October 1990

Deciding Not to Decide Using Resource-Bounded Sensing

Greg Hager
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-78.

Deciding Not to Decide Using Resource-Bounded Sensing

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. In previous work we have described a procedure for computing the *solution set* of parametric equations describing a sensor-object imaging relationship, and also discussed the use of task-specific information to support set-based decision-making methods.

In this paper, we investigate the implications of allowing one of the decision-making options to be "no decision," whereupon a human might be called to aid or interact with the system. In particular, this type of capability supports the construction of supervised or partially autonomous systems. We discuss how such situations might arise and give concrete examples of how a system might reach such a decision using our techniques.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-90-78.

Deciding Not to Decide Using Resource-Bounded Sensing

MS-CIS-90-78
GRASP LAB 239

Greg Hager

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

October 1990

Deciding Not to Decide Using Resource-Bounded Sensing

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

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. In previous work we have described a procedure for computing the *solution set* of parametric equations describing a sensor-object imaging relationship, and also discussed the use of task-specific information to support set-based decision-making methods.

In this paper, we investigate the implications of allowing one of the decision-making options to be "no decision," whereupon a human might be called to aid or interact with the system. In particular, this type of capability supports the construction of supervised or partially autonomous systems. We discuss how such situations might arise and give concrete examples of how a system might reach such a decision using our techniques.

1 Introduction

A central problem in sensor data fusion is the reduction or transformation of data to a canonical form suitable for reasoning, acting, or other types of decision making. An important aspect of any method for modeling data is a description of how *accurate* a model is or, conversely, what range of model variation is possible relative to a series of observations.

There are many factors which lead to uncertainty about data models. Even in tightly controlled environments, sensor data will have limited accuracy, may be contaminated by random errors, and may occasionally be completely unreliable. The effect of these errors can often be reduced by using more data. If the data is structurally incomplete, then some aspects of the model cannot be fixed, and again more data is required to fully constrain the model. Finally, the choice of modeling primitive is itself *a priori* information. In uncontrolled environments, it may not be possible to choose a modeling structure that is guaranteed to be adequate for all situations likely to be encountered. Consequently, incorrect decisions may be made due to *data model inadequacy.* This may be thought of as another type of uncertainty—namely uncertainty due to an inadequate description language. Furthermore, *this* type of uncertainty cannot be reduced by acquiring and processing more data, but instead requires some type of model revision.

Uncertainty about a data model does not mean that it is of no use; it merely places a limit on the types of decisions than can be made with reliability. Therefore, we assert that what constitutes an *acceptable* level of uncertainty is fundamentally *task specific*. Information about what decisions are to be made and what degree of model accuracy is required to make them provides a benchmark against which a given model instance can be compared and evaluated.

This comparison may also yield clues as to what type of additional data or model refinement would increase the reliability of model-based decisions. However, as indicated above, if the uncertainty is due to model inadequacy, then gathering more data will not reduce model uncertainty. Even when the uncertainty is due to inadequate data, the *cost* of gathering and processing more data may be unacceptably high. In this case, if other "higher-level" (including human) supervision is available, it may be better to refer the problem to this outside agency than to make poor and/or costly decisions.

In effect, this is a step toward "partially autonomous" systems—that is, systems which can perform most operations autonomously, but which turn to a "supervisor" in exceptional cases that exceed their capabilities. For example, [Tsikos, 1987] describes a system for classifying and sorting postal packages. A partially autonomous system would classify all packages which fit neatly into established categories such as "letter," "box," and "tube," and would turn difficult to classify or unidentifiable objects over to a human supervisor. Space robotics and underwater robotics both rely on teleoperation with a significant time lag. Partially autonomous sensor systems would decrease the reliance of the system on the operator by operating autonomously until an exceptional situation is reached, at which point the operator is consulted for further instruction. There are many other examples.

In this paper, we describe a set of techniques that facilitate the use of task-related information to make sensor-based decisions, and discuss how these techniques can be used to decide *not to decide* by referring the decision to a higher level. Our approach is based on parametric descriptions of data similar to those used by standard fitting techniques. However, while standard fitting techniques choose a single *point* estimate to represent the data, we compute the set (referred to as the *solution set*) of models compatible with observed data up to sensing error. Task-specific accuracy criterion are used to determine if the solution set is sufficiently small to support accurate, reliable decision making.

The major assumption we make is that observation error is *bounded*. This allows us to describe the relationship between sensor observations and model parameters in terms of systems of inequalities, and the computation of the solution set as an incremental exploitation of these inequalities. We believe that one of the major contributions of this work is the development of powerful inequality-based methods for describing and examining the properties of data. We show how inequalities allow us to address issues such as data contaminated with outliers, structural deficiencies in the parameter model, and data segmentation in a precise, quantifiable way.

The remainder of this paper is organized as follows. In the next section, we describe how *interval-based* methods are employed to compute a solution set from observed data. We illustrate how these techniques work in the simple case of fitting points to a line. In Section 3, we describe how task-related information is used to reach decisions, and in particular how and when it is decided not to decide. In Section 4 we sketch how these ideas extend to segmentation. In Section 5 we

briefly describe the results of experimental trials, and in Section 6 we review our results and discuss some open problems.

2 Interval-Based Sensor Data Fusion

In this section we describe of how interval-based techniques can be applied to sensor-data fusion problems. This is an abbreviated version of material found in [Hager, 1990b; Hager, 1990c].

2.1 Problem Description

In this paper, we assume the relationship between an observation, z of dimension m, and model parameters, p of dimension s, is described by an implicit function of the form

$$g(p, z + v, d) = 0, \quad p \in \mathcal{P} \subseteq \Re^s, \quad z \in \mathcal{Z} \subseteq \Re^m, \quad d \in \mathcal{D}, \quad v \in \mathcal{V} \subseteq \Re^m \tag{1}$$

where the vector d denotes additional kinematic or physical degrees of freedom of the sensor system (calibration or control parameters), and v is a *nuisance* parameter denoting non-deterministic disturbances of the sensor output.

The sensor data fusion problem is to recover model geometry, 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, we say a data pair $\langle z_i, d_i \rangle$ is *consistent* with a parameter vector p only if $g(p, z_i + v, d_i) = 0$ for some $v \in \mathcal{V}$.

We now state the version of the sensor data fusion problem that 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*
>
> $$\mathcal{P}^* = \{p \in \mathcal{P} \mid p \text{ is consistent with each } \langle z_i, d_i \rangle,\ 1 \leq i \leq n\}.$$

We refer to $\mathcal{P}^*$ as the *solution set* consistent with a series of observations. Generally speaking, solution sets are 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. We now describe the generalized bisection algorithm we have developed for this purpose.

2.2 A Brief Review of Interval Analysis

Our solution to the problem of isolating solution sets makes heavy use of concepts from *interval analysis*. 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 $\Re^s$. 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 $w : [\![\Re]\!]^s \to \Re^s$ by $w(\mathbf{n}) = \overline{\mathbf{n}} - \underline{\mathbf{n}}$; and the center function $c : [\![\Re]\!]^s \to \Re^s$ by $c(\mathbf{n}) = (\overline{\mathbf{n}} + \underline{\mathbf{n}})/2$.

2.3 Interval Functions

Suppose we are given a function $h : \Re^s \to \Re$. 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, $\mathbf{x}$ describing an s-rectangle and wish to compute its projection. For a given continuous function $h : \Re^s \to \Re$, we can define an *interval function*, $\mathbf{h} : [\![\Re]\!]^s \to [\![\Re]\!]$ 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 2.1 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 may 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. For example, suppose $\mathbf{x} = [-1,1]$

and we compute $\mathbf{x} * \mathbf{x}$. The standard interval multiplication 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 directly 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. In subsequent sections, we assume that all interval computations produce the minimal correct interval.

We now 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

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

We note that, in addition to the possibility of overly conservative scalar intervals, if we consider functions with non-scalar range it is usually 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 $\mathbf{H}(\mathbf{m}) \subseteq \mathbf{H}(\mathbf{n})$.

2.4 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.

For an interval tree in $[\![\Re]\!]^s$ with leaf node $\mathbf{n}$, we define the following three operations:

bisect($\mathbf{n}, d$):

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

remove($\mathbf{n}$):

1. If $\mathbf{n}$ is the root of the tree, signal error.
2. Let $\mathbf{p} = \text{parent}(\mathbf{n})$.
3. $D_{\mathbf{p}} := D_{\mathbf{p}} - \{\mathbf{n}\}$.

4. If $|D_{\mathbf{p}}| = 0$, execute *remove*($\mathbf{p}$).

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

1. For each dimension i, $i = 1, \ldots, s$, trisect $\mathbf{n}$ 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 $0 \notin \mathbf{g}(\mathbf{n}_{i,j}, z_i - \mathbf{v}, d_i)$ for some $\langle z_i, d_i \rangle \in O$, then $\mathbf{n}_{i,j} := \emptyset$.
3. $\mathbf{n} := \bigcap_{1 \le i \le s} \bigcup_{1 \le j \le 3} \mathbf{n}_{i,j}$

The function of *reduce*() is to reduce the size of an interval, or eliminate it entirely based on constraints imposed by observed data. 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. Furthermore, in [Hager, 1990b], we have shown that the number of constraints used by *reduce*() at any node is bounded from above by $2s$. Consequently, this operation requires no more than $6s^2$ interval projections.

2.5 Interval Bisection

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. A simplified version of our interval bisection algorithm is:

Algorithm 2.1

generalized-bisection($\mathbf{n}, \mathbf{g}, O$) :

1. *(Initialization)*
   (a) Set a vector of coordinate tolerances, $\epsilon_i, 1 \le i \le s$.
   (b) $\mathcal{Q} := \{\mathbf{n}\}$.
2. *(Reduction)*
   (a) If $\mathcal{Q} = \emptyset$, stop.
   (b) Remove an interval $\mathbf{x}$ from $\mathcal{Q}$.
   (c) Compute *reduce*($\mathbf{x}, \mathbf{g}, O$).
   (d) If $\mathbf{x} = \emptyset$, execute *remove*($\mathbf{x}$) and go to step 2.
   (e) If $\mathrm{w}(\mathbf{x}) \le \epsilon$ then $\mathcal{L} := \mathcal{L} \cup \{\mathbf{x}\}$ and go to step 2.
3. *(Bisection)*
   (a) Choose a dimension $1 \le d \le s$ such that $\mathrm{w}(\mathbf{x})_d \ge \epsilon_d$.

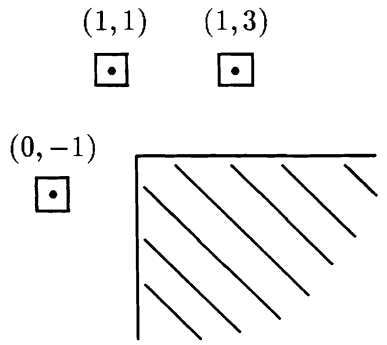

Figure 1. Several points to be fit to a line up to the given tolerance boxes.

(b) Execute *bisect*$(\mathbf{x}, d)$
(c) $\mathcal{Q} := \mathcal{Q} \cup D_{\mathbf{x}}$.
(d) Go to step 2.

At every iteration, the set of intervals $\mathcal{Q} \cup \mathcal{L}$ is a partitioning of the current best approximation to the solution set.

Example 2.2 Suppose we have the points shown in Figure 1 and we wish to determine the parameters of a line through the points. The error in observation is that described by the solid boxes enscribed about the points. The equation of a line is

$$ax + b = y.$$

Including observation error, the compatibility of a point (x, y) with an interval $\mathbf{p} = (\mathbf{a}, \mathbf{b})$ of line parameters is described by the interval expression

$$0 \in \mathbf{g}((\mathbf{a}, \mathbf{b}), (x, y)) = \mathbf{a}(x + \mathbf{v}_x) + \mathbf{b} - y - \mathbf{v}_y.$$

We choose three interval vectors $\mathbf{p_1} = ([-1, 0], [-1, 0])$, $\mathbf{p_2} = ([1, 2], [-1, 0])$, and $\mathbf{p_3} = ([0, 1], [-3, 1])$, and set $\mathbf{v}_x = \mathbf{v}_y = [-1/4, 1/4]$.

If we test $\mathbf{p_1}$ on the point $(0, -1)$, we compute $\mathbf{g}(\mathbf{p_1}, (0, -1)) = [-1/2, 3/2]$ which contains 0 and is judged consistent. If we were to continue bisection and interval evaluation based on this single point, the result would be a collection of intervals falling along a line in (a, b) space. Any parameters outside these intervals would be inconsistent with the observed data. By analogy, in more complex models, inadequate data leads to an envelope of surfaces, all of which are consistent with the observed data.

Now we add the point $(1, 1)$ and compute $\mathbf{g}(\mathbf{p_1}, (1, 1)) = [-7/2, -3/4]$. This interval does not contain 0 and is therefore inconsistent. Hence, this interval can be rejected as it does not describe the observed data. It is easy to check that $\mathbf{p_2}$ is consistent with the

points $(0,1)$ and $(1,1)$, but is inconsistent with $(1,3)$. The interval $\mathbf{p}_3$ is consistent will all three data points.

The interval $\mathbf{p}_3$ describes a *family* of lines consistent with the data up to observation error. This information is already good enough to answer simple questions such as whether the distance of any line in this family to a given point is larger or smaller than a set threshold. However, if we were to ask if the data do or do not suggest that the surface crosses the corner depicted in the figure, we could not answer with certainty. Furthermore, it is obvious from the figure that the data *cannot* be fit to a line consistently; there is no *single* line that passes through all error boxes. Consequently, the bisection procedure will eventually terminate with an empty solution set. For the moment, we will assume that observed data fits the model up to the specified tolerances, and discuss what to do if this is not true in Section 5.

3 Task Formulation and Solution Behavior

A sensing task description provides information about how sensor information is used to make a specific decision, and about the consequences of making an incorrect decision. In [Hager, 1990a], we describe tasks from a very general point of view. In this article, we focus on tasks of the "hit-or-miss" variety, examples of which we have taken from the domain of classifying and manipulating mail pieces. The principal aspect of these tasks is that they introduce geometric constraints on how model parameters are to be used or interpreted. We will describe how task-specific decisions are reached and evaluated based on concepts borrowed from Bayesian decision theory [Berger, 1985]. We note that the use of Bayesian methods is not central to the discussion, and that parallel results could be obtained using worst case analysis or minimax methods.

3.1 Two Problem Families

We consider two types of problems: continuous estimation problems and discrete "classification" problems. Both example problems will be based on *superellipsoidal* data models. The implicit equation for a superellipsoid in standard orientation is:

$$g_{\texttt{isq}}(s,\gamma,l;u) = \left[\left(\frac{u_x-x}{s_1}\right)^{\frac{2}{\gamma_2}} + \left(\frac{u_y-y}{s_2}\right)^{\frac{2}{\gamma_2}}\right]^{\frac{\gamma_2}{\gamma_1}} + \left(\frac{u_z-z}{s_3}\right)^{\frac{2}{\gamma_1}} - 1 = 0.$$

The vector $s = [s_1, s_2, s_3]$ 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. We assume surface points are observed directly using range sensing. We refer the reader to [Solina & Bajcsy, 1990] for a more extensive discussion of superellipsoids and their properties.

Peg-In-Hole Abstractly, the "peg-in-hole" problem is to estimate the location of an object, fitting, hole, or other geometric entity to the precision required to successfully mate it with a second object. In our case, we will determine if a gripper of a given size could encompass an observed

object modeled as a superellipsoid. Conceptually, 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, $l(\cdot)$, for an object of size $2s_o$ and location x_o, and a gripper of fixed size $2s_g$ with location x_g as

$$l(x_g; x_o, s_o) = \begin{cases} 0 & \text{if } x_o \in [x_g - s_g + s_o, x_g + s_g - s_o]; \\ 1 & \text{otherwise} \end{cases}$$

with the convention that the interval is empty if the lower bound exceeds the upper bound. The function l returns 0 if and only if the two locations are close enough to ensure capture without collision. In particular, if $s_o > s_g$, the function returns 1 for all object and gripper locations, thereby indicating the object is not graspable.

Define $S(x; \mathbf{x}_o, \mathbf{s}_o) = \{(x_o, s_o) \in (\mathbf{x}_o, \mathbf{s}_o) \mid l(x; x_o, s_o) = 1\}$, and let the operator $\mu(\cdot)$ denote the volume of a set in $\Re^n$. Then the risk of an interval vector $\mathbf{p} = (\mathbf{x}_o, \mathbf{s}_o)$ can be calculated as

$$R_1(x; \mathbf{x}_o, \mathbf{s}_o) = \frac{\mu(S(x, \mathbf{x}_o, \mathbf{s}_o))}{\mu(\mathbf{p})}.$$

If the model parameters were distributed uniformly in $\mathbf{p}$, $R_1(x; \mathbf{x}_o, \mathbf{s}_o)$ is the probability that the proposition "object will be captured" is false in the interval $\mathbf{p}$.

Rather than blindly choosing a location regardless of whether the object is graspable, we choose between the options "yes" (supplying a grasp location) or "no." The loss of the various decision alternatives can be described in terms of a decision table as

decision\ world	"yes"	"no"
"yes"	0	$l_{y,n}$
"no"	$l_{n,y}$	0

This table is interpreted as follows: if the observed object is not graspable, but the decision "yes" is reached, the loss for this decision is $l_{y,n}$. Conversely, if the observed object is graspable and "no" is decided, the loss is $l_{n,y}$. No loss is incurred for a correct decision.

We redefine the risk function in terms of a joint decision about location and graspability as

$$R(x, d; \mathbf{x}_o, \mathbf{s}_o) = \begin{cases} l_{y,n} R_1(x, \mathbf{x}_o, \mathbf{s}_o) & d = \text{"yes"}; \\ l_{n,y}(1 - R_1(x, \mathbf{x}_o, \mathbf{s}_o)) & d = \text{"no"}. \end{cases}$$

We will allow grasping along either coordinate x or coordinate y, so 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 calculate two risks, $R(x, d_x; \mathbf{x}, \mathbf{s}_1)$ and $R(y, d_y; \mathbf{y}, \mathbf{s}_2)$. At a global level, both alternatives will be evaluated and as we will describe below, and the decision with minimum risk will be chosen.

Classification In a classification problem, we are interested in determining if model parameters support one or more of n "labelings", a_i, $i = 1, \ldots n$, of an object. For example, in the case of

superellipsoids, we can define a predicate

$$\texttt{round}(\gamma) = \begin{cases} 0 & \text{if } \gamma > .9 \\ 1 & \text{otherwise,} \end{cases}$$

a predicate

$$\texttt{square}(\gamma) = \begin{cases} 0 & \text{if } \gamma < .3 \\ 1 & \text{otherwise,} \end{cases}$$

and a predicate

$$\texttt{thin}(s) = \begin{cases} 0 & \text{if } s < 10 \\ 1 & \text{otherwise.} \end{cases}$$

Using these predicates, we can define four composite classifications, of superellipsoids in standard position, `flat()`, `carton()`, `cylinder()`, and `parcel()` as:

$$\begin{aligned} \texttt{flat}(p) &= \texttt{thin}(s_3) \\ \texttt{cylinder}(p) &= \neg\texttt{flat}(p) \wedge (\texttt{round}(\epsilon_2) \wedge \texttt{square}(\epsilon_1)) \\ \texttt{carton}(p) &= \neg\texttt{flat}(p) \wedge (\texttt{square}(\epsilon_1) \wedge \texttt{square}(\epsilon_2)) \\ \texttt{parcel}(p) &= \neg\texttt{flat}(p) \wedge \neg\texttt{cylinder}(p) \wedge \neg\texttt{carton}(p) \end{aligned}$$

We refer a classification as *complete* if all allowed model parameters receive some classification. As this classification is complete and the categories are non-overlapping, we can define a classification function $C(\cdot)$ mapping a parameter vector to a classification.

An error in classification is the mislabelling of an object. If the classification is complete, and the categories are non-overlapping, the loss function for a classification problem can be visualized as a matrix of alternatives of the form:

decision/world	a_1	a_2	...	a_n
a_1	0	$l_{1,2}$	...	$l_{1,n}$
a_2	$l_{2,1}$	0	...	$l_{2,n}$
		$\vdots$		
a_n	$l_{n,1}$	$l_{n,2}$	...	0

By analogy with the previous example, if the observed object should be classified as a_i and the classification decision is a_j, then the loss is $l_{j,i}$. The zeros on the diagonal indicate that there is no penalty for correctly classifying an object.

As above, we define $S(a_i, \mathbf{p}) = \{p \in \mathbf{p} \mid C(p) = a_i\}$, and the risk function $R(a_i, \mathbf{p})$ is given by

$$R(a_i, \mathbf{p}) = \sum_{j=1}^{n} l_{i,j} \frac{\mu(S(a_i, \mathbf{p}))}{\mu(\mathbf{p})}.$$

In the example classification scheme presented above, the predicates are all thresholds on scalar values, so it is simple to construct an interval function that computes the proportion of the scalar interval argument which satisfies the corresponding constraint. The individual classifications are

non-overlapping, so by replacing the simple predicates by the corresponding interval predicate, and replacing $\wedge$ with $*$ and $\vee$ with $+$, we arrive at expressions which compute the proportion of the area of an interval which supports each of the first three classifications. We attribute the remaining volume to the `parcel()` classification.

Global Risk We now view each interval $\mathbf{n}$ of the partition $\mathcal{C} = \mathcal{Q} \cup \mathcal{L}$ generated by bisection as having some probability $\lambda_{\mathbf{n}}$ of capturing the model parameters, and assume this probability is distributed uniformly within the interval (we refer the reader to [Hager, 1990a] for an extended discussion of how these probability values are calculated). To compute a global risk for a decision a_i, we compute a local risk value for each interval and compute the sum of these values weighted by the associated probability, that is

$$R^g(a_i, \mathcal{C}) = \sum_{\mathbf{n} \in \mathcal{C}} \lambda_{\mathbf{n}} R(a_i, \mathbf{n}).$$

The optimal decision, a^* is the a_i with minimal risk. For convenience, we define $R^*(\mathcal{C}) = R^g(a^*, \mathcal{C})$.

3.2 Minimal Cost Decisions

The bisection procedure can be viewed as an iterative operator, F, yielding a sequence of sets $\mathcal{P}_0, \mathcal{P}_1 = F(\mathcal{P}_0), \mathcal{P}_2 = F(\mathcal{P}_1), \ldots$ such that

- $\mathcal{P}_0 \supseteq \mathcal{P}$,
- $\mathcal{P}^* \subseteq \mathcal{P}_{k+1} \subseteq \mathcal{P}_k$,
- $\mathcal{P}_k \rightarrow \mathcal{P}^*$

where $\mathcal{P}^*$ is the solution set defined in Section 2.

If $\mathcal{P}_k$ is represented by an interval partition $\mathcal{C}_k$, then $\mathcal{P}_k$ has risk value $R^*(\mathcal{C}_k)$. Furthermore, if each processing step has some fixed cost c, then the computational cost of computing the kth solution set is ck. Assuming decision losses represent the same type of cost, we wish to process information to the time step k^* such that

$$R^*(\mathcal{C}_{k^*}) + ck^* = \min_k R^*(\mathcal{C}_k) + ck.$$

That is, we wish to minimize the combined cost of decision error and computation.

To solve this expression in the general case requires the ability to *predict* the effect of further processing in an effective fashion, a problem that is usually quite difficult to solve. However, if the generalized bisection procedure constructs a sequence of solution sets such that $R^*(\cdot)$ is a convex function with respect to number of iterations, then the number of iterations can be governed by stopping when the condition

$$dr = R^*(\mathcal{C}_k) - R^*(\mathcal{C}_{k-1}) < c.$$

is true. No prediction is required, so this test costs nothing more than a single step past the ideal solution. In particular, if we have bounds on $R^*(\cdot)$, usually indicating a sure success or failure, then processing will stop when that bound is reached.

3.3 The Effect of Adding "No-Decision"

In both problems described above, we defined a decision matrix describing the consequences of incorrect decisions. The assumption is that the decision alternatives cover all possibilities, and the consequences of decision alternatives are known.

Not making a decision is to forgo taking any action whatsoever. The consequences of this action are to incur a fixed cost related to the effects of deferring or referring a decision. Given a decision table for actions a_i, $i = 1, n$, we can add the distinguished action na as follows:

decision\ world	a_1	a_2	...	a_n
a_1	0	$l_{1,2}$	...	$l_{1,n}$
a_2	$l_{2,1}$	0	...	$l_{2,n}$
		$\vdots$		
a_n	$l_{n,1}$	$l_{n,2}$	...	0
na	c_{na}	c_{na}	...	c_{na}

The effect of introducing the "no action" decision is to introduce a *fixed cost* c_{na} which is incurred *no matter what* if no decision is made. It follows directly that the na decision will be made only if this decision is likely to be less costly than at least one of the other decision alternatives. We note that if the decision table is not a complete classification of the parameter space, then there is an implicit "no action" alternative built into the decision process. In this case, the table above would be modified to include a final column labelled na containing values representative of the consequences of classifying an unclassifiable situation into each of the other categories.

We now assume that the decision-making problem has some minimal risk b; in our examples, $b = 0$. If the risk level at step k is r, the rate of change of r is dr, and the risk function is a convex function of iterations, then in the best case, we will reach the risk level b in $k^* = (b - r)/dr$ time steps. The additional cost of computing to this level is ck^*, and for all $i < k$, $R^*(C_{k+i}) + ci > ck^*$. Then, if $ck^* + b > c_{na}$, it is less costly to decide na then to continue computing. Note that this is true even though it may be the case that $b < c_{na}$. In other words, even in the best of all worlds, the resource expenditure associated with rendering no decision is less than the expenditure required to compute a perfect decision.

4 Experimental Results

All of the techniques and examples described to this point have been implemented and tested in simulation and on real data. We briefly summarize our experimental results here, and refer the reader to [Hager, 1990c] for a more extensive description.

In simulation, we uniformly sample a superellipsoid to compute 150 synthetic range points, add noise to these points, and run the bisection algorithm on the resulting data. Our method for handling outliers, a common occurrence in sensor data, is to increase the number of data points which must be incompatible with an interval projection in order to reject an interval. We set this

threshold to values ranging from 5% to 10% of the data. It is assumed that all other range readings are no more than ± 3.0mm in error. Under these conditions, all of the problems we have described are uniquely solvable in less than 1000 iterations. In some cases, such as recognizing that an object is flat, fewer than 10 iterations suffice. When the number of iterations is limited, either at a fixed value or by using the cost constraint introduced by *na*, the problems resulting in *na* are usually those requiring shape information.

Experimentally, we acquire range data by shining a laser on a surface and computing the distance to the surface via triangulation with a fixed camera [Tsikos, 1987]. We have tested two systems: a fixed scanner mounted above a linear stage, and a mobile system mounted on a robot arm. The latter system is more flexible, but it is still in the testing stage. Therefore we will confine our comments to the fixed scanner.

The fixed scanner system collects a frame buffer of information which we threshold to remove any background points. We uniformly sample the remaining data, choose 150 uniformly distributed points, and run the recovery procedure with the error parameters stated above. The primary difference between the simulations and recovery from real data is that real data is often inadequate to recover model structure to the precision of the simulation. However, this is not a failure of the recovery methods. Rather, due to shadowing, the laser scanner returns a hemisphere of data at best. This data is inadequate to fully determine the model structure, and the method accurately reflects this fact. As in simulation, simple questions such as determining whether an object is flat are answered quickly and correctly. Because of the poor quality of the data, more "no-decision" results are returned than in the comparable simulation. We do not see this as a failure of the method, but rather as proof that the data is inadequate to reliably answer the question posed.

5 Modifications for Inconsistent Data

In Example 2.2 we noted that the three observed points could not be fit to a line up to observation error. However, the interval $\mathbf{p}_3$ was judged consistent with the observed data. This is not a mistake. The interval contains a consistent parameter vector for each data point. If we reduce it in size, we eventually exclude all parameter vectors consistent with some point, at which point the interval is judged inconsistent with the observed data. Note, however, that the rejected interval is still compatible with two of the three points, just as $\mathbf{p}_2$ was compatible with two of the three points.

When global inconsistency occurs, one of the following is true:

1. The sensor has been incorrectly characterized.
2. The data set contains outlier data which should be ignored.
3. The geometric model is insufficient to described the observations.

If we assume the sensor is correctly characterized, and outliers have been accounted for as discussed in the previous section, then the only conclusion is that the model is inadequate. With respect to the example tasks described in Section 3, if no consistent interval is sufficient to arrive at an

acceptable decision, then the system will *automatically* choose the *na* action. In other words, lack of an adequate data model may cause the system to refer the problem to an outside entity.

We can go a step further. If we bisect a consistent interval and determine that both children are inconsistent, we can examine what data points caused the rejection of each interval and *segment* the data into two subsets. Now, we can continue the bisection procedure using the two subsets of data, and implicitly fitting *two* data models. Returning to Example 2.2, by segmenting the data into a set containing point 1 and point 2, and a set containing point 2 and point 3, we can refine the description of the data enough to determine that a surface passing through the points *is* disjoint of the pictured corner.

This is an exciting possibility, as we can now make the segmentation of data *task-dependent.* Furthermore, in cases such as the cited example, decision making on segmented data is no different than decision making on unsegmented data. In both cases we have a set of models, and are evaluating the consequences of decisions relative to that set. This clearly demonstrates the power and flexibility of set-based decision-making techniques.

6 Conclusions

We have presented a set of techniques for sensor data fusion that are based on interval analysis and generalized bisection. Given a sensor description, a parametric model, and observed data, these techniques compute a set of models known to be compatible with observed data. We have shown how a task description can be used to reach decisions and evaluate their reliability. By introducing the decision "no action," these methods can be applied in problem domains where knowledge about the problem domain is incomplete, but supervision is available. This extension is simple, natural, and requires no additional computational apparatus. We have briefly described the results of applying these techniques to real data.

There are still several problems with the methods we have described. For complex models, particularly when the data is structurally inadequate, convergence can be unacceptably slow. This is partially due to the simplicity of the *reduce*() procedure, and partially due to the use of intervals as the basic representation primitive. We are currently investigating what types of constraints lead to fast convergence, as well as looking into alternative implementations of *reduce*(). The segmentation methods described in Section 5 have not been fully implemented and tested, so recovery is still limited to isolated objects.

We believe that the problem area we have described is crucially important to robotics, and that general, effective solutions in this area will have substantial impact on the field. What is perhaps more important than our computational results, is the observation that posing a sensor data fusion problem in terms of constraints can lead to a compact, consistent, precise language for describing, analyzing, and implementing solutions. While many of the issues we have raised, including discarding outlier data, segmentation, and sensor planning, are still difficult problems in this paradigm, the ability to use task-specific information provides a means of constraining and evaluating solutions that does not otherwise exist. The primary aim of our future research is to further exploit the use of task-specific information in these problem areas.

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 his helpful comments on an earlier draft of this paper.

References

Adams, E., (1980). On sets of solutions of collections of nonlinear systems in $\Re^n$. In Nickel, K., editor, *Interval Mathematics*, pages 247–256, Academic Press, New York.

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

Berger, J. O., (1985). *Statistical Decision Theory and Bayesian Analysis.* Springer-Verlag, New York, 2nd edition.

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

Hager, G., (1990a). *Computational Methods for Sensor Fusion and Planning.* Kluwer, Boston.

Hager, G. D., (1990b). Interval-based bisection methods for sensor data fusion. Technical report in preparation.

Hager, G. D., (1990c). Interval-based methods for sensor data fusion. Submitted for publication in the 1991 IEEE conference on Robotics and Automation.

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.

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

Solina, F. and R. Bajcsy, (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.