Towards Geometric Decision Making in Unstructured Environments

Gregory D. Hager

Research Report YALEU/DCS/RR-960
May, 1993

This work is supported by DARPA grant N00014-91-J-1577, by National Science Foundation grants IRI-9109116 and DDM- 9112458, and by funds provided by Yale University.

Towards Geometric Decision Making in Unstructured Environments

Gregory D. Hager
Department of Computer Science
P.O. Box 2158 Yale Station
Yale University
New Haven, CT 06520

Abstract

This paper presents an approach to sensor-based decision making in unstructured environments that relies on describing geometric structures by parameterized volumes. This approach leads to large systems of nonlinear, stochastic inequalities. We describe how these inequalities can be solved using interval bisection, discuss the structural and statistical convergence of the technique, and present some preliminary experimental results.

1 Introduction

The increasing emphasis on adaptability and autonomy in robotics places high demands on the ability of a system to react to sensor data. This is particularly true when the system must operate in partially or totally unstructured environments. However, unstructured situations pose challenging problems in sensor data fusion and sensor-based decision making.

Solving a sensor data fusion or decision making problem involves stating a set of constraints that describe the relationship between problem inputs (sensor data) and desired solutions (model parameters, segmentations, decisions, and so forth), and developing a method for resolving the constraints when applied to real inputs. One of the difficulties in unstructured environments, particularly when geometric issues are involved, is finding a set of constraints that are physically plausible, solvable, and useful for the problem at hand.

Many sensor data modeling techniques *fit or match* one or more modeling primitives to the data and consequently rely on strong assumptions about the form and/or physical behavior of observed objects. There has been work in this area based on parametric patches [2, 3, 8, 9], generalized cylinders [15], superquadrics [17], deformable models [14, 18], parameterized model recognition [5, 7] and many other variations too numerous to mention here. By increasing the complexity of

[0]This is a revised and updated version of an article that will appear in the Proceedings of the 1991 IROS conference, Osaka, Japan, November 1991.

such models, it is presumably possible to model nearly anything to acceptable level of accuracy. However the complexity of the resulting description may rival that of the data itself! Moreover, more complex modeling schemes usually lead to more difficult problems in detecting and accommodating outliers in the data and attaching some meaning to fitting error.

Instead of modeling geometry using surfaces defined by equalities, we propose modeling *volumes* defined by *inequalities*. By using inequalities, it is possible to make correct, consistent decisions about observed surfaces that do not necessarily correspond to a specific geometric form.

For example, suppose that the task is to pack a variety of irregularly shaped objects into a confined space. One approach to this problem would be to fit some type of regular surface to the object, and to find a placement of these regular surfaces within the space. However, due to fitting errors it is quite possible that the chosen placement will not work with the actual objects. By changing the equalities describing the surfaces to inequalities, it is possible to compute *bounding* descriptions for the objects and find an arrangement of these bounding descriptions. In this case, the placement is guaranteed to work (modulo some assumptions about the accuracy and completeness of the observed data).

Solving sets of nonlinear inequalities is a difficult problem. Least squares, Bayesian estimation, or energy minimization—all popular methods for solving equality constraints—are often difficult to apply when nonlinear inequalities are involved. Recently, we have been investigating solving sensor data fusion problems using interval-based constraint solving methods [6]. The procedure we use is a combination of minimization and constraint solving. This method works directly on the problem constraints, and convergence can be demonstrated for an extremely wide variety of problems with very few statistical or structural assumptions. This approach has the further advantage of generating *sets* of solutions. Hence, it is possible to evaluate the robustness of actions taken based on the solution. For example, a small solution set for the placement of an object in a container suggests that the likelihood of collision when physically placing the object is relatively large.

The remainder of this paper describes some preliminary results on applying the constraint solving paradigm to the large systems of inequalities arising from sensor data fusion and decision making problems. The major points discussed are: the solution of systems of nonlinear constraints and the properties of constraints that contain random variables, formulating the constraints for determining a packing of a small number of objects, and the discussion of some preliminary experimental results for the packing problem. Readers interested in further experimental results of interval-based constraint solving are invited to consult [6].

2 Modeling with Constraints

In [6] we discuss a methodology for developing systems of constraints describing sensor data fusion problems. Briefly, we assume that data is observed with bounded error. We state two general classes of constraints: sensing-related constraints and task-related constraints. Sensing-related constraints describe the relationship between observations and an underlying physical or geometric, parametric model. These constraints are usually independent of the specific problem that is being solved.

Conversely, task-related constraints, as the name suggests, describe the conditions that must be satisfied to reach a decision or take an action for a particular problem.

For example, the definition of the packing problem employs sensor-related constraints that describe enclosing volumes for the objects themselves, and task-related constraints that describe containment and disjointness of all objects when packed. We formulate this problem using a superquadric representation of two-dimensional contours. Superquadrics have the advantage of allowing a range of sizes and shapes, although they can only accurately represent convex bodies. A superquadric in standard position is described by a vector $s = [a, b, \gamma]$ where a and b represent size along the x and y axes, and γ encodes shape. Let $l = [x, y, \theta]$ describe a location and orientation in the plane, $T(l, u)$ denote mapping of the point u under spatial transformation defined by l, and $T^{-1}(l, u)$ denote the inverse mapping. The functional form that describes a superquadric in the plane is:

$$\begin{aligned} g'([a,b,\gamma],[x,y]) &= \left(\left|\frac{x}{a}\right|^{2/\gamma} + \left|\frac{y}{b}\right|^{2/\gamma}\right)^{\gamma/2} \\ g(l;s,z) &= g'(s, T^{-1}(l,z)) \end{aligned}$$

If $g(l; s, z) \geq 1$ the data point lies outside the superquadric, and conversely $g(l; s, z) \leq 1$ indicates that the data point is within the region enclosed by the superquadric.

The packing problem requires parameters for n superquadrics describing objects, and n translations and orientations of the objects within the container. Let l and s denote locations and superquadric parameters as above, $m = [s, t, \alpha]$ denote a position within the container, and $p = [m_1, l_1, s_1, \ldots, m_n, l_n, s_n]$ denote the vector describing the entire system. Given sets of data vectors $O_1, O_2, \ldots, O_n$ from n objects and a set C of data vectors from a container, the system of constraints for object i is:

$$\begin{aligned} \forall z \in O_i, &\quad g(l_i; s_i, z) \leq 1 \\ \forall z \in C, &\quad g(m_i; s_i, z) \geq 1 \\ \forall j \neq i, \quad \forall z \in O_i, &\quad g(m_j; s_j, T(m_i, T^{-1}(l_i, z))) \geq 1 \end{aligned}$$

A satisfactory solution to a sensor data fusion problem requires satisfaction of *both* sensor-related and task-related constraints. In general, task-related constraints may be interpreted *universally*—the criteria must hold for *all* consistent models, or *existentially*—the criteria must hold for some consistent model. The proper interpretation depends on the criteria themselves, as well as the definition of what constitutes a consistent model for the data. For example, to grip an object, we must prove that no matter what the exact size of the object is, it is less than the gripper opening. If the geometric model is defined using equality, then any consistent model "fits" the data up to observation error. In this case, it must be proven that *all* possible models are smaller than the gripper opening. Conversely, if the geometric model is defined using an inequality so that it bounds the object, then it suffices to exhibit *some* model that bounds the data and is small enough to fit into the gripper. In this article, we will only be concerned with feasibility—that is, existential problems.

In the remainder of this article the vector Z represents the "giant" vector resulting from the conjunction of all data vectors z, and p of dimension s represents the collection of all problem parameters. With these conventions, by appropriate reindexing, renaming of variables, and algebraic manipulation we can rewrite any conjunction of constraints in the form $g^*(p, Z) \leq 0$ where $\leq$ is interpreted to apply componentwise. A parameter vector p is *consistent* with a data vector Z if and only if $g^*(p, Z) \leq 0$. If this condition does not hold, the parameter vector is *inconsistent*. The set of all consistent interpretations of a vector of data, Z, with respect to a system of constraints, g^*, is

$$\mathcal{P}^* = \{p \in \mathcal{P} \mid g^*(p, Z) \leq 0\}.$$

We refer to $\mathcal{P}^*$ as the *solution set* consistent with a series of observations. The problem is to determine if this set is nonempty, and if so to generate some portion of it.

3 Interval-Based Constraint Solving

There are several approaches to solving systems of equalities and inequalities [10]. Most techniques encode some or all of the constraints as an objective function and perform constrained or unconstrained minimization. Our constraint solving techniques are based on *interval arithmetic*. The seminal work on the subject of interval-based computation is Moore [11]. More modern expositions include [1] and the proceedings of a quintennial conference [12, 13].

In the following $\Re$ represents the real line. We denote the closed interval from a to b, $a \leq b$, by $[a, b]$. If a and b are vectors in $\Re^s$, $a_i \leq b_i$, $i = 1 \ldots s$, then we regard the set $[a, b] = [a_1, b_1] \times \ldots \times [a_s, b_s]$ as an interval vector in $\Re^s$. We distinguish between point-valued and interval-valued variables by writing the latter in bold-face type, and we denote the space of intervals in $\Re^s$ by $[\Re]^s$. So, if $x \in \Re^s$, we may write $x \in \mathbf{x} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \in [\Re]^s$, indicating that a real vector x falls within some real interval vector $\mathbf{x}$ with lower vector $\underline{\mathbf{x}}$ and upper vector $\overline{\mathbf{x}}$. We often take the liberty of mixing point values with interval values within expressions in which case a point value, x, should be thought of as the degenerate interval $\mathbf{x} = [x, x]$. We define two special operators, the width function $w : [\Re]^s \to \Re^s$ by $w(\mathbf{n}) = \overline{\mathbf{n}} - \underline{\mathbf{n}}$; and the center function $c : [\Re]^s \to \Re^s$ by $c(\mathbf{n}) = (\overline{\mathbf{n}} + \underline{\mathbf{n}})/2$.

For a continuous function $h : \Re^s \to \Re$, we require that an *interval function*, $\mathbf{h} : [\Re]^s \to [\Re]$ satisfy the following condition:

$$\mathbf{h}(\mathbf{x}) \supseteq \{y \mid y = h(x), x \in \mathbf{x}\}. \qquad (1)$$

For example, given two intervals $\mathbf{x}$ and $\mathbf{u}$ in $[\Re]^s$, we can define binary $+$ and unary $-$ as

$$\mathbf{x} + \mathbf{u} := [\underline{\mathbf{x}} + \underline{\mathbf{u}}, \overline{\mathbf{x}} + \overline{\mathbf{u}}] \quad \text{and} \quad -\mathbf{x} := [-\overline{\mathbf{x}}, -\underline{\mathbf{x}}].$$

Binary $-$ can be defined by $\mathbf{x} - \mathbf{u} = \mathbf{x} + (-\mathbf{u})$. Likewise,

$$\mathbf{x} * \mathbf{u} := [\min(\underline{\mathbf{x}}\underline{\mathbf{u}}, \overline{\mathbf{x}}\underline{\mathbf{u}}, \underline{\mathbf{x}}\overline{\mathbf{u}}, \overline{\mathbf{x}}\overline{\mathbf{u}}), \max(\underline{\mathbf{x}}\underline{\mathbf{u}}, \overline{\mathbf{x}}\underline{\mathbf{u}}, \underline{\mathbf{x}}\overline{\mathbf{u}}, \overline{\mathbf{x}}\overline{\mathbf{u}})] \quad \text{and}$$

$$1/\mathbf{x} := [1/\overline{\mathbf{x}}, 1/\underline{\mathbf{x}}].$$

Division of two intervals is then $\mathbf{x}/\mathbf{u} = \mathbf{x}*(1/\mathbf{u})$. All of these operations form the minimal bounding interval satisfying (1).

In general, a continuous function h maps a compact set to a compact set, hence $\mathbf{y} = \mathbf{h}(\mathbf{x})$ is a closed interval, and therefore a point in $[\Re]$.[1] Consequently, interval extensions exist for most commonly encountered functions as well as for comparison operators and set-theoretic operations. The only non-standard operation is $\cup$, which we define as the minimum bounding interval of its operands instead of strict set-theoretic union. Also, we note that interval comparison is a three-valued logic: two intervals may be related by less than, greater than, or neither.

We define the *interval extension* of a function $H : \Re^s \rightarrow \Re^m$ with component functions $h_i : \Re^s \rightarrow \Re$, $i = 1, \ldots, m$ as

$$\mathbf{H}(\mathbf{x}) = \mathbf{h}_1(\mathbf{x}) \times \mathbf{h}_2(\mathbf{x}) \times \ldots \times \mathbf{h}_m(\mathbf{x}).$$

Given interval versions of the basic algebraic and trigonometric operators, the interval extension of a function can be computed by replacing all point operators and functions with their corresponding interval equivalents. In short, the interval extension of a function provides a convenient means for computing the range of a function applied to interval arguments.

3.1 Interval Bisection

As noted previously, we assume that observation errors are bounded so that a data vector z can be represented as an interval $\mathbf{z} = [z - t, z + t]$ where t is a known error bound. $\mathbf{Z}$ represents the conjunction of these vectors. If the interval extension of g^*, $\mathbf{g}^*$, is evaluated on a parameter interval vector $\mathbf{n}$ and data interval vector $\mathbf{Z}$, and $\mathbf{g}^*(\mathbf{n}, \mathbf{Z}) \leq 0$, then every point in $\mathbf{n}$ is consistent with Z. Every point in an interval vector $\mathbf{n}$ is inconsistent with Z if, for some component k of $\mathbf{g}^*$, $\mathbf{g}^*_k(\mathbf{n}, \mathbf{Z}) > 0$. If neither case holds, then no decision can be made. Using these ideas, we define an interval *reduce*() operator as:

reduce$(\mathbf{n}, \mathbf{g}^*, \mathbf{Z})$:

1. For each component i, $i = 1, \ldots, s$, trisect[2] $\mathbf{n}$ in i, yielding intervals $\mathbf{n}_{1,1}, \mathbf{n}_{1,2}, \ldots, \mathbf{n}_{s,2}, \mathbf{n}_{s,3}$.
2. If for some component k, $\mathbf{g}^*_k(\mathbf{n}_{i,j}, \mathbf{Z}) > 0$ then $\mathbf{n}_{i,j} := \emptyset$.
3. $\mathbf{n} := \bigcap_{1 \leq i \leq s} \bigcup_{1 \leq j \leq 3} \mathbf{n}_{i,j}$

Intuitively, *reduce*() tests each component of an interval parameter vector to see if a subinterval can be "sliced off" by showing it is inconsistent with the system of constraints. This simple "first-order" version of interval reduction will be used throughout this paper. The use of higher order or other modified versions of *reduce*() are discussed in [6].

[1]For the interested reader, we note that it is relatively straightforward to define a topology on the space of closed intervals so that the continuity of a function h defined on $\Re^s$ carries over to its interval equivalent $\mathbf{h}$ defined on $[\Re]^s$ [1, 11].

[2]Trisection is the division of a node into three equal parts by division of single component.

For an interval n, *bisect*(n, d) computes the set of subintervals resulting from bisecting component d of n. The skeleton of an algorithm for incremental computation of a solution set is:

```
generalized-bisection(n, g*, Z) :

1. (Initialization)
    (a) $\mathcal{Q} := \{\mathbf{n}\}$.
    (b) $\mathcal{L} := \emptyset$.
2. Termination Test
    (a) If $\mathcal{Q} = \mathcal{L} = \emptyset$, stop with inconsistency.
    (b) If $\mathcal{L} \neq \emptyset$ stop with success.
3. (Reduction)
    (a) Remove an interval x from $\mathcal{Q}$.
    (b) Compute reduce(x, g*, Z).
    (c) If $\mathbf{x} = \emptyset$, go to step 2.
    (d) If $\mathbf{g}^*(\mathbf{x}, \mathbf{Z}) \leq 0$, then $\mathcal{L} := \mathcal{L} \cup \{\mathbf{x}\}$. Go to step 2.
4. (Bisection)
    (a) Choose a dimension $1 \leq d \leq s$ with $w(\mathbf{x}_d) \geq 0$.
    (b) $\mathcal{Q} := \mathcal{Q} \cup$ bisect(x, $d$).
    (c) Go to step 3.
```

Intuitively, the algorithm can be thought of as incrementally "classifying" portions of the parameter space as inside or outside the solution set. At every iteration, the set of intervals $\mathcal{Q} \cup \mathcal{L}$ is a partitioning of the current best approximation to the solution set. The algorithm terminates with success if a feasible point is found and inconsistency if the system of constraints has no solution.

There are two points of nondeterminism in the algorithm: the choice of node to take from the queue, and the choice of dimension to bisect. It is crucial that certain fairness properties hold in order to guarantee convergence. In practice, this is seldom a problem, although *good* choices can significantly improve convergence. These points will be discussed in Section 4.

3.2 Some Comments on Convergence

There are two aspects to the convergence of constraint solving methods: structural and statistical. By structural convergence, we mean the convergence of the computed set of feasible points to the true set of feasible points for a given system of constraints. Statistical convergence refers to the decrease in size of the solution set as more (stochastic) constraints are added to the system. We cannot provide a complete discussion of convergence in this paper, but we will strive to establish some intuition with minimal notation and theory. All results will apply to scalar systems of constraints, though most extend to multidimensional systems in a straightforward way.

For a fixed system of constraints, if the interval operators are properly defined and satisfy (1), the algorithm will halt with success only if there is a point that satisfies all constraints. Assuming the constraint functions obey a Lipschitz condition, that variables representing data only appear once in interval expressions, and that certain fairness properties hold when choosing nodes from the active queue and choosing components to bisect, it is possible to demonstrate that the interval bisection procedure converges to the solution set of a system of constraints.

Interval bisection makes no assumptions about the nature of sensing errors. However, in practice, most observation errors are time-varying to some degree. In such cases, repeated intersection over time leads to increasingly tighter bounds on the unknown quantity. For example, by straightforward manipulation the system $z_i = x+v_i$, $v_i \in [-1,1]$, yields the constraint $x \in [z_i-1, z_i+1]$ for each i. Assuming v varies within its uncertainty interval, the intersection will never be empty and will become continually smaller. For example, given the sequence of observations $10, 9.5, 10.5, 9.1, 10.1$, under the above constraint the final value *must* fall in the interval $[9.5, 10.1]$. More generally, the following properties hold:

Let $X_1, X_2, \ldots X_n$ be a sequence of independent random variables such that

- $X_i \in [\theta - a_i, \theta + b_i]$, and
- for any $\delta > 0$, there is an $\epsilon > 0$ such that $P(X_i < \theta - a_i + \delta) \geq \epsilon$ and $P(X_i > \theta + b_i - \delta) \geq \epsilon$ (the distribution places mass around the endpoints of the interval).

Let $S_n = \bigcap_{i=1}^{n} [X_i - b_i, X_i + a_i]$. Then

1. For all $n > 0$, $\theta \in S_n$.
2. For any $c > 0$, $\lim P(w(S_n) > c) \to 0$ as $n \to \infty$. (the sequence of sets converges in probability [4]).

An immediate consequence of the above is the following:

Let $X_1, X_2, \ldots X_n$ be a sequence of random variables as above and let f be a continuous function defined on
$\bigcup_{i=1}^{n} [\theta - a_i - b_i, \theta + a_i + b_i]$. Then

1. $f(\theta) \in S_n^f = \bigcap_{i=1}^{n} f([X_i - b_i, X_i + a_i])$.
2. For any $c > 0$, $\lim P(w(S_n^f) > c) \to 0$ as $n \to \infty$.
3. $\theta \in \{a \mid f(a) \in S_n^f\}$.

In other words, convergence is preserved under continuous transformations. Furthermore, if f^{-1} is also a continuous function then the uncertainty interval on θ is a closed interval. Although these results are formulated for equalities, the same results can be derived for inequalities using similar methods of argument.

While we have not formally stated and proved such a proposition, these two results suggest that interval bisection on increasingly larger systems of stochastic constraints converges to the true underlying solution set of the system.

Finally, a word about so-called *outliers*. Our convention is that an outlier is an observation that exceeds the error bound used to formulate the system of constraints. Therefore these observations potentially render the system of constraints inconsistent. We use a very straightforward approach to handling outliers in data. If it is suspected that c percent of the data is outlier data, then no interval is rejected until it is inconsistent with $c'n$ constraints, where n is the total number of constraints in the system and $c' > c$. For large data sets this has proved to be a very workable approach to robustness, and can also be proved statistically convergent under reasonable assumptions.

4 Implementation and Experimentation

We have performed some initial tests of the constraint solving system on the problem of two-dimensional packing. This has been carried out both in simulation and with real data.

The constraint solving system we use, implemented in C++, supports writing procedures to compute the truth value of constraints such as the inequalities presented above, and provides "binding" operators for associating data with the constraints. Nodes are chosen from the active queue by looking at the extent to which they satisfy the system of constraints. This measure of satisfaction is much like an objective function in that it governs and guides the search for a solution to the constraints. The function is formulated as follows.

Suppose a constraint is of the form $g(p,z) \leq \alpha$. Expanding in a Taylor's series about a point p_0 and dropping higher order terms leads to the linear form:

$$g(p_0, z) + g'(p_0, z) \cdot (p - p_0) \leq \alpha.$$

The gradient vector $g'(\cdot,\cdot)$ is the local normal to the constraint surface in parameter space. So, a coarse approximation to the distance from the point p_0 to the constraint surface is given by:

$$d(p_0, z, g) = \frac{g(p_0, z) - \alpha}{\|g'(p_0, z)\|}, \quad \|g'(p_0, z)\| \neq 0.$$

Here, we have preserved sign, so negative distances indicate that the parameter point satisfies the constraint. Let $C_i = \langle g_i, O_i \rangle$ represent a constraint in the form described above together with a set of observations O_i, and let $C = \{C_1, C_2, \ldots, C_n\}$. Then we define the objective function as

$$E(p, C) = \max_{\langle g,O\rangle \in C} \max_{z \in O} d(p, z, g)$$

In other words, the objective function is approximately the distance (in parameter space) to the boundary of the feasible set.

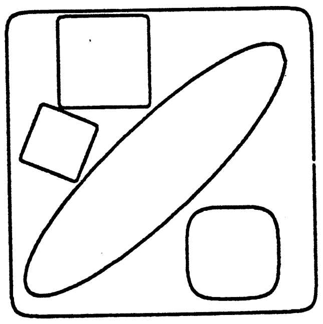

Figure 1. The configuration of several simulated objects as generated by the packing algorithm.

Placements for packing are computed sequentially. First, the largest object is chosen and bisection carried out until the value of the above-defined objective function is negative for some point within an interval (indicating that the interval contains at least one feasible point). These points are sampled as *reduce*() is computed: each is the center point of one of the subintervals that *reduce*() evaluates. The system parameter vector is then extended to accommodate another object, and the algorithm searches for a joint placement of two objects with the placement of the first restricted to the interval known to contain a feasible point. This process is iterated for as many objects as needed. It is important to note that, while the placement for prior objects is restricted to an interval, the data modeling constraints are computed anew. This allows the algorithm to adjust modeling parameters to accommodate the additional constraints from newly introduced objects.

To give an example, suppose we are given a square container with 20 mm sides and origin at the upper left corner. An object with size 23mm by 6mm is introduced. The initial bounds on placement are all positions in the container and angles in the range $-\pi/4$ to $\pi/4$. Bounds on the size of the object are computed by fitting a rough bounding box to the data. After 18 iterations, the algorithm produces intervals $[9.6, 10.4]$ and $[8.9, 11.1]$ on position within the container, and an interval of $[\pi/6, \pi/4]$ on orientation. Note, in particular, that the algorithm has correctly deduced that the object must be tipped to fit into the container. A second object of size 6mm square is introduced. After another 40 iterations, intervals of $[6.7, 8.0]$, $[16.3, 17.0]$ and $[\pi/12, \pi/12]$ are produced for the translation and orientation parameters of this object. The bounds on placement of the first object are also smaller. A slightly rounded version of the second object is added, and after an additional 23 iterations the results are the intervals $[13.6, 15.6]$, $[3.0, 4.9]$ and $[-\pi/4, \pi/4]$ describing the placement. Finally, a small square of size 4mm on a side is added and after 38 additional iterations the algorithm finds placement $[3.0, 3.7]$, $[11.2, 11.9]$ and $[-\pi/4, -\pi/12]$. One of the final configurations is shown in Figure 1.

The algorithm will also determine when objects cannot be fit by noting that the problem is inconsistent: there are no feasible points. The same comments hold for multiple objects. However,

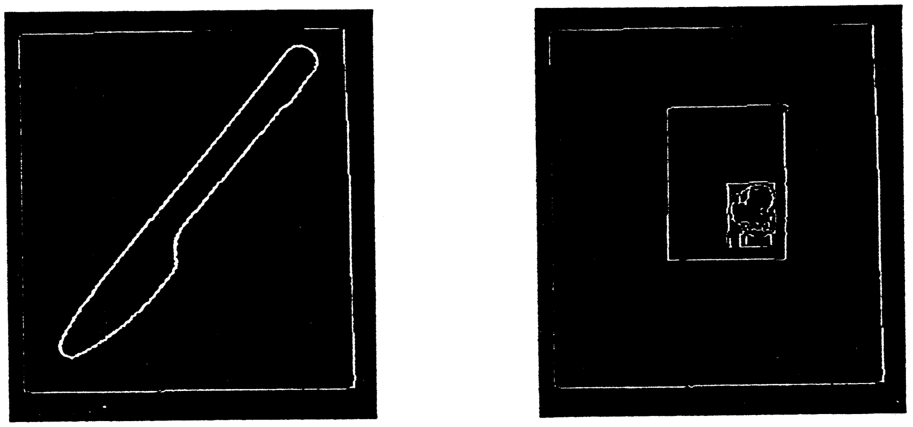

Figure 2. The generated placement of the test objects is depicted by superimposing their contours on the contours of the container.

it is always possible that there *is* a solution for multiple objects that is lost by fixing the position of the first object to a particular region. The latter can be overcome by either solving for a placement of the objects simultaneously or generating additional possibilities for the placement of the first object.

The algorithm has also been tested on real data. The example presented here is to pack a knife and a small piece of paper into a rectangular area (a cd case). The data used were image contours of the objects. The edges were extracted using the "vgef" operator (an implementation of the filter described in [16]) supplied with the Khoros system [19].

Finding a placement for the individual objects took relatively little time. For the paper alone, the algorithm took 6 iterations to generate the first feasible point. The knife, since it is substantially "tighter" fit, required 50 iterations to generate the first feasible point. Placing both objects simultaneously particularly difficult because the paper only will fit on one side of the knife. At a very early point in the solution process, the algorithm must decide on the approximate region to place the paper. If the correct initial placement region for the paper is chosen, the current implementation is able to generate and verify a placement for both the knife and paper in 101 iterations. If the initial choice is wrong, it spends several hundred iterations proving that there is no placement on one side before moving to the other side. This is a general problem when there are symmetries in problems. For this reason, we plan to perform a "beam" search when symmetries such as this appear. This will avoid the backtracking problem at the cost of maintaining multiple search paths.

Figure 2 shows the placements that were generated for the single objects, and Figure 3 shows a placement that was generated for both objects. Note that the knife has a clean, but irregular outline, and that the paper is somewhat irregular, and also contains interior texture. None of these factors affect the operation of the algorithm.

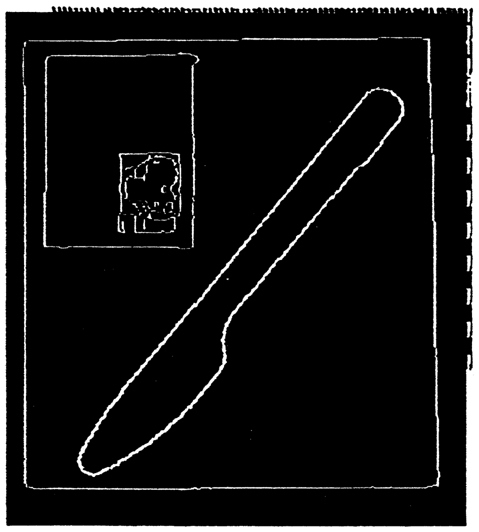

Figure 3. A placement that was generated for the two test objects.

5 Discussion

We have presented an approach to sensor-based decision making in unstructured environments. The features of this approach are conceptual simplicity, generality, and provable convergence for a wide variety of problems. In the next stage of our work, we plan to couple this packing algorithm with our robot arm. As noted in the previous section, the algorithm we have described is able to generate placements for single objects relatively effectively. The constraints for generating placements for multiple objects requires projecting the data for the objects into their proposed positions, and verifying the placement of a new object. A more natural approach is to generate the placement for a single object, place the object into the container with the arm, acquire another image, place a second object, and so forth. These techniques may also be combined with "active" exploratory techniques that simply try to insert an object and only invoke the geometric method described above when no solution is found.

Acknowledgements: This research was supported by DARPA grant N00014-91-J-1577, by National Science Foundation grants IRI-9109116 and DDM-9112458, and by funds provided by Yale University.

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, N.Y., 1983.

[2] P. Allen. *Robotic Object Recognition Using Vision and Touch.* Kluwer, MA, Boston, 1988.

[3] P. J. Besl. Geometric modeling and computer vision. *Proceedings of the IEEE*, 76(8):936–958, August 1988.

[4] P. Bickel and K. Doksum. *Mathematical Statistics.* Holden-Day, Oakland, CA, 1977.

[5] W. Grimson. Disambiguating sensory interpretations using minimal sets of sensory data. In *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, pages 286–291. San Fransisco, April 1986.

[6] G. D. Hager. Interval-based bisection methods for sensor data fusion. To appear as Yale computer science technical report 863, 1991.

[7] D. Huttenlocher and S. Ullman. Object recognition using alignment. In *Proceedings of the First International Conference on Computer Vision*, pages 102–111. IEEE Computer Society Press, Washington, D.C., 1987.

[8] D. Kriegman and J. Ponce. On recognizing and positioning curved 3D objects from image contours. *IEEE Trans. Pattern Anal. Machine Intell.*, 12(12):1127–1137, 1990.

[9] A. Leonardis, A. Gupta, and R. Bajcsy. Segmentation as the search for the best description of the image in terms of primitives. Technical Report MS-CIS-90-30, University of Pennsylvania, Philadelphia, PA, 1990.

[10] D. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, New York, N.Y., second edition, 1984.

[11] R. E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1966.

[12] K. Nickel, editor. *Interval Mathematics 1980.* Academic Press, New York, N.Y., 1980.

[13] K. Nickel, editor. *Interval Mathematics 1985*, volume 212 of *Lecture Notes in Computer Science.* Springer-Verlag, New York, N.Y., September 1985.

[14] A. Pentland and S. Sclaroff. Closed-form solutions for physically-based shape modeling and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):715–729, July 1991.

[15] K. Rao and R. Nevatia. Computing volume descriptions from sparse 3-D data. *International Journal of Computer Vision*, 2:33–50, 1988.

[16] J. Shen and S. Castan. An optimal linear operator for edge detection. In *Proceedings of the 1986 Conference on Computer Vision and Pattern Recognition.* IEEE Computer Society Press, Washington, D.C., 1986.

[17] F. Solina and R. Bajcsy. Recovery of parametric models from range images: The case for superquadrics with global deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):131–147, February 1990.

[18] D. Terzopoulos, A. Witkin, and M. Kass. Symmetry-seeking models for 3-D object reconstruction. In *Proceedings of the First International Conference on Computer Vision*, pages 269–276. June 1987.

[19] The Khoros Group. *The Khoros Users Manual.* The University of New Mexico, Albuquerque, NM, 1991.