Interval-Based Techniques for Sensor Data Fusion

MS-CIS-90-80
GRASP LAB 241

Greg Hager

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

October 1990

Interval-Based Techniques for Sensor Data Fusion

Greg Hager University of Pennsylvania
GRASP Lab - Room 301C
3401 Walnut St.
Phila., PA 19104/6228

September 14, 1990

Abstract

We view the problem of sensor-based decision-making in terms of two components: a sensor fusion component that isolates a set of models consistent with observed data, and an evaluation component that uses this information and task-related information to make model-based decisions. This paper describes a procedure for computing the *solution set* of parametric equations describing a sensor-object imaging relationship. Given a parametric form with s parameters, we show that this procedure can be implemented using a parallel array of $6s^2$ processors. We then describe an application of these techniques which demonstrates the use of task-related information and set-based decision-making methods.

1 Introduction

A central problem in sensor data fusion is the description of observed data in a canonical form useful for decision making. One approach to describing sensor data is to choose a restricted class of parametric primitives, and fit the data to one or more of these primitive elements. The basic notion of line or curve fitting has been used for many years and pervades many scientific disciplines. Within the vision and robotics community there has been much recent work on fitting relatively complex parametric forms such as generalized cylinders or superquadrics to range, tactile, and visual data.

While fitting is a powerful notion, many of the techniques currently employed have severe shortcomings. In particular, methods such as least squares tend to have very primitive notions of fit accuracy, and consequently it is difficult to determine the extent to which the observed data determines the model parameters or how well the model matches the data. Given the usual inaccuracies of sensor data as well, it is quite possible that the *point* estimate delivered by least squares or similar techniques is a poor representation of the observed data.

An alternative approach is to directly compute the *set* of all model points consistent with observed data, and to use task-related information to make model-based decisions. For example, when grasping an object with a gripper, the location of the object must be known to an accuracy sufficient to place the gripper on the object without a collision. Given the set of all object locations and sizes, we can determine when the uncertainty in size and location has been reduced to a level where the operation is guaranteed of succeeding (if it is indeed possible). For small objects and large grippers, the set can be very large and the operation will still succeed, while for objects which are nearly the size of the gripper opening, a very precise estimate is required and therefore the solution set must be small.

In our past work, we have examined decision-theoretic techniques [Hager, 1990; Hager & Mintz, 1989] for describing task-related constraints, and developed grid-based approximation techniques for implementing decision-theoretic sensor information fusion and sensor planning. Those techniques, while very general, suffered from severe complexity limitations when fusing large volumes of data into complex models. Examination of their behavior led to the observation that important factor in system performance was not its use of a statistical description of observation error, but rather the fact that we assumed observation error was *bounded.* We can improve grid-based method performance by considering two separate sub-problems: the problem of determining the set of models consistent with observations up to a known, bounded error, and the problem of building a probability description on the resulting set.

This article addresses the problem of making effective use of tolerances to support *set-based* estimation and decision-making. We describe a technique for computing set-based estimates that is able to solve problems of much higher parametric complexity using larger volumes of data than our previous grid-based methods. This method uses generalized bisection based on interval projections. The process can be thought of as incrementally building a search tree describing a partioning of the solution space, and pruning portions of the tree that are incompatible with observation. One of the major results in developing this technique is that we can show upper bounds on the amount of data needed at each node of the tree, and consequently we can construct a sensor information processor consisting of a data selection component pipelined with a parallel processor array with size that is a quadratic function of the problem dimensionality.

In the next section, we state the class of sensor data fusion problems we are considering, and define the precise problem we are attempting to solve. In section 3, we introduce interval analysis and describe our generalized bisection procedure. In Section 4 we describe a data selection algorithm that allows us to reduce the complexity of our basic computational operation to a quadratic function of model dimensionality. In Section 5 we describe an application using set-based decision criteria. We conclude with a discussion of future enhancements and applications of this work. We note that a more extended version of the results in this article can be found in [Hager, 1990].

2 The Problem

In its most general form, the relationship between an observation, z of dimension m, and model parameters, p of dimension s, can be written

$$\begin{aligned} z &= H(x,d,v), \quad z \in \mathcal{Z} \subseteq \Re^m, \quad x \in \mathcal{X}, \quad d \in \mathcal{D}, \quad v \in \mathcal{V} \subseteq \Re^n, \\ 0 &= g(p,x), \quad p \in \mathcal{P} \subseteq \Re^s. \end{aligned} \tag{1}$$

where the implicit function g describes model geometry, and the explicit function H describes sensing geometry. The vector d denotes additional kinematic or physical degrees of freedom of the sensor system (calibration or control parameters), and the additional parameter v is a *nuisance* parameter denoting non-deterministic disturbances of the sensor output. The vector x is an observable quantity which bears a known relationship to both the sensor outputs and to the underlying model parameters.

There are several specializations of these forms which occur in practice:

- For convenience sake, for fixed $\mathcal{V}$ we can write a single form

$$g^*(p,z,d) = \begin{bmatrix} g(p,x) \\ z - H(x,d,v) \end{bmatrix} = 0, \quad \text{for some } x \in \mathcal{X} \text{ and } v \in \mathcal{V}. \tag{2}$$

- If H is bijective for fixed d and v, then we could write (1) as

$$g(p, H^{-1}(z,d,v)) = 0, \quad \text{for some } v \in \mathcal{V}.$$

- If g can be written explicitly, we can compose the two functions and write

$$z = H(p,d,v), \quad v \in \mathcal{V}. \tag{3}$$

- If, in addition, the error vector v affects the system additively, we may write

$$z = H(p,d) + v, \quad v \in \mathcal{V} \subseteq \Re^m. \tag{4}$$

In the rest of this article, we will use the form (2) except in some special cases where the special structure of (3) or (4) is crucial to the discussion. Problems expressed in the form of (1) tend to be more complex to solve, both mathematically and computationally than problems which can be described using explicit equations. It is usually preferable to reduce problems to explicit form (3) or explicit additive form (4) whenever possible.

This method of problem description is both general and practical for a wide variety of sensing applications. Moreover, it *separates* the description of the sensing apparatus from the data representation. Consequently, information from several different sensors can be integrated into the same representational format and, conversely, information from one sensor can be integrated into several different representational formats merely by changing the coupling between sensor description and geometric model. The following is a simple example that is also used later in this article:

Example 2.1 Pentland [1986] introduced *superquadrics* as a modeling primitive, and Solina [1987] developed a least-squares algorithm for recovering superellipsoids (convex superquadrics) from range data. Superellipsoids are described by an implicit parametric equation of the form:

$$g_{\texttt{isq}}(a,\gamma,l;u) = \left[\left(\frac{u_x - x}{a_1} \right)^{\frac{2}{\gamma_2}} + \left(\frac{u_y - y}{a_2} \right)^{\frac{2}{\gamma_2}} \right]^{\frac{\gamma_2}{\gamma_1}} + \left(\frac{u_z - z}{a_3} \right)^{\frac{2}{\gamma_1}} - 1 = 0.$$

The vector $[a] = [a_1, a_2, a_3]$ can be interpreted as the size of the superellipsoid, the vector $[\gamma] = [\gamma_1, \gamma_2]$ governs the shape of the superellipsoid, and $l = [x, y, z]$ is a location in space. This form also be augmented with a rotation transformation to describe superquadrics in arbitrary orientation in space as well as other parametric deformations [Solina & Bajcsy, 1990], but we will not use these generalizations in this article.

We can also describe a superellipsoid situated at the origin with explicit form

$$E(a,\gamma;\eta,\omega) = \begin{bmatrix} a_1 C_\eta^{\gamma_1} C_\omega^{\gamma_2} \\ a_2 C_\eta^{\gamma_1} S_\omega^{\gamma_2} \\ a_3 S_\eta^{\gamma_1} \end{bmatrix}, \quad 0 \le \eta \le \frac{\pi}{2}, \quad 0 \le \omega \le \frac{\pi}{2}, \quad 0 < \gamma_i \le 2, \tag{5}$$

where $C_x = \cos(x)$ and $S_x = \sin(x)$. This form describes the other seven octants by appropriate introduction of plus and minus signs. The angles η and ω are additional parameters required to convert the implicit form to an explicit form. The explicit description of an arbitrary superellipsoid located in space can be expressed as

$$g_{\mathbf{esq}}(l, a, \gamma; \eta, \omega) = {}^{b}T_{o}(l)E(a, \gamma; \eta, \omega).$$

The information observed by a laser scanning device located at the origin of the world coordinate system is simply

$$z = u + v, \quad v \in \mathcal{V}.$$

The variable $v \in \mathcal{V}$ describes quantization error in x and y due to finite resolution of the scanner cells, and errors in z due to inaccuracies in computed range. The conjunction or composition of this expression with either the implicit or explicit form of the superellipsoid yields a well-formed description of the sensor-object observation relationship.

The sensor data fusion problem is to recover model geometry as expressed by the parameter vector p from a series of data pairs $\langle z_i, d_i \rangle$, $i = 1, \ldots n$ to the accuracy required for the specific task being performed. As it turns out, in many applications the error in sensor data is relatively small and it is often reasonable to assume that v comes from a *bounded* set, $\mathcal{V}$. In this case, given sensor data pairs we can define the *solution set* consistent with a set of observations. We now state the version of the sensor data fusion problem we consider in the remainder of this article:

Given a data set O consisting of pairs of vectors $\langle z_i, d_i \rangle$, $i = 1 \ldots n$ and a sensor-object description g^*, compute an approximation to

$$\mathcal{P}^* = \{p \in \mathcal{P} \mid 0 = g^*(p, z_i, d_i),\ 1 \leq i \leq n\}$$

Generally speaking, solution sets will be of such complexity that, except for trivial cases, even closed form *approximations* to this set are difficult to develop. Hence, our interest in computational techniques for approximating this set. To compute an approximation, we define an operator, F, yielding a sequence of sets $\mathcal{P}_0, \mathcal{P}_1, \ldots$ such that

- $\mathcal{P}_0 = \mathcal{P}$,
- $\mathcal{P}_{k+1} = \mathcal{P}_k \cap F(\mathcal{P}_k)$,
- $\mathcal{P}_k \rightarrow \mathcal{P}^*$.

The grid-based technique mentioned above can be thought of as a member of this class of approximation algorithms. However, as indicated in the introduction, the complexity of this method makes its application prohibitive for a number of problems of practical interest. In particular, when isolating the solution set it make sense to develop the grid incrementally based on the structure of the sensing equations. We now describe the generalized bisection algorithm we have developed for this purpose.

3 A Brief Review of Interval Analysis

Our solution to the problem of isolating solution sets makes heavy use of concepts from *interval analysis*. Interval analysis originated in the attempt to build rigorously correct numerical procedures for processes such as root bracketing, differential equation solving, numerical integration, or function minimization. The heart of the approach is represent a real number by a *bracketing interval*, each endpoint corresponding to a number having an exact representation within the machine, and to ensure that all mathematical operations on numbers preserve this property. The seminal work on the subject is Moore [1966]. More modern expositions include [Alefeld & Herzberger, 1983] and the proceedings of a quintennial conference[Nickel, 1980; Nickel, 1985]. Specific papers we have found most relevant to the problems we will be discussing include [Adams, 1980; Sikorski, 1982; Eiger *et al.*, 1984; Kearfott, 1987; Kearfott, 1987; Kearfott, 1990].

Notation and Terminology In the following, let $\Re$ denote the real line and $\Re^s$ denote Euclidean s-space. We denote the open interval from a to b in $\Re^1$ by (a,b) and the closed interval by $[a,b]$. If a and b are points in $\Re^s$, then we regard the set $(a,b) = (a_1,b_1) \times \ldots \times (a_s,b_s)$ as a generalized open interval in $\Re^s$, and $[a,b] = [a_1,b_1] \times \ldots \times [a_s,b_s]$ a generalized closed interval in $\Re^s$. Given a set $\mathcal{S}$ in $\Re^s$, we define the *bounding interval* of $\mathcal{S}$ as the smallest generalized closed interval containing $\mathcal{S}$. Henceforth we drop the term "generalized" when it is apparent from context that the interval is in $\Re^n$, $n > 1$.

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 $x \in \Re^s$ is some real number, we may write $x \in \mathbf{x} = [\underline{\mathbf{x}}, \overline{\mathbf{x}}] \in [\![\Re]\!]^s$, indicating that a real value x falls within some real interval value $\mathbf{x}$ 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 $w : [\![\Re]\!]^s \rightarrow \Re^s$ by $w(\mathbf{n}) = \overline{\mathbf{n}} - \underline{\mathbf{n}}$; and the center function $c : [\![\Re]\!]^s \rightarrow \Re^s$ by $c(\mathbf{n}) = (\overline{\mathbf{n}} + \underline{\mathbf{n}})/2$. A *simple sectioning* of an interval $\mathbf{n}$ in dimension i will be a division of $\mathbf{n}$ into nonempty components $\mathbf{a}$ and $\mathbf{b}$ such that: $\mathbf{n} = \mathbf{a} \cup \mathbf{b}$, and $\overline{\mathbf{a}_i} = \underline{\mathbf{b}_i}$. A *sectioning* is the above generalized to more than two intervals. An *m-sectioning*, $m \geq 2$ is a sectioning into m components of the same size.

3.1 Approximating the Range of a Function

Suppose we are given a function $h : \Re^s \rightarrow \Re$. For any vector x, we can calculate, y, the image of x under h by $y = h(x)$. Now, suppose that instead of a *value* x, we are given an *interval* of values, $[\underline{x}, \overline{x}]$ describing an s-rectangle and wish to compute its projection. For a given continuous function $h : \Re^s \rightarrow \Re$, we can define an *interval function*, $\mathbf{h} : [\![\Re]\!]^s \rightarrow [\![\Re]\!]$ by

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

Note that 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 $[\![\Re]\!]$. 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 extension $\mathbf{h}$ defined on $[\![\Re]\!]^s$ [Moore, 1966; Alefeld & Herzberger, 1983].

Example 3.1 Given two intervals $\mathbf{x}$ and $\mathbf{u}$ in $[\![\Re]\!]^s$, we can define the functions 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 $\mathbf{x} - \mathbf{u} = \mathbf{x} + (-\mathbf{u})$. Moreover, these operations always form the minimal bounding interval of the range of the underlying operator applied to the intervals $\mathbf{x}$ and $\mathbf{u}$.

Given such interval extensions for the basic algebraic and trigonometric operators, the most straightforward approach to computing the extreme values of a function is to take the algebraic description of the function, and replace all of the operators with the corresponding interval operators.

The major disadvantage of the direct use of interval computations is that they often compute supersets of the exact range sets. This happens because each occurrence of a variable in an expression is treated as a *different occurrence* of an interval variable. Consider the following simple case:

Example 3.2 Defining the multiplication operation between two intervals is somewhat more complicated than $+$ or $-$. Perhaps the most straightforward definition is

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

If $\mathbf{x}$ and $\mathbf{u}$ are *independent* interval variables, then this operation again forms the minimal interval covering the product of the two expressions. However, suppose $\mathbf{x} = [-1, 1]$ and we compute $\mathbf{x} * \mathbf{x}$. The above operation yields the interval $[-1, 1]$. But, if the interval variable $\mathbf{x}$ corresponds to a bracketing of a single fixed quantity, the *minimal* interval is $[0, 1]$.

This inaccuracy can be reduced by suitable rewriting of expressions and by implementing the interval computations of more complex expressions containing multiple occurrences of the same variables. For instance, in the example above it is quite simple to implement a "squaring" operator which computes the minimal range interval. Although this can become a difficult process, in most cases it is possible to use the initial interval implementation to prototype a solution, and then to incrementally refine the interval computation to provide better performance and use of information. In subsequent sections, we will assume that all interval computations produce the minimal correct interval.

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

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

We note that, in addition to the possibility of overly conservative scalar intervals, if we consider functions with non-scalar range it is often the case that there is *no* exact interval describing the range. The best we can hope for is the minimal bracketing interval. We note without proof that if $\mathbf{m} \subseteq \mathbf{n}$, then $\mathbf{H}(\mathbf{m}) \subseteq \mathbf{H}(\mathbf{n})$.

3.2 Interval Trees

An *interval tree node* will consist of a closed interval $\mathbf{n} = [\underline{\mathbf{n}}, \overline{\mathbf{n}}]$ and a set of two or more children, $D_{\mathbf{n}}$. For the sake of convenience, we will identify a tree node with its associated interval and write, for example, $\mathbf{n} \prec \mathbf{m}$ to indicate that the node $\mathbf{n}$ is higher in the tree than the node $\mathbf{m}$.

An interval tree node, $\mathbf{n}$, is *consistent* if $\mathbf{n}$ is nonempty *and* $\mathbf{n}$ is a leaf, or $\mathbf{n}$ is an inner node and $\mathbf{m} \subseteq \mathbf{n}$ for all $\mathbf{m} \in D_{\mathbf{n}}$. The node is *minimal* if no smaller interval satisfies the latter criterion. In short, a minimal, consistent node has a non-empty interval which encloses the intervals of all of its children, and no smaller interval could enclose those children. As a direct consequence, if $\mathbf{m} \preceq \mathbf{n}$, then $\mathbf{n} \subseteq \mathbf{m}$.

An *interval tree* is consistent if all of its nodes are consistent, and minimal if all of its nodes are minimal. Furthermore, we will refer to the tree as *reduced* if all non-leaf nodes have at least two children. In particular, for binary interval trees, a reduced tree corresponds to a full binary tree.

For an interval tree in $[\![\Re]\!]^s$ with leaf node $\mathbf{n}$, we define the operator $bisect(\mathbf{n}, d)$, $d \leq s$ as:

1. Bisect $\mathbf{n}$ creating two new intervals $\mathbf{n}_1$ and $\mathbf{n}_2$.
2. $D_{\mathbf{n}} := \{\mathbf{n}_1, \mathbf{n}_2\}$.

For a node $\mathbf{n}$ which is not the root of the tree, we define an operator $remove(\mathbf{n})$ by

1. Let $\mathbf{p} = \text{parent}(\mathbf{n})$.
2. $D_{\mathbf{p}} := D_{\mathbf{p}} - \{\mathbf{n}\}$.
3. If $|D_{\mathbf{p}}| = 0$, execute $remove(\mathbf{p})$.
4. If $|D_{\mathbf{p}}| = 1$,
   (a) If $\mathbf{p}$ is the root of the tree, mark the remaining child in $D_{\mathbf{p}}$ as the root, otherwise
   (b) Let $\mathbf{g} = \text{parent}(\mathbf{p})$.
   (c) $D_{\mathbf{g}} := (D_{\mathbf{g}} - \{\mathbf{p}\}) \cup D_{\mathbf{p}}$

Note this operation preserves consistency. It *does not* preserve minimality for nodes $\mathbf{m}$ such that $\mathbf{m} \prec \mathbf{u}$. Step 4 preserves the property of being a reduced tree.

3.3 Interval Reduction

With this, the interval reduction operation used in the rest of this report can be described as:

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

1. For each dimension i, $i = 1, \ldots, s$, trisection $\mathbf{n}$ in dimension i, yielding sets $\mathbf{n}_{1,1}, \mathbf{n}_{1,2}, \ldots, \mathbf{n}_{s,2}, \mathbf{n}_{s,3}$.
2. For all $\mathbf{n}_{i,j}$, if $0 \notin \mathbf{g}^*(\mathbf{n}_{i,j}, z_i, d_i)$ for some $\langle z_i, d_i \rangle \in O$, then $\mathbf{n}_{i,j} := \emptyset$.
3. $\mathbf{n} := \bigcap_{1 \leq i \leq s} \bigcup_{1 \leq j \leq 3} \mathbf{n}_{i,j}$

If each system output depended on a single model parameter, then each interval component could be subdivided and reduced independent of other interval components and the procedure outlined above is nearly optimal. Conversely, its performance will clearly degrade for systems with high degrees of coupling.

To understand why we have used trisection rather than bisection in *reduce*(), consider the system $z = x/y \pm t$ where t is small. If x is zero and y is some positive value, a bisection of a parameter space symmetric about 0 in the x coordinate *will not lead to a reduction.* The t interval surrounding $x/y = 0$ intersects both interval projections though the actual solution set may occupy a very small area in the parameter space. By employing trisection we avoid this problem by making it much less likely that an observation can be contained in all three interval projections. Trisection also increases the rate of convergence at the expense of more processing per *reduce*() call.

We note that this operation can be executed almost entirely in parallel by computing each element of the interval projection of each section independently. This requires $3sm$ processors (recall s is the size of the parameter vector and m is the size of the observation vector). The rate of speedup over serial execution depends on the number of common subexpressions in the interval function.

As an additional optimization, we can dismiss constraints as having been fully exploited. For sensor descriptions in explicit additive form (4), we see that if

$$\mathbf{h}_i(\mathbf{p}) \subseteq z_i - \mathbf{v}_i$$

then further bisection will not lead to any reduction based on this test. Thus, this component can be marked inactive. If all components become inactive, then $\mathbf{p}$ is completely contained in the solution space. Consequently, we may add a step to *reduce*() that checks to see if the interval projection is contained in the tolerance envelope:

4. If the sensor description can be written explicitly and additively, $\mathbf{H}(\mathbf{n}, d) \subseteq z - \mathcal{V}$, mark $\mathbf{n}$ as contained.

3.4 Modified Bisection

Historically, generalized bisection has been used to bracket the roots of nonlinear equations. We make two major modifications to the standard procedure. First, the algorithm constructs an explicit solution tree and, given a solution tree and additional data, the algorithm combines information starting at the root of the solution tree. This allows elimination of portions of the solution set high in the tree and saves unneeded work. Second, the algorithm checks for containment of an interval in the solution set, and terminates the search along that branch. In the following $\mathbf{g}^*(\cdot)$ is the interval extension of $g^*(\cdot)$, $\mathbf{n}$ is a tree node corresponding to a bracketing interval for the solution set, and O is a series of data/description vector pairs.

Algorithm 3.1

generalized-bisection($\mathbf{n}, \mathbf{geofn}^*, O$)

1. *(Initialization)*
   (a) Set a vector of coordinate tolerances, $\epsilon_i, 1 \leq i \leq s$.

(b) $\mathcal{Q} := \{\mathbf{n}\}$.

2. *(Reduction)*
   (a) If $\mathcal{Q} = \emptyset$, stop.
   (b) Remove an interval $\mathbf{x}$ from $\mathcal{Q}$.
   (c) Compute $\mathbf{x} := \mathit{reduce}(\mathbf{x}, \mathbf{geofn}^*, O)$.
   (d) If $\mathbf{x} = \emptyset$, execute $\mathit{remove}(\mathbf{x})$ and go to 2.
   (e) If $\mathrm{w}(\mathbf{x}) \leq \epsilon$ or $\mathbf{x}$ is marked as contained, then $\mathcal{L} := \mathcal{L} \cup \{\mathbf{x}\}$ and go to step 2.
   (f) If $D_{\mathbf{x}} = \emptyset$, go to step 4, otherwise go to step 3.
3. *(Following Tree)*
   (a) For each $\mathbf{c} \in D_{\mathbf{x}}$,
      i. compute $\mathbf{c} := \mathbf{c} \cap \mathbf{x}$.
      ii. If $\mathbf{c} = \emptyset$, execute $\mathit{remove}(\mathbf{c})$.
   (b) If $D_{\mathbf{x}} = \emptyset$, execute $\mathit{remove}(\mathbf{x})$, otherwise $\mathcal{Q} := \mathcal{Q} \cup D_{\mathbf{x}}$.
   (c) Go to step 2.
4. *(Bisection)*
   (a) Choose a dimension $1 \leq d \leq s$ such that $\mathrm{w}(\mathbf{x})_d \geq \epsilon_d$.
   (b) Execute $\mathit{bisect}(\mathbf{x}, d)$
   (c) $\mathcal{Q} := \mathcal{Q} \cup D_{\mathbf{x}}$.
   (d) Go to step 2.

Let $\preceq$ be the natural partial ordering of nodes defined by the tree. We note the following few facts about the algorithm:

- If $\mathbf{n}$ and $\mathbf{m}$ are both on the queue, then $\mathbf{n} \not\preceq \mathbf{m}$ and $\mathbf{m} \not\preceq \mathbf{n}$.
- From the use of *bisect*(), *reduce*(), and *remove*(), it is easy to show that if $\mathbf{n} \not\preceq \mathbf{m}$ and $\mathbf{m} \not\preceq \mathbf{n}$, then $\mathbf{n}$ and $\mathbf{m}$ are disjoint. Consequently, all operations on individual queue elements are independent of one another.
- A node with children is no longer downward consistent after an application of *reduce*() at step 2c. Hence, consistency must be enforced before the children are added to the queue. This is the reason for step 3a.
- The tree generated by this algorithm is not minimal since we are using *remove*(). Consequently, a sweeping operation is needed to restore minimality after the algorithm runs.
- A node enters the final partition if it either reaches a minimal size, or it can be shown to be fully contained in the solution set.
- If we drop step 4 from *remove*(), the algorithm continues to perform correctly, though the trees generated are not in reduced from. When following a tree this may result in wasted computation since a parent and a single child must be identical in a minimal consistent tree.

4 Data Selection

In this section, we introduce some ideas based on systems of linear inequalities, and use these ideas to summarize some results that lead to a method for choosing a subset of the available data to be used in *reduce*(). The extended version of this report [Hager, 1990] proves that the set chosen by this procedure is optimal for the *reduce*() operation presented the previous section when applied to linear systems. We describe how these results extend to the nonlinear case, and present simulation trials the indicate the procedure is very close to optimal in the nonlinear case.

For notational convenience, we extend the usual comparison operators, $\leq$ and $\geq$ to vectors in componentwise fashion. Then for linear, explicit, additive systems, the consistency test used in *reduce*() can be written

$$z \in H\mathbf{p} + \mathbf{v}$$

which can be rewritten as

$$\underline{H\mathbf{p}} + \underline{\mathbf{v}} \leq z \leq \overline{H\mathbf{p}} + \overline{\mathbf{v}}. \tag{6}$$

The original expression can also be written as

$$H\mathbf{p} \cap (z - \mathbf{v})$$

which in turn can be written using vector comparison as the conjunction of two constraints:

$$\begin{aligned} \overline{H\mathbf{p}} &\geq \underline{z - \mathbf{v}} \\ \underline{H\mathbf{p}} &\leq \overline{z - \mathbf{v}}. \end{aligned} \tag{7}$$

These tests are a conjunction of individual scalar linear constraints. If we let h_i denote the ith row of H, then (6) on an interval $\mathbf{p}$ becomes

$$\underline{h_i\mathbf{p}} + \underline{\mathbf{v}_i} \leq z_i \leq \overline{h_i\mathbf{p}} + \overline{\mathbf{v}_i}, \quad i = 1 \ldots k.$$

4.1 Some Facts About Systems of Linear Constraints

An affine constraint $z \leq hp + v$, $p \in \Re^s$ can be thought of as a tuple of the form $\langle h, z - v \rangle \in \Re^{s+1}$. An affine constraint $z \geq hp - v$, $p \in \Re^s$ becomes $\langle -h, -(z + v) \rangle$ and an equality can be expressed as the conjunction of two inequality constraints. Consequently, a tuple $\langle h, \alpha \rangle = c \in \Re^{s+1}$ defines a half space in $\Re^s$, $-c$ defines the dual halfspace, and the intersection of these two spaces is the hyperplane $\{p \,|\, \alpha = hp\} \subseteq \Re^s$. Henceforth, $\mathcal{C}$ will denote a set of affine inequality constraints $c_i \in \Re^{s+1}$, $i = 1, 2, \ldots, k$. Furthermore, let a_d be the unit vector in coordinate direction d. Then the constraint $u_d = \langle a_d, \alpha \rangle$ describes a halfspace defined by a plane perpendicular to coordinate axis d. Consequently, an interval, $\mathbf{n}$, can be represented by the $2s$ linear constraints

$$\mathcal{C}_{\mathbf{n}} = \bigcup_{d=1}^{s} \{\langle a_d, \underline{\mathbf{n}} \rangle, \langle -a_d, -\overline{\mathbf{n}} \rangle\}.$$

In the sequel, $\mathcal{C}_{\mathbf{n}}$ will denote the linear constraints corresponding to the interval $\mathbf{n}$.

We will say a point $p \in \Re^s$ *satisfies* a constraint $c = \langle h, \alpha \rangle$ if and only if $\alpha \leq hp$. A point p satisfies a set of constraints, $\mathcal{C}$, written $p \models \mathcal{C}$, if and only if p satisfies every $c \in \mathcal{C}$. Given two sets of constraints $\mathcal{C}_1$ and $\mathcal{C}_2$, we say $\mathcal{C}_1$ *is a consequence of* $\mathcal{C}_2$, written $\mathcal{C}_2 \Rightarrow \mathcal{C}_1$ if and only if for all p such that $p \models \mathcal{C}_2$, it is also the case that $p \models \mathcal{C}_1$. Note that any subset of a given set of constraints is a consequence of the original set, that is $\mathcal{C} \Rightarrow \mathcal{C}' \subset \mathcal{C}$. We will call a set of constraints, $\mathcal{C}$, *minimal* if and only if there is no $c \in \mathcal{C}$ such that $\mathcal{C} - \{c\} \Rightarrow \mathcal{C}$.

From the form of our constraint systems, we see that each pair of constraints representing an interval test defines a closed convex strip of space between two hyperplanes in $\Re^s$ and a set of constraints $\mathcal{C}$ defines an s-dimensional *convex polytope* $\mathcal{S}_\mathcal{C} = \{p \mid p \models \mathcal{C}\}$. The goal of the bisection method is to represent this polytope to a given degree of accuracy using intervals.

We define the *minimal interval*, $\mathbf{n}$, enclosing a polytope $\mathcal{C}$ as an interval set of constraints $\mathcal{C}_\mathbf{n}$ such that

- $\mathcal{C} \Rightarrow \mathcal{C}_\mathbf{n}$.
- $\mathcal{C}_\mathbf{n}$ is minimal.
- For any other interval $\mathcal{C}_\mathbf{m}$ such that $\mathcal{C} \Rightarrow \mathcal{C}_\mathbf{m}$, $\mathcal{C}_\mathbf{n} \Rightarrow \mathcal{C}_\mathbf{m}$.

We note that it follows directly from the above definitions that if $\mathcal{C}_\mathbf{n}$ and $\mathcal{C}'_\mathbf{n}$ are the minimal intervals enclosing $\mathcal{C}$ and $\mathcal{C}'$ respectively, and $\mathcal{C} \Rightarrow \mathcal{C}'$, then $\mathcal{C}_\mathbf{n} \Rightarrow \mathcal{C}'_\mathbf{n}$.

If $\mathcal{C}$ describes a closed, bounded, polytope, the minimal interval bounding $\mathcal{C}$ is clearly unique and contains $2s$ constraints. Furthermore, we can place bounds on the maximal size of a set $\mathcal{C}' \subseteq \mathcal{C}$ such that the minimal interval surrounding $\mathcal{C}'$ is identical to that surrounding $\mathcal{C}$. Using basic results in convexity [Rockafellar, 1970, pg. 160] we can obtain the following result:

Result 4.1 If $\mathcal{C}_\mathbf{n}$ is the minimal enclosing interval of a set of constraints $\mathcal{C}$, there is a $\mathcal{C}' \subseteq \mathcal{C}$ such that

- $|\mathcal{C}'| \leq 2s^2$, and
- $\mathcal{C}' \Rightarrow \mathcal{C}_\mathbf{n}$.

In other words, the minimal enclosing interval can be defined based on no more than $2s^2$ constraints from $\mathcal{C}$.

$2s^2$ constraints are needed in the "worst case" where each face of the minimal enclosing interval is determined by a disjoint set of s constraints. This number is larger than needed for the version of *reduce*() we have described. The following statements formalize this idea:

Definition 4.1 A constraint $c \in \mathcal{C}$ *dominates* a constraint $d \in \mathcal{C}$ on an interval $\mathcal{C}_\mathbf{n}$ (written $c_1 \, \mathrm{dom}_\mathbf{n} \, c_2$) if and only if

$$(\mathcal{C}_\mathbf{n} - \{d\}) \cup \{c\} \Rightarrow (\mathcal{C}_\mathbf{n} - \{c\}) \cup \{d\}$$

Geometrically speaking, dominance on an interval means that the polytope defined by the dominating constraint and the base interval is contained within the polytope formed by the dominated constraint and the base interval (constraint b of Figure 1). The set subtraction is included so that we can talk of constraints dominating a side of the enclosing interval in a natural

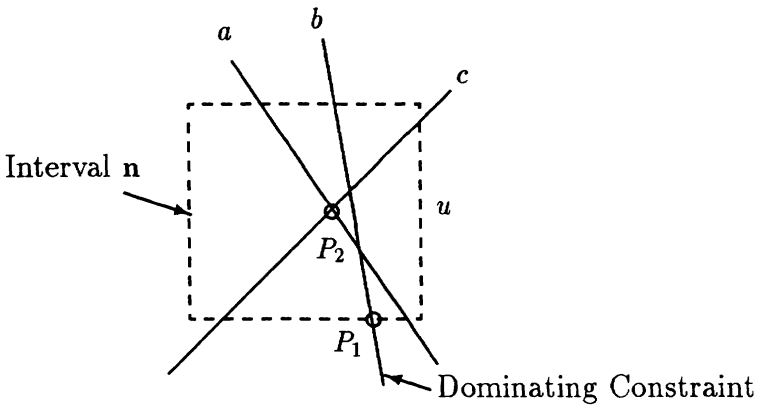

Figure 1. The constraint b dominates u on the interval $\mathbf{n}$. *reduce*() would be able to move the face u to the right until it touched point P_1. However, the minimal interval would require moving the face u until it touched point P_2 at the intersection of a and c.

fashion. We note that by this definition there is always a dominator of a coordinate hyperplane u, though it may be that hyperplane itself.

The crucial property of dominators is expressed in the following:

Result 4.2 The procedure *reduce*() applied to a set of constraints C on an interval $\mathcal{C}_\mathbf{n}$ is only effective if there are constraints $c_i \in C$ which dominate faces $u_j \in \mathcal{C}_\mathbf{n}$ on $\mathbf{n}$.

Effectively, we can choose a constraint for each "face" of the interval $\mathbf{n}$, that is, only $2s$ constraints, and use this subset in *reduce*(). Based on our previous analysis, this means that reduce performs $6s^2$ interval projections and tests. This simplicity comes at a cost, however. As shown in Figure 1, *reduce*() does not always compute the minimal interval, in this case formed by the conjunction of constraints a and c. For a proof that generalized bisection using *reduce*() converges, we refer to [Hager, 1990].

4.2 Choosing Constraints

We first note that, given a constraint c and an interval $\mathcal{C}_\mathbf{n}$ it is possible to determine the intersection of the hyperplane defined by the constraint interpreted as an equality and the "walls" of the surrounding interval. If we choose a particular dimension, then we can look at the minimal and maximal values assumed by the parameter in that dimension on the plane within the interval. Depending on the "direction" of the halfspace, the minimal or maximal value determines the minimal or maximal value of that hyperplane of the minimal enclosing interval, respectively. Using this information, we can determine the strongest set of constraints.

We now proceed to formulate this mathematically and algorithmically. Given the equation of a hyperplane, $z - h_i p - v = 0$, we can solve for the parameter p_j in terms of the rest of the parameter vector as

$$p_j = \frac{z - v - \sum_{k=1,k\neq j}^{s} h_{i,k}p_k}{h_{i,j}} = \frac{z - v - h_i p + h_{i,j}p_j}{h_{i,j}}, \quad h_{i,j} \neq 0.$$

Then on an interval $\mathbf{p}$, the extreme values of p_d are given by

$$\mathbf{r}_{i,j} = \frac{z - \mathbf{v}_i - h_i\mathbf{p} + h_{i,j}\mathbf{p}_j}{h_{i,j}}, \quad h_{i,j} \neq 0.$$

We adopt the convention that when $h_{i,j} = 0$ the interval ranges from $-\infty$ and ∞.

We now note the following facts without proof for constraints of the form $\langle \alpha, h \rangle$:

1. If c_i dominates c_j in a direction a_d, then $\overline{\mathbf{r}}_{i,d} \leq \overline{\mathbf{r}}_{j,d}$.
2. If c_i dominates u_d and c_j does not dominate u_d, then $\overline{\mathbf{r}}_{i,d} \leq \overline{\mathbf{r}}_{j,d}$.

For constraints of the form $\langle -\alpha, -h \rangle$, replacing $\leq$ with $\geq$ and $\overline{\mathbf{r}}_{i,j}$ by $\underline{\mathbf{r}}_{i,j}$ results in the dual statements of these facts.

Our choice procedure, choose($\mathbf{n}$), uses the above to choose a set of $2s$ linear constraints as follows:

Algorithm 4.1

1. Compute the interval matrix $\mathbf{r}_{i,j}$, $i = 1 \ldots k$, $j = 1 \ldots s$.
2. Set $\mathcal{C} = \emptyset$.
3. For each $i = 1, \ldots, s$
   (a) $c^l = \arg\max_j \underline{\mathbf{r}}_{i,j}$.
   (b) $c^u = \arg\min_j \overline{\mathbf{r}}_{i,j}$.
   (c) $\mathcal{C} := \mathcal{C} \cup \{c^l, c^u\}$.

We insert this step between 2b and 2c of the bisection algorithm, and modify *reduce*() to use the set $\mathcal{C}$.

This algorithm has order $O(s * m)$. Thus, it has a running time that grows as a linear function (with a low constant, in practice) of the number of constraints present in the system, and similarly scales linearly with the size of the observation vector.

For parallel implementations of reduce, Algorithm 4.1 finds the optimal set of constraints. For serial implementations of reduce it would be possible to design an algorithm which, after choosing a constraint, projects the bounding interval after applying constraint, recomputes $\mathbf{r}_{i,j}$ based on the new bounding interval, and makes the next choice based on these updated matrices. However, this procedure requires somewhat more computation and also relies heavily on the linear structure of the system description. If the linear constraints are the result of linearizing a nonlinear description, this heavy reliance on linearity may make the procedure more unstable than that given above.

4.3 The Nonlinear Case

We note that Definition 4.1 and Result 4.2 are *independent* of linearity. Thus, the notions of dominance, and the characterization of when $2s$ constraints can be chosen all follow exactly as presented above using nonlinear constraints resulting from a nonlinear sensor-model description.

To implement data selection, we linearize the nonlinear form $g'(\cdot)$ by taking the first two terms of a Taylor series expansion about the center point of an interval, and use the resulting affine form as an approximate linear description of the system of equations. We must point out, however, that the correct functioning of the selection algorithm for an implicit form $g^*(p, z, d) = 0$ requires viewing the function as $f(u) = g^*(p, z, d) = c$ and carrying out the expansion with respect to u. That is, the expansion is with respect to *both* p and z as well as components of d which have any uncertainty associated with them.

4.4 Some Test Cases

We have experimented with a number of test problems of varying complexity (ranging from 2 to 26 parameter dimensions). In these trials we choose the node on the queue with the largest volume as the node to expand, and bisect the dimension for which the Jacobian value multiplied by the interval width is the largest. This results in a fair cycling of nodes and bisection dimensions. We compute the volume of the solution space and compare the rate of decline of volume between the bisection algorithm with data selection and without data selection.

For problems which can be described with explicit equations, the data selection software is fully tested and seems to perform very well. We have tested it on highly overconstrained linear systems and the system performance with and without data selection is identical in every way except that slightly more constraints are marked inactive when no data selection is used. This is to be expected since constraints which would be marked inactive but are dominated by other constraints never enter the bisection procedure when using data selection, and hence are never marked inactive. We have also tested several simple (three or four parameter) nonlinear systems with exactly the same results.

To examine the behavior of data selection in a slightly more complex domain, we have tested its performance on recovery of superellipsoids described using an implicit equation. Our current software does not accommodate Jacobians over parameters other than the model parameters, so in order to test data selection, we have reduced the error of observation to near zero. Data was generated artificially by sampling the superellipsoid in explicit form using a matrix of 64 angles. In this case, there is no point in multiple samples (the data are identical), through we allow two iterations to test the effects of reinitializing at the root of tree. We varied the number of iterations in each case, but by looking at the location of the spike (indicating the return to the root of the tree), it is possible to determine the number used in each trial.

Figure 2 shows the results of several trials. In each, we isolated different parameter groups and compared rates of convergence. Each is a log plot, and the lower (dotted) curves represent convergence when using no data selection while the upper (solid) curves represent convergence using data selection. Again, these are curves of the volume of the solution space. In this case, we see that when recovering position and size parameters there is virtually *no* difference between using all available data and using selected data. Only when determining shape do we see some slight divergence between the two curves. These parameters are highly nonlinear, and the Jacobian is therefore very unstable. Consequently, the data selection algorithm makes an occasional wrong choice, and a node is bisected rather than reduced. This increases the number

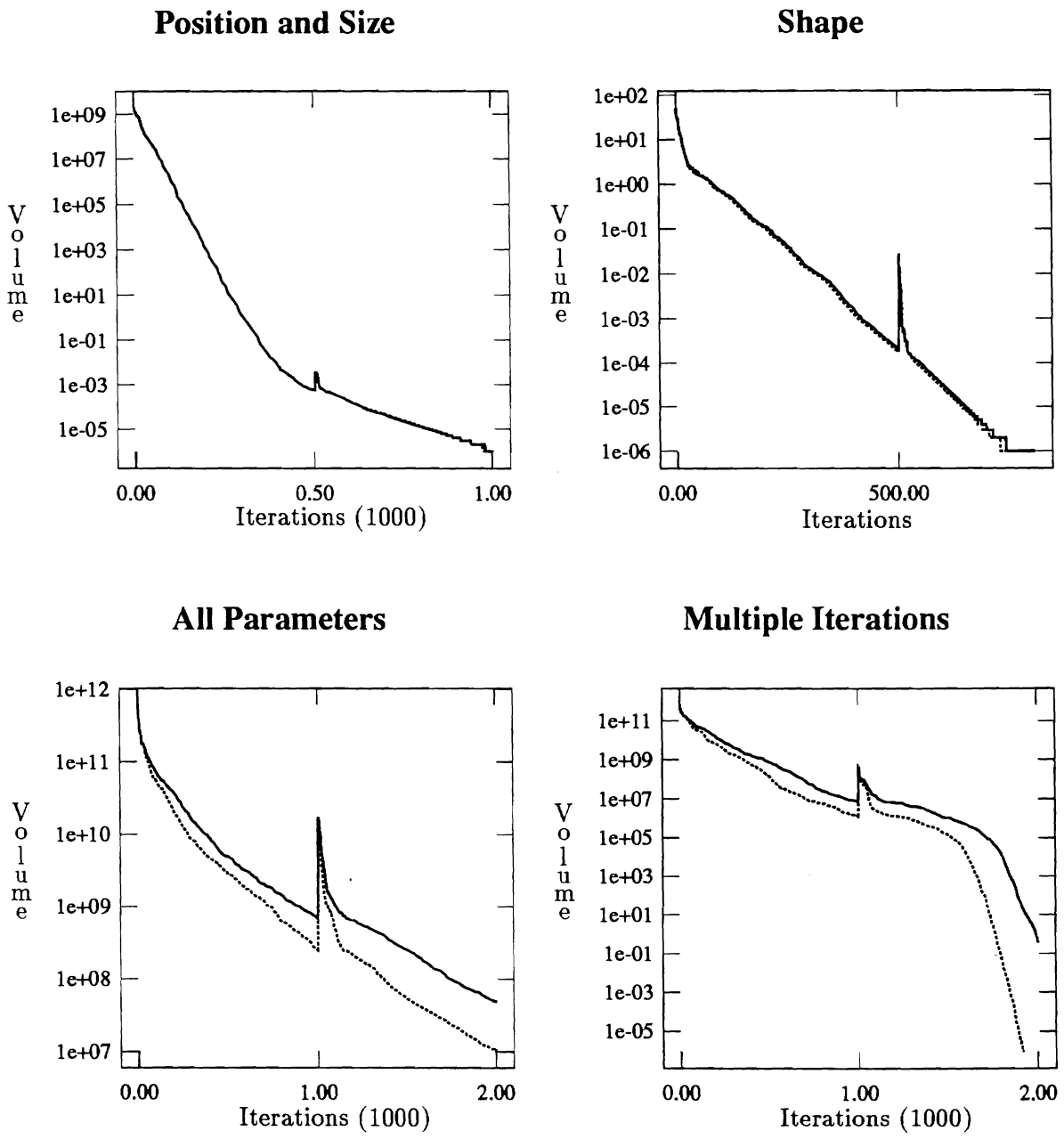

Figure 2. Convergence curves when determining superellipsoid parameters for various parameters subsets. The plot show the volume as a function of *reduce*() applications. The dotted line is when no data selection used, and the dotted line is for an algorithm using data selection.

Operation	Min Time		Average Time		Max Time	
Sel/No Sel	Sel	No Sel	Sel	No Sel	Sel	No Sel
Trisection	0.02	0.13	0.19	0.78	0.50	1.83
Choice	0.13	NA	0.18	NA	0.25	NA
Total	0.20	0.15	0.38	0.79	0.70	1.83

Table 1. Timing figures for two versions of the bisection algorithm. Note that the final row includes some overhead for operations not listed in the table.

of leaves in the tree and reduces the number of times an individual node is processed.

These errors are clearly seen in the final graph where we show convergence when recovering all 8 parameters. In this case, the difference in solution set volume is approximately one half order of magnitude. By choosing more than the minimal number of constraints for these dimensions, the difference between these curves could be reduced.

The simulations described above were carried out using a modified form of *reduce*() that allowed only one interval reduction per application. Reductions of an interval change the optimal set of constraints, so this modification ensures that the selection algorithm gets a "fair chance" to reselect data every time the interval size changes. In this sense, the trials are slightly biased toward the data selection algorithm. The final graph (lower right) shows the effect of allowing *reduce*() to iterate through all dimensions. We see that the difference between using all of the available data and using selected data is still small, and convergence is much more rapid.

When comparing execution time, there is a clear savings in computation by doing data selection. Table Table 1 shows the time to choose data, the time to perform *reduce*() and the total time to expand a node. We see that in the average case, using data selection halves the amount of time needed to expand a node. This performance gain increases with increasing data set size. Our feeling is that in most cases, any loss in rate of convergence is more than made up by this increase in execution speed.

5 An Application of Set-Based Estimation

In this section we describe an application of set-based estimation. The task is to estimate the location of an object described as a superellipsoid to the precision required to place a gripper on the object without collision. This is a member of a collection of tasks which correspond to the well-known "peg-in-hole" problem. Abstractly, the problem is to estimate the location of an object, fitting, hole, or other geometric entity to the precision required to mate with the object within a given tolerance.

We show the results of applying the generalized bisection algorithm to this problem in simulation, and describe an implementation that uses range data from two different range sources.

5.1 Task Formulation

Abstractly, we can pose the problem of capturing an object using configuration space. That is, we shrink the size of the gripper opening by the size of the object and consider the problem of capturing a point in the reduced opening. In one dimension we define a capture predicate u for

an object of size $2s_o$ and location l_o, and a gripper of size $2s_g$ with location l_g as

$$u(l_o, s_o, l_g, s_g) = \begin{cases} 1 & \text{if } l_o \in [l_g - s_g + s_o, l_g + s_g - s_o] \\ 0 & \text{otherwise} \end{cases}$$

with the convention that the interval is empty if the lower bound exceeds the upper bound. The function u returns 1 if and only if the location of the object and the location of the gripper would allow capture without collision. In particular, if $s_o > s_g$, the functions returns 0 for all object and gripper locations indicating the object is not graspable.

We now assume that l_o and s_o are independent random variables with known distributions π_l and π_s respectively, and that we have an estimate l^* for object location. The gripper will be placed so that $l_g = l^*$. For fixed s_o, taking expectations of the above yields

$$R(s_o) = E^{\pi_l}[\, u(l_o, s_o, l^*, s_g) \,|\, s_o, l^*, s_g \,] = P(l_o \in [l^* - s_g + s_o, l^* + s_g - s_o]).$$

That is, for fixed size and random object location, the expected value of u is the probability of capturing the object without collision based on the estimated location l^*. If the capture interval is nonempty, this is simply $R(s_o) = \pi_l(l^* + s_g - s_o) - \pi_l(l^* - s_g + s_o)$. If we now consider size to be a random variable, the payoff u^* can be calculated as

$$u^* = E^{\pi_s}[\, R(s_o) \,].$$

u^* can be thought of as the probability that the proposition "object will be captured" is true given what is currently known.

We now view each interval of the partition generated by bisection as having some probability λ of capturing the model parameters, and assume this probability is distributed uniformly within the interval (we refer the reader to [Hager, 1990] for an extended discussion of how these probability values are calculated). Hence, we can calculate u^* locally on an interval by taking π_l and π_s to be uniform distributions. The results is a value ranging from 0 to 1 indicating the probability of capture for parameters in this interval. To compute a global payoff we compute a local payoff value for each interval and take the sum these values weighted by their associated probability.

Since grasping could be done along coordinate x or coordinate y, we take the *or* of the two propositions "object will be captured along dimension x" and "object will be captured along dimension y." In terms of probability, this means that we compute u_x^* using location and size parameters in x, and and u_y^* using location and size parameters in y, and then take the maximum of these values.

5.2 Simulation Analysis

We have implemented and tested this particular task on superellipsoids with the $s_g = 50$mm corresponding approximately to the opening of a hand in our lab. The initial range of values for superellipsoid location parameters was $\pm$50mm, for size 20 to 140mm, and for shape 0.1 to 1.0. We allowed the estimator to run until the payoff value reached one, indicating a probability one solution for successfully acquiring the object, or zero indicating that the object was to large to be grasped. Estimates of object location were made by taking the centroid of the solution set. The following table shows the results of simulated runs for several different sized spherical objects.

Object Size	90	85	70	60	55	45	40	35	30
Iterations (W Shape)	2	8	18	164	180	290	259	455	674
Iterations (WO Shape)	2	8	18	28	36	69	70	94	148

Table 2. Number of iterations to decide graspability of an object as a function of object size.

The main points to notice are that, as one would expect, objects far larger than what is graspable are decided very quickly without exploiting the supplied information to any great extent. What is perhaps surprising is that the same statement is not true for objects smaller than the given 5 centimeter bound. This is so for two reasons: the effect of sensor observation error is larger on small objects, and the parameter sensitivities lead the method to spend a greater amount of effort recovering shape parameters. In the final row of Table 2, we show the results if we hold the superellipsoid shape parameters fixed. Here we see that the number of iterations is far smaller. One of our future research projects is to make the bisection search process task sensitive so that these differences can be reduced.

5.3 Experimental Apparatus

The application described above has been also been implemented for real laser scanner data. We use a different model description since, as we know the spatial location of the laser scanner cells, it is possible to solve the explicit form of the superellipsoid for the two angles η and ω as a function of scanner cell location, and consequently we can write an explicit expression describing the object sensor relationship that only depends on location size and shape parameters. Details of this formulation can be found in [Hager, 1990].

We use two different laser scanner systems. The first is a linear stage which returns a frame buffer of depth information [Tsikos, 1987], and the second is a scanner mounted on a robot arm which returns a single scanner stripe. In the first case we cannot process all of the data in one step, so we choose a large search window and select 100 points from the data in that window. Using the formulation developed above, we decide to resample when the superellipsoid has been localized so that is can be captured in a window one half the area of the previous window. This continues until a decision about the original task can be reached.

The mobile scanner allows us to process a higher density of data locally, but it must often *actively* sample to acquire enough data to reach a decision. We use the middle of the tolerance interval on model position to choose a sampling point, and vary rotations from 0 to 90 degrees to get a complete "picture" of the object. Of course, it sometimes happens that the first scan line provides enough information to decide whether the object is graspable, and from where. Two are often enough to decide that it is not graspable.

We have only begun testing this version of the program on real data. In particular, the mobile scanner still suffers from calibration inaccuracies. In the next months we expect to improve the system performance and increase its reliability.

6 Conclusions

We have described an algorithm for approximating the set of parametric descriptions of an object from sensor data with known, bounded error. One of the most important features of this algorithm is that the *reduce*() operation applied to a problem with model dimension s can be implemented in parallel using $6s^2$ interval processors. This suggests a pipelined architecture consisting of a data selection processor and an interval reduction processor could yield very fast performance (we estimate less than a millisecond to execute *reduce*() for the superellipsoid problems we have described). For very large data sets, the data selection processor might become a bottleneck, but we speculate that an additional coarse data selection algorithm could be used to "sift" data into different branches of the solution tree.

We are currently working on the applications described in the final section to verify that these methods can function reliably and effectively on real data. In the future, we foresee several improvements to the methods we have described. Briefly:

1. We have begun to fold task constraints into the bisection process. First, the bisection procedure we have described builds the tree based solely on parameter sensitivities. However, if a particular task framework is known, it is possible to direct the search so as to fulfill the given task objectives as quickly as possible.

2. It is often possible to quickly detect that a high quality decision will require extensive computational effort. This information can be used to either terminate the search, saving time for other more easily decided tasks, or for deciding to gather more information that allows the given question to be decided more easily.

3. The formulation we have described assumes that the only discrepancy between observed data and the model is due to sensor error. However, it is often the case that the observed object is only approximated by the supplied model, and extra geometric error must be tolerated. We would like to reflect this additional "geometric uncertainty" back on the model parameters in some principled fashion.

We have made initial steps on all three points, and refer the reader to [Hager, 1990] for their current status.

We believe that set-based estimation or decision-making techniques have a large role to play in the construction of intelligent systems that must make decisions based on sensor data. The application we have described in the previous section is just one of a number of common operations in robotics which could be conveniently cast using set-based techniques. Some of our recent work supports this view in a very practical setting [Atiya & Hager, 1990]. Consequently, we believe that the analysis and understanding of methods such as the one we have presented will have a substantial impact on sensor-based robotics.

Acknowledgements: The following funding agencies supported this work: DARPA Grant N0014-88-K-0630 (administered by ONR), AFOSR Grants 88-0244, AFOSR 88-0296; Army/DAAL 03-89-C-0031PRI; NSF Grants CISE/CDA 88-22719, IRI 89-06770; and Du Pont Corporation. The author would like to thank Jerome Kodjabachian for reading earlier drafts of this paper.

References

Adams, E. 1980. *Interval Mathematics*, Academic Press: New York.

Alefeld, G. and Herzberger, J. 1983. *Introduction to Interval Computations.* Academic Press: New York.

Atiya, S. and Hager, G. 1990. Experiments with a real-time vision-based robot navigation. Submitted for publication in the 1991 IEEE conference on Robotics and Automation.

Eiger, A., Sikorski, K. and Stetnger, F. 1984. A bisection method for systems of nonlinear equations. *ACM Transactions on Mathematical Software*, 10(4):367–377.

Hager, G. 1990. *Computational Methods for Sensor Data Fusion and Sensor Planning.* Kluwer: Boston.

Hager, G. and Mintz, M. 1989. Computational methods for task-directed sensor data fusion and sensor planning. To appear in the International Journal of Robotics Research.

Hager, G. D. 1990. The use of interval-based bisection methods for sensor data fusion. Technical report in preparation.

Kearfott, R. B. 1987. Abstract generalized bisection and a cost bound. *Mathematics of Computation*, 49(179):187–202.

Kearfott, R. B. 1990. Preconditioners for the interval Gauss-Seidel method. *SIAM Journal of Numerical Analysis*, 27(3).

Kearfott, R. B. 1987. Some tests of generalized bisection. *ACM Transactions on Mathematical Software*, 13(7):197–220.

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

Nickel, K., editor, 1980. *Interval Mathematics 1980.* Academic Press: New York.

Nickel, K., editor, 1985. *Interval Mathematics 1985.* Volume 212 of *Lecture Notes in Computer Science*, Springer-Verlag: New York.

Pentland, A. 1986. Perceptual organization and the representation of natural form. *Artificial Intelligence*, 28(3):293–332.

Rockafellar, R. T. 1970. *Convex Analysis.* Princeton University Press: Princeton, NJ.

Sikorski, K. 1982. Bisection is optimal. *Numerische Mathematik*, 40:111–1117.

Solina, F. 1987. *Shape Recovery and Segmentation with Deformable Part Models.* PhD thesis, University of Pennsylvania, Philadelphia. Available as Dept. of Computer Science report MS-CIS-87-93.

Solina, F. and Bajcsy, R. 1990. Recovery of parametric models from range images: the case for superquadrics with global deformations. *IEEE Trans. Pattern Analysis Machine Intelligence*, 12(2):131–147.

Tsikos, C. I. 1987. *Segmentation of 3-D Scenes Using Multi-Modal Interaction Between Machine Vision and Programmable, Mechanical Scene Manipulation.* PhD thesis, University of Pennsylvania.