outside the object. If the common face is the *left* adjacent face in the representation for the edge, then the edge direction must be reversed. (Whether an face is left or right can be derived from the sign of the determinant formed from face normal, edge direction, and relative position of face and edge.) We also move the edge position vector $\mathbf{p}$ to the other endpoint,

so that the line segment is still in the b direction from it (thus $\mathbf{p} := \mathbf{p} + l\mathbf{b}$ and $\mathbf{b} := -\mathbf{b}$).

Suppose we are given two data edges, $(\mathbf{b}_{d1}, \mathbf{p}_{d1}, l_{d1})$ and $(\mathbf{b}_{d2}, \mathbf{p}_{d2}, l_{d2})$, with corresponding model edges $(\mathbf{b}_{m1}, \mathbf{p}_{m1}, l_{m1})$ and $(\mathbf{b}_{m2}, \mathbf{p}_{m2}, l_{m2})$. The edge angle constraint is:

$$|\arccos(\mathbf{b}_{d1} \cdot \mathbf{b}_{d2}) - \arccos(\mathbf{b}_{m1} \cdot \mathbf{b}_{m2})| < \epsilon$$

where ϵ is derived from error bounds.

The next test is new, and powerful in eliminating interpretations. Let

$$\mathbf{d}_d = \mathbf{p}_{d2} - \mathbf{p}_{d1} \qquad \mathbf{d}_m = \mathbf{p}_{m2} - \mathbf{p}_{m1}$$

Then compute the distance from each edge position p to the point of nearest approach of the line of this edge to the line of the other edge in its pair (see Figure 1). If the lines are actually coplanar, then they intersect, and we could simply solve for this single intersection point. But given data and numerical accuracy considerations, it is better to use a robust method which can handle non-intersection:

$$s_{d1} = \frac{\mathbf{d}_d \cdot \mathbf{b}_{d1} - (\mathbf{b}_{d1} \cdot \mathbf{b}_{d2})(\mathbf{d}_d \cdot \mathbf{b}_{d2})}{1 - (\mathbf{b}_{d1} \cdot \mathbf{b}_{d2})^2}$$

with corresponding calculations for s_{d2}, s_{m1}, and s_{m2}. (This result can be derived by observing that the line segment connecting the two points of closest approach must be perpendicular to each of the two lines). Now the constraints are:

$$s_{m1} - l_{m1} - \epsilon < s_{d1} - l_{d1} \qquad \text{and} \qquad s_{d1} < s_{m1} + \epsilon$$

with corresponding constraints for s_{d2}. This means simply that no position in the known portion of the data edge may lie outside the endpoints of its model edge, as the two are laid out on the line from their intersection points with the other edge in the pair (see Figure 1) within data error bounds. (If a vertex endpoint for the new data edge is known, then these constraints can be tighter, since the distance from the intersection to that point is known exactly in the model, so there is no longer an interval, except for that provided by the error bound ϵ.)

We also check for the special cases in which the edges are parallel ($\arccos(\mathbf{b}_{m1} \cdot \mathbf{b}_{m2}) < \epsilon$) or anti-parallel ($\pi - \arccos(\mathbf{b}_{m1} \cdot \mathbf{b}_{m2}) < \epsilon$). Here we first test the distance between the parallel lines:

$$|\,\|\mathbf{d}_d - (\mathbf{d}_d \cdot \mathbf{b}_d)\mathbf{b}_d\| - \|\mathbf{d}_m - (\mathbf{d}_m \cdot \mathbf{b}_m)\mathbf{b}_m\|\,|$$

We also require that the relative positions of the known intervals two data edges be possible according to the relative position of the model edges, within data error bounds. If the edges are anti-parallel (the much more likely case, given the clockwise consistency requirement), the constraints are:

$$\mathbf{d}_m \cdot \mathbf{b}_m - (l_{m1} + l_{m2}) - \epsilon < \mathbf{d}_d \cdot \mathbf{b}_d - (l_{d1} + l_{d2})$$

$$\mathbf{d}_d \cdot \mathbf{b}_d < \mathbf{d}_m \cdot \mathbf{b}_m + \epsilon$$

where $\mathbf{b}_d = (\mathbf{b}_{d1} - \mathbf{b}_{d2})/\|\mathbf{b}_{d1} - \mathbf{b}_{d2}\|$.

2.5 Simulation results

We have verified this overall matching procedure and the geometric constraints, by implementing them in a system which takes input from a simulated sensor move execution module. The data components are generated by selecting components from the correct model, rotating and translating them, and then introducing small errors by hand, ad hoc.

Some of the object models are shown in Figure 2. All the constraints described in this section were implemented and tested on those objects. The system was able successfully discriminate each of them from short sequences of data components: 3 or 4 faces and 2 or 3 edges.

Figures 5and 6show a test run using a database of only two models, BOX01 and WEDGE01. After the database of models is loaded, the first sequence of data components is given. The matcher builds an interpretation tree, which shows 30 interpretations which are consistent with the 5 data components. Then another 2 data components are given, and now the matcher is able to prune away all the interpretations but one for WEDGE01.

Of course, for objects with symmetry, such as the boxes, a unique interpretation of the pose is not possible. BOX01 has 4 possible poses.

BOX01N is non-convex, and is successfully distinguished from BOX01P.

An interesting result is that the system can successfully distinguish between mirror pairs, such as the tetrahedra in Figure 3. When given data from the TETRA01L, the system will reject TETRA01R.

2.6 Discussion

We have tried to choose and tailor the constraints particularly for this task of recognition by surface tracing, with its sparse, concentrated, and connected data. The primary place of adjacency constraints follows from the connectedness of the data.

Our emphasis on edge constraints comes from several considerations:

- An edge can be acquired by an active touch sensing system in several ways: (1) edge detection "vision" processing of tactile array data; (2) sudden position, velocity, and force changes; (3) intersection of faces detected by the segmentation procedure.
- Without vision, it is not easy to reliably get the position and size of a face. How do we know when we have done enough exploration of a data face in order to justify rejecting a certain model face on the grounds that its diameter is too large?

- Edges give more precise position information, and therefore produce tighter positional constraints than faces. (Assuming orientation is known and a single position on the component, a face has two remaining positional degrees of freedom, an edge only one).

- A pair of edges can define a complete coordinate frame transform, position and orientation. When the direction signs are forced to be consistent, then the rotational frame is unique, which is why the mirror tetrahedra can be distinguished.

The key insight in getting edge pair constraints to work was in seeing both the need and the method for forcing consistent direction senses between the data and model pair. This is done by appealing to a known common adjacent face.

Although the geometric constraints were developed for adjacent and near-adjacent components, they easily apply to the non-adjacent cases. The face angle constraint obviously does not depend on adjacency, since the face normals must have consistent senses because of the requirement that they point physically outward. For the edge pair constraints, the way to solve the sense ambiguity problems is to require *two* known adjacent faces, one for each edge in the pair. Then a consistent clockwise sense can be enforced on all the edges, and all the constraints apply as before.

One difference is that the two edges will in general be skew (i.e. their lines do not intersect). But this is no problem for the constraints already given above, since those make no assumption of intersection. And this non-intersection adds an additional constraint which can be tested: The data and model pairs must agree on the distance between the lines at their points of closest approach, within data error bounds.

$$\left| \frac{\mathbf{b}_{d1} \times \mathbf{b}_{d2}}{\|\mathbf{b}_{d1} \times \mathbf{b}_{d2}\|} \cdot (\mathbf{p}_{d2} - \mathbf{p}_{d1}) - \frac{\mathbf{b}_{m1} \times \mathbf{b}_{m2}}{\|\mathbf{b}_{m1} \times \mathbf{b}_{m2}\|} \cdot (\mathbf{p}_{m2} - \mathbf{p}_{m1}) \right| < \epsilon$$

3 Strategies for choosing moves

A very interesting question in active touch perception is how to choose the next exploratory move. Our objective is to choose the move which can be expected to eliminate the most interpretations with the least cost of exploration. Since there are multiple infinities of physically possible paths for an active touch sensor, we must have some approach which reduces them to a class which we can handle. Our scheme is to consider only a small class of explorative "primitives" (where each primitive is a motion path together with a termination condition). And there will be only three sequences of these primitives which are admissible as a "move" for purposes of evaluating cost and benefit for planning.

3.1 How much does a move cost?

Previous work on free-space strategies assumed that each move-through-free-space-until-contact move cost an identical amount. This was reasonable, since there was only one kind of move, and there was no simple way to know how much it cost to get from one path to the next (considering robot inverse kinematics, collision avoidance, etc). But for surface tracing moves, we know the path because part of the problem is to choose that path; so we do have a more refined measure which is reasonable: the distance travelled along the path. We can multiply the distance by a difficulty factor, since we might find, for instance, that more care must be used in following an edge than in tracing a straight path across a plane. For other actions, we can assign a constant value, such as for moving just far enough to acquire the surface normal of a plane.

Here are some sensing and movement primitives which we want to use (with their associated cost in parentheses). s refers to the distance travelled, and the c values are constants.

- trace a path in a chosen direction along a face ($c_f\, s$).
- trace along an edge in a chosen sense ($c_e\, s$).
- trace a circular path in a chosen sense around a vertex ($c_v\, s$).
- while moving on a face, detect contact with an edge (c_j).
- while moving on an edge, detect contact with a vertex (c_k).
- move on a face just far enough to acquire an estimate of its surface normal (c_n). This might call for tracing a non-straight path.
- move on an edge just far enough to acquire an estimate of its direction (c_b).

4 Admissible moves for planning

Rather than consider every possible primitive at every decision point, we will restrict our planner to certain sequences of primitives, to limit the searching. We will call these admissible sequences simply "moves". Here are the three moves, together with the new data each is expected to acquire, and the costs each is expected to incur.

1. Face move. Beginning on or near a face, trace across the face in a chosen direction until reach an edge. Move just far enough on the edge to acquire its direction. Move just far enough out onto the face beyond this edge to acquire its normal, and then end up near that edge. Assuming that we had

already acquired the face which the sensor started on, this move should yield a new edge (and its direction) and a new face (and its normal). Its cost is $c_f s + c_j + c_b + c_n$. (See Figure 4a.)

2. Edge move. Beginning on or near an edge, move along it in a chosen sense until a vertex is reached. This yields a new vertex for the edge. Its cost is $c_e s + c_k$. (See Figure 4b.)

3. Vertex move. Beginning near a vertex and near a known edge, move on a circular path around the vertex in a chosen sense until the next edge is reached. Move just far along the edge to acquire its direction. Then move onto the next face just far enough to get an estimate of its normal, and then end up near the new edge and still near the vertex. This yields a new edge (and its direction) which is not parallel to the starting edge, and a new face (and its normal). Its cost is $c_v s + c_j + c_b + c_n$. (See Figure 4c.)

The guiding ideas for constructing these moves are that after moving across a face to reach an edge, it does not cost that much more to acquire its direction; and after that it does not cost much more to acquire the normal of the (planar) face that lies on the other side of the edge (i.e. c_b and c_n are small relative to $c_f s$). Also, it is good to "stop" and plan the next move while the sensor is near an edge or near a vertex, since more options are available there — which is why each of the three sequences ends with one of those conditions obtaining.

4.1 Strategies for each kind of move

To choose the next move, we first find out which of these conditions are true: near a face, near an edge, near a vertex. ("near" means simply that the distance from the current sensor position to the closest point on a data component of the specified type is less than some arbitrary amount. "Near a face" is always true by definition of a polyhedron, and "near an edge" is always true when "near a vertex" is true.) These are the pre-conditions of the face, edge, and vertex moves, respectively. The candidate moves are the ones whose pre-conditions are satisfied.

For each interpretation, we will have a known model-to-data coordinate frame transform. This is because prior moves were chosen to guarantee that the interpretation would contain a non-parallel edge pair. (See further below on how to handle the initial problem position). As noted above in the section on geometric constraints, a correspondence of data and model non-parallel edge pairs implies a unique coordinate frame transform for the object pose. Actually, the explicit rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ are not needed. This is because all that matters for purposes of matching and planning are the relations between the sensor position and the nearby edges and faces. These can easily be derived from the edge pair, if two values

are remembered from the edge pair constraint test in the matching procedure, and the move execution module keeps track of the relation between the current sensor position and the data position of the most recently contacted edge.

Since the model-to-data transform is known (implicitly) for each interpretation, we can predict the results of each candidate move in the hypothetical world of that interpretation. E.g., "If interpretation Q312 is true, then if we move along edge $E_{2,5}$, we will reach a vertex after travelling a distance of 2.3 plus or minus the data error bounds". So for each candidate move, we can assemble all the predicted outcomes, one from each interpretation in the tree. The results from the actual move will be consistent with only some of the predictions, and the other interpretations will be eliminated.

The "efficiency" of a candidate move is the expected number of interpretations eliminated per unit of move cost. (The details of how to do this for each move type are described further below.) The move with the highest efficiency is chosen as the best move, and is then executed.

A special case is the initial problem position, where no transform is known which can be used for the prediction of move outcomes. The way out of this is not to do any planning in the initial situation. Instead, we can execute a fixed sequence of initial moves, which will guarantee that when it is completed, and the matching is done on the resulting data, each valid interpretation will have a known transform. One such sequence is this: Beginning with the sensor in a general position on the interior of a face, execute a face move, then an edge move, and then a vertex move. This will yield a non-parallel edge pair and a vertex, and three faces, which are more than sufficient to determine the transform.

4.1.1 Efficiency of an edge move

If the sensor is on an edge, then it has two candidate edge moves, one in each directional sense (unless it is already at one of the vertex endpoints, in which case it has only one direction available). Consider an edge move in a direction toward a vertex which is not yet known. Suppose there are five interpretations, and their predicted distances to the vertex are $1.0, 2.8, 3.0, 3.2, 5.0$. If the data error bound for distance along an edge is ± 0.2, then the number of interpretations eliminated if the specified interpretation is true are $4, 3, 2, 3, 4$. (These results are not strictly correct, since the distance acquired from the data may be different from that predicted by the interpretation, due to error. So the number of interpretations actually eliminated might be greater or less than the amount given.) If $c_e = 2.0$ and $c_k = 0.4$, then the cost under each interpretation is $2.4, 6.0, 6.4, 6.8, 10.4$. The efficiency is the expected interpretations eliminated per expected unit cost: $1.67, 0.5, 0.31, 0.44, 0.38$, so the efficiency for this move is the average of the five, 0.66.

4.1.2 Efficiency of a vertex move

The efficiency of a vertex move is measured in basically the same way, except that edge angle is used to eliminate interpretations, and that distance is measured along a circular path. One additional wrinkle is that because the vertex move acquires a new face as well as a new edge, there is another constraint available to eliminate interpretations. This means that if two interpretations are not distinguishable by the edge angle constraint, we can check to see if they would be by the face angle constraint. If yes, we can increase the expected number of interpretations eliminated by one.

4.1.3 Efficiency of a face move

It is more complicated to evaluate the efficiency of face moves. First, there are an infinite number of them available (and that is only counting the ones with straight line paths). Second, there are three constraints available: edge pair angle, edge pair distance, and face pair angle. We simplify by considering only straight line paths, and only at discrete intervals in the domain of possible direction angles.

For each such candidate straight-line path, have each interpretation predict the distance until an edge is reached. Then calculate efficiency as for the edge move, except that when two distances cannot be distinguished, can also try to distinguish the two interpretations by edge angle, and then by face angle constraints.

4.2 Discussion

We have implemented the procedures for finding the admissible candidate moves, and for evaluating the efficiency of edge moves and choosing the best one; and have integrated these into our matching system. We have not yet implemented the efficiency measure for the vertex and face moves. So we have not yet demonstrated the system planning the entire sequence of sensor moves to recognize an object, though we moving quickly toward that goal.

5 Conclusion and further work

We have presented our work toward matching and planning modules for our robot active touch system. One thing we have not dealt with is the "multi-finger" aspect of our Utah/MIT hand. The matching part is essentially unchanged, but the planning gets far more complicated than for a single touch sensor. As a first step toward multi-finger strategy, we have implemented an algorithm for how to coordinate the multiple fingers and arm to reach simultaneously multiple specified locations, or

say if they are unreachable [Roberts, 1989]. The set of admissible moves, and the cost and efficiency measures presented in this paper should remain relevant to the multi-finger problem.

References

Allen, P. K. 1987. *Object Recognition Using Vision and Touch*, Kluwer.

Allen, P. K. and K. S. Roberts. 1989. "Haptic Object Recognition Using a Multi-fingered Dextrous Hand," *Intl Conf Robotics Automation*.

Allen, P. K. and P. Michelman. 1989. "Acquisition and Interpretation of 3D Sensor Data from Touch," *Workshop on Interpretation of 3D Scenes, Austin TX*.

Allen, P. K., P. Michelman, and K. S. Roberts. 1989. "An Integrated System for Dextrous Manipulation," *Intl Conf Robotics Automation*.

Bajcsy, R. 1985. Shape from touch. In *Advances in Automation and Robotics*, ed. G. Saridis, JAI Press.

Baumgart, B. G. 1972. "Winged-edge polyhedron representation," Stanford University T.R. AIM-179.

Bay, J. S. 1989. "Tactile Shape Sensing via Single- and Multi-Fingered Hands," *Intl Conf Robotics Automation*, pp. 290–295.

Briot, M., M. Renaud, and Z. Stojilkovic. 1978. "An approach to spatial pattern recognition of solid objects," *IEEE Systems, Man, and Cybernetics*, vol. SMC-8, pp. 690–694.

Dixon, J. K., S. Salazar, and J. R. Slagle. 1979. "Research on tactile sensors for an intelligent robot," *Proc. 9th ISIR*, pp. 507–518.

Ellis, R. E. 1987. *A Tactile Sensing Strategy for Model-based Object Recognition*, PhD thesis, U. Massachusetts.

Fearing, R. S. 1988. "Tactile Sensing for Dextrous Manipulation," *Workshop on Dextrous Robot Hands (ICRA Philadelphia, April 24)*.

Gaston, P. C. 1983. *Robotic Tactile Recognition*, S.M. thesis, MIT EECS dept..

Gaston, P. C. and T. Lozano-Perez. 1984. "Tactile Recognition and Localization Using Object Models: The Case of Polyhedra on a Plane," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 3, pp. 257–266.

Gibson, J. J. 1966. *The Senses Considered as Perceptual Systems*, Houghton Mifflin, Boston.

Grimson, W. E. L. and T. Lozano-Perez. 1984. "Model-based recognition and localization from sparse range or tactile data," *Int. J. Robotics Research*, vol. 3, no. 3, pp. 3–35.

Grimson, W. E. L. 1986. "Sensing Strategies for Disambiguating Among Multiple Objects in Known Poses," *IEEE J Robotics Automation*, vol. RA-2, no. 4, pp. 196–213.

Ivancevic, N. S. 1974. "Stereometric pattern recognition by artificial touch," *Pattern Recognition*, vol. 6, pp. 77–83.

Kinoshita, G., S. Aida, and M. Mori. 1975. "A pattern classification by dynamic tactile sense information processing," *Pattern Recognition*, vol. 7.

Klatzky, R. L., S. J. Lederman, and V. Metzger. 1985. "Identifying objects by touch: An "expert system"," *Perception & Psychophysics*, vol. 37, no. 4, pp. 299–302.

Klatzky, R. L. and S. J. Lederman. 1986. "Hand movements: a window into haptic object recognition," U. Calif Santa Barbara Cognitive Science Technical Report 8606.

Marik, V. 1981. "Algorithms of the complex tactile information processing," *Int. Joint Conf. Artificial Intelligence*, pp. 773–774.

Okada, T. and S. Tsuchiya. 1977. "Object Recognition by Grasping," *Pattern Recognition*, vol. 9, no. 3. pp. 111–119.

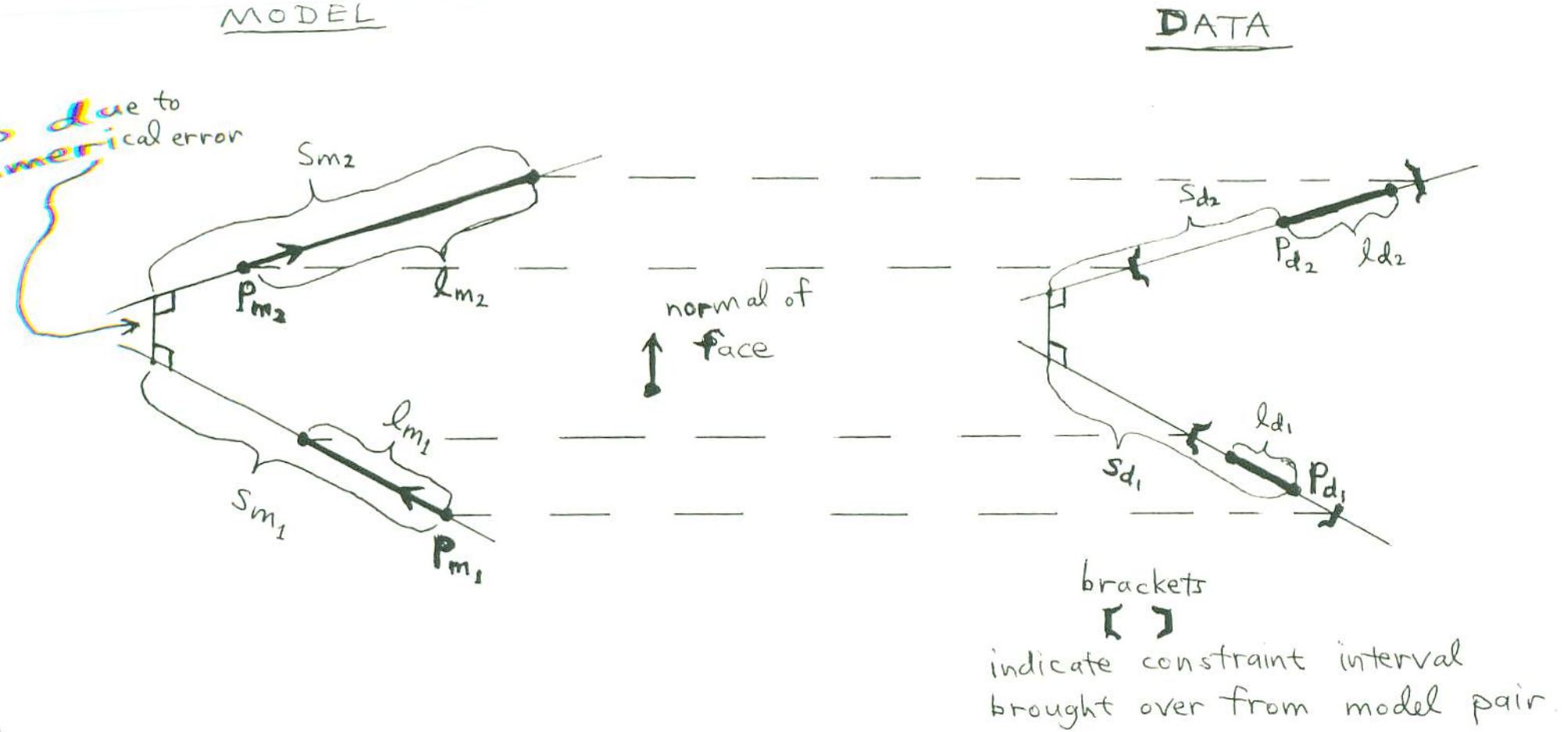
Roberts, K. S. 1989. "Coordinating a Robot Arm and Multi-finger Hand," *Columbia University Computer Science technical report CUCS-481-89.*

Schneiter, J. L. 1986. "An objective tactile sensing strategy for object recognition and localization," *Intl Conf Robotics Automation*, pp. 1262–1267.

Stansfield, S. A. 1986. "Primitives, features, and exploratory procedures: building a robot tactile perception system," *Intl Conf Robotics Automation*, pp. 1274–1279.

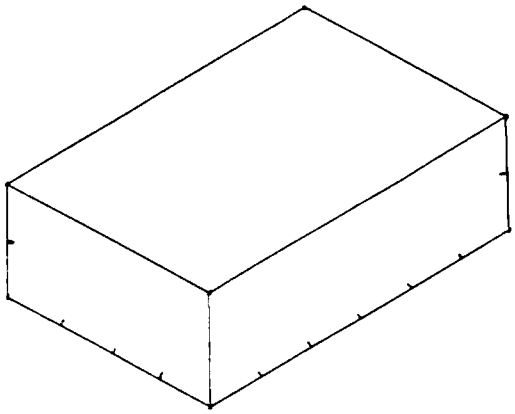
Stansfield, S. A. 1987. "Visually-aided tactile exploration," *Intl Conf Robotics Automation*, pp. 1487–1492.

Stojilkovic, Z. and D. Saletic. 1975. "Learning to recognize patterns by Belgrade hand prosthesis," *Proc. 5th ISIR*, pp. 407–413.

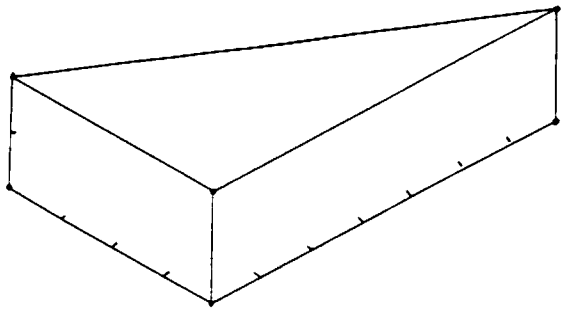


Edge pair constraints on position

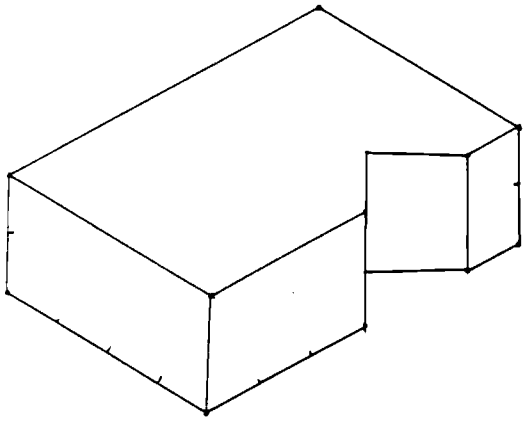
Figure 1



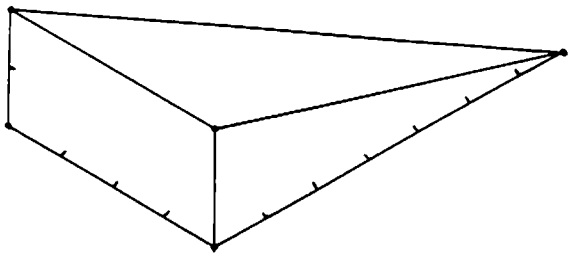
BOXØ1



WEDGEØ1

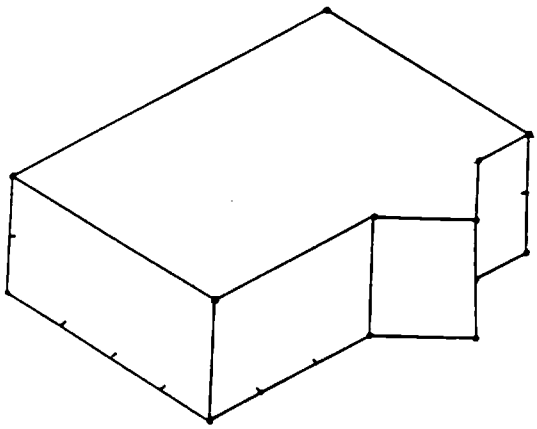


BOXØ1N

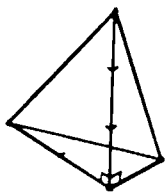


PYRØ1

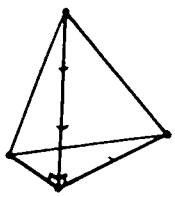
Figure 2



BOXØ1P



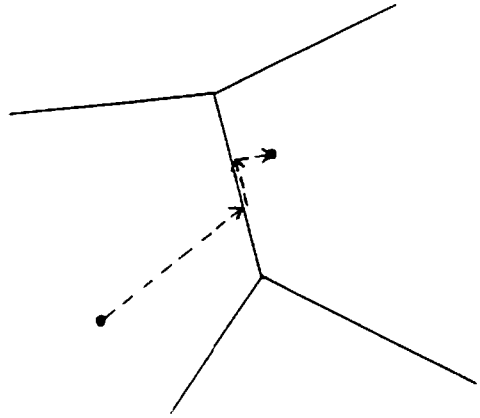
TETRA01L



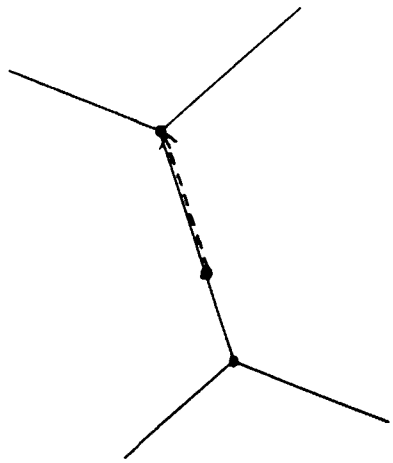
TETRA01R

Mirror tetrahedra

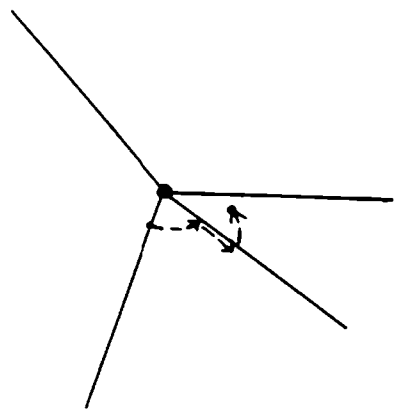
Figure 3



(a) Face move



(b) Edge move



(c) Vertex move

Figure 4

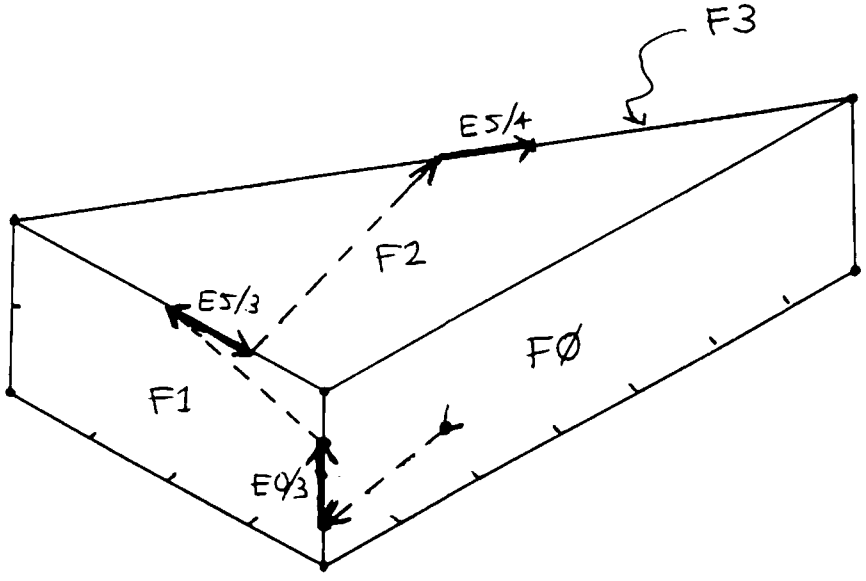


Figure 5:
Exploration Path for the test run in Figure 6.

```
;;; Loading source file "models01.lisp"
;;; Loading source file "objects/box01.lisp"
;;; Loading source file "objects/wedge01.lisp"
> object_list
(BOX01_OBJ WEDGE01_OBJ)
> (explore "data/wedge01_01")
;;; Loading source file "data/wedge01_01.lisp"

;;-------------------------------------------------------------
(next_data_set
  (
    (F0          ())
    (E0/3        (adjacent E0/3 F0))
    (F1          (adjacent F1 E0/3))
    (E5/3        (adjacent E5/3 F1))
    (F2          (adjacent F2 E5/3))
  ))

;; Build the Interpretation Tree, one node at a time.
(this_ITnode = NIL NIL) ---> t
  (this_ITnode = DATA02_F0 BOX01_F0_RIGHT) ---> t
    (this_ITnode = DATA02_E0/3 BOX01_E1/5) ---> t
      (this_ITnode = DATA02_F1 BOX01_F4_FRONT) ---> t
        (this_ITnode = DATA02_E5/3 BOX01_E4/5) ---> t
          (this_ITnode = DATA02_F2 BOX01_F2_TOP) ---> t
    (this_ITnode = DATA02_E0/3 BOX01_E5/6) ---> t
      (this_ITnode = DATA02_F1 BOX01_F2_TOP) ---> t
        (this_ITnode = DATA02_E5/3 BOX01_E6/7) ---> t
          (this_ITnode = DATA02_F2 BOX01_F1_BACK) ---> t

        ...

        ...

  (this_ITnode = DATA02_F0 WEDGE01_F4_FRONT) ---> t
    (this_ITnode = DATA02_E0/3 WEDGE01_E0/3) ---> t
      (this_ITnode = DATA02_F1 WEDGE01_F3_LEFT) ---> t
        (this_ITnode = DATA02_E5/3 WEDGE01_E3/5) ---> t
          (this_ITnode = DATA02_F2 WEDGE01_F2_TOP) ---> t
    (this_ITnode = DATA02_E0/3 WEDGE01_E3/4) ---> t
      (this_ITnode = DATA02_F1 WEDGE01_F2_TOP) ---> nil
    (this_ITnode = DATA02_E0/3 WEDGE01_E1/4) ---> nil
    (this_ITnode = DATA02_E0/3 WEDGE01_E0/1) ---> t
      (this_ITnode = DATA02_F1 WEDGE01_F0_BOTTOM) ---> t
        (this_ITnode = DATA02_E5/3 WEDGE01_E0/2) ---> t
          (this_ITnode = DATA02_F2 WEDGE01_F3_LEFT) ---> t

interpretation count = 30

Itree =
NIL
  BOX01_F0_RIGHT
    BOX01_E1/5
      BOX01_F4_FRONT
        BOX01_E4/5
          BOX01_F2_TOP
    BOX01_E5/6
      BOX01_F2_TOP
        BOX01_E6/7
          BOX01_F1_BACK

        ...

        ...

  WEDGE01_F4_FRONT
```

Figure 6:

Test run on database of BOX01, WEDGO1

```
    WEDGE01_E0/3
      WEDGE01_F3_LEFT
        WEDGE01_E3/5
          WEDGE01_F2_TOP
    WEDGE01_E0/1
      WEDGE01_F0_BOTTOM
        WEDGE01_E0/2
          WEDGE01_F3_LEFT

;;-------------------------------------------------------------
(next_data_set
  (
    (E5/4        (adjacent E5/4 F2))
    (F3          (adjacent F3 E5/4))
  ))

;; Build the Interpretation Tree, one node at a time.
(this_ITnode = DATA02_F2 BOX01_F2_TOP) ---> nil
(this_ITnode = DATA02_F2 BOX01_F1_BACK) ---> nil

        ...

        ...

(this_ITnode = DATA02_F2 WEDGE01_F4_FRONT) ---> nil
(this_ITnode = DATA02_F2 WEDGE01_F0_BOTTOM) ---> nil
(this_ITnode = DATA02_F2 WEDGE01_F2_TOP) ---> t
  (this_ITnode = DATA02_E5/4 WEDGE01_E4/5) ---> t
    (this_ITnode = DATA02_F3 WEDGE01_F1_BACK) ---> t
(this_ITnode = DATA02_F2 WEDGE01_F3_LEFT) ---> nil

interpretation count = 1

Itree =
NIL
  WEDGE01_F4_FRONT
    WEDGE01_E0/3
      WEDGE01_F3_LEFT
        WEDGE01_E3/5
          WEDGE01_F2_TOP
            WEDGE01_E4/5
              WEDGE01_F1_BACK
```

Figure 6, continued