Task-Directed Computation of Qualitative Decisions From Sensor Data

Gregory D. Hager

Research Report YALEU/DCS/RR-921
August 1992

Task-Directed Computation of Qualitative Decisions From Sensor Data.

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

Telephone: (203) 432-6432
E-mail: hager@cs.yale.edu

Abstract

This article describes a novel approach to sensor-based decision making that involves formulating and incrementally solving large systems of parametric constraints. The constraints describe both *a model for sensor data, and the criteria for correct decisions about the data. An incremental constraint solving technique we have developed performs the minimal model recovery required to reach a decision. This method is straightforward to apply, is easily parallelized, and convergence can be demonstrated under very weak structural and statistical assumptions.*

The approach is demonstrated on several different decision-making problems involving manipulation and categorization of objects. Objects are observed with a range scanner and described by a superellipsoid data model. The experiments indicate that simultaneous solution of both model constraints and decision criteria can lead to efficient and effective decision making, even when the observed data does not strongly determine a data model.

Submitted for review to IEEE Transactions on Robotics and Automation.

1 Introduction

A common approach to sensor-based decision making or recognition is to fit observed data to a parametric model, and then to use the computed model parameters as a basis for further decision making or action. Examples of this general paradigm include work based on generalized cylinders [20, 24], superquadrics [4, 12, 28], deformable models [23, 30], and parametric patches of various types [2, 6, 17, 18]. However, much of this work has focussed on the problems of reconstruction for the purpose of recognition. Not all activities need the precision and detail required for recognition. Operations such as grasping, route-finding or planning can often be based on *qualitative decisions* that can be made from a partial, imprecise reconstruction of the world.

To better understand what is meant by a "qualitative" decision, consider two uses of sensor information: moving a robot arm while maintaining a fixed distance and orientation from a surface; and determining whether a mobile robot will fit through an opening. The first task requires constant quantitative knowledge about the world—point estimates of the distance and orientation of the surface—in order to control the arm. The second task does not require a point estimate. For example, suppose that a set of parameterized closed curves contains an element correctly describing the observed opening. If all members of that set are large enough for the robot, then so is the opening in question—a more exact description of the opening is not needed. More generally, this paper considers a *qualitative decision* to be a decision that can be made from a set of parameters known to contain a value describing ground truth. As such, qualitative decisions are essentially propositions on a set of parametric models. The proposition is true if it is satisfied by every parameter vector consistent with observed data.

Although common point-estimation techniques augmented with some measure of fitting error can, in principle, be used to make qualitative decisions, this approach has several drawbacks. Except in certain special circumstances, obtaining point estimates requires an iterative optimization procedure. With such procedures it is practically impossible to determine the accuracy of an intermediate parameter estimate: the procedure must be run until it converges before an error estimate can be computed. Consequently, optimization-based point estimation techniques divorce the decision-making process from the estimation process. As a result, the effort spent in the fitting stage may be inappropriate to the decision to be made: even simple, obvious conclusions may require substantial amounts of computation. Furthermore, quantitative statements about fitting error usually require some knowledge of the sensor error distribution, and often rely on linearizations or other simplifications of the underlying data model. In cases where the final fit is not "good" (*eg.* the observations are not compatible with the model) the error estimates may themselves be significantly in error. Finally, it is often difficult to guarantee the global convergence properties of an optimization algorithm. Hence, even when the optimization process appears to converge, the correctness of the decisions based its results may be called into doubt.

This paper describes a conceptual framework for making qualitative decisions from sensor information that avoids optimization and point estimation entirely. Instead, decision making and model recovery are both described in terms of constraint satisfaction. Decisions are computed by a provably correct and complete numerical constraint solving algorithm. A particularly interesting aspect of this algorithm is that it uses information about the decisions to be reached to control model recovery. As is demonstrated later in this paper, this integration of model recovery and

decision making often drastically reduces the amount of computation required to reach a decision. In some cases, this focusing of effort naturally leads to a type of selective perception: the algorithm focuses on objects or object qualities that are important for the decision under consideration and avoids "wasting" effort computing irrelevant information.

The central assumption underlying these techniques is that sensing errors are *bounded.* A bounded sensor observation constrains model parameters to a set. If this set is guaranteed to contain *all* consistent descriptions for observed data, then it is a conservative, quantitative description of the information the data contains. Adding more constraints to the system reduces the size of this set and permits more precise or detailed knowledge about the world. If the set becomes empty, then the system of constraints is inconsistent—the world does not conform to the model chosen to describe it.

The idea of using bounded errors and constraint satisfaction is not a new one. However, most work in set-based estimation [5, 26, 27, 32] has focussed on linear systems of constraints. Likewise, recent work in object recognition using bounded error models [10, 11] has mostly focussed on sets of affine constraints. Taylor [29] has addressed the issue of error propagation though nonlinear kinematic equations using linearization methods. However, in most cases correctness cannot be guaranteed due to linearization errors. In addition, the error representation can become extremely expensive to manipulate as the number of constraints present in the system becomes large. The work of Brooks [7] also shares many similarities with the approach outlined in this paper, including the heavy use of intervals to manipulate sets of parameters. However, Brooks focuses on *symbolic* computation of constraint satisfaction on a restricted class of relatively low-dimensional geometric constraints. The representation method and techniques used are limited in their capabilities, are computationally expensive, and cannot be proved complete.

The contribution of this article is a technique the permits the correct and complete manipulation of large systems of nonlinear constraints, and the use of decision-directed computation to further increase the effectiveness of the algorithms. This paper emphasizes the practical and experimental aspects of these algorithms while a companion paper [16] contains more theoretical results including proofs of convergence. The remainder of this article is organized as follows: Section 2 introduces the concepts needed for developing systems of constraints that describe the relationship between sensor observations and parametric models, and the relationship between models and decisions. Section 3 describes interval-arithmetic methods for incrementally solving a system of constraints and reaching model-based decisions. Section 4 presents a set of experiments that were performed on real and synthetic range data. Section 5 describes extensions to the basic algorithm, including performance issues and parallelization. The final section discusses some of the limitations of the methods used, and points out interesting areas of future research.

2 Developing Systems of Constraints

The key to reliable model-based machine perception is the choice of the proper set of constraints to be used in interpreting sensor data [8, 14]. In this article, a decision-making problem is represented by two classes of constraints: data constraints and decision constraints. Data constraints describe the relationship between observations and a physical or geometrical model. They are typically

independent of any particular task. Conversely, decision constraints describe the conditions required to reach a decision or take an action. Decision constraints are independent of the characteristics of sensor data, but require an analysis of the physical and geometric constraints involved in the task to be performed.

The explicit incorporation of decision constraints means that a data model must not only be geometrically or physically plausible, but also reasonable for the types of decision criteria that must be expressed. Specifically, there is an important distinction between decisions that can be made from local information (a subset of available data) and those requiring global information (an analysis of all available data). For example, the stability of a three-point grasp can be determined from the local character of an object surface: in principle, a structured, global model is not needed. Conversely, deciding whether an object is small enough to be grasped is readily accomplished based on a global representation with explicit size parameters. This article, focuses on the use of parametric, global models for sensor-based decision making.

2.1 Data Constraints

For methodological clarity, data constraints are divided into three broad categories: constraints describing the geometric model, constraints describing the geometric characteristics of sensor imaging, and constraints that describe any additional physical or geometric characteristics of models or data. In what follows the degrees of freedom in the model, referred to as *model parameters*, are represented by the vector p which comes from a specified set $\mathcal{P}$. Sensor observations are represented by the vector u. The three categories of constraints outlined above take the following general form:

Category 1: For each type of sensor data available, a description of the relationship between observable quantities, x, and sensor observations u:

$$u = H(x, d, v), \quad u \in \mathcal{U} \subseteq \Re^m, \quad d \in \mathcal{D}, \quad v \in \mathcal{V} \subseteq \Re^n. \tag{1}$$

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 representing non-deterministic disturbances or other accuracy limitations of the sensor output.

Category 2: An implicit function relating each observed quantity, x, to model parameters, p:

$$g(p, x) = 0, \quad p \in \mathcal{P} \subseteq \Re^s, \quad x \in \mathcal{X}. \tag{2}$$

All parameters that are involved in decision-making must appear in one of these forms.

Category 3: Additional constraints expressing other geometric or physical properties of the sensor-environment-object relationship:

$$C(p, x, d) \leq 0. \tag{3}$$

It is important to note that constraints in categories 1 and 2 are exactly those employed in most current optimization or estimation-based sensor data fusion or reconstruction work [8, 9, 14]. Constraints in category 3 are somewhat unusual to see as they often significantly increase the complexity of optimization-based approaches. Geometrically, these forms define a surface in space, possibly subject to some other inequality constraints. Consequently, these constraints are most compatible with an environment where an acceptable structural form for observed objects is known *a priori*. The extensions needed to apply this framework to less structured environments are currently under development [15].

2.2 Examples

The following examples will be used to illustrate the above concepts, and also appear later in the experimental section (Section 4).

2.2.1 Single Objects

Superellipsoids were introduced by Solina [28] as a global modeling primitive for range data. A point in 3D space, $u = [u_x, u_y, u_z]$ lies on the surface of a superellipsoid in standard orientation if the following constraint is satisfied:

$$g(s, \gamma, l; u) = \left[\left| \frac{u_x - l_x}{s_x} \right|^{\frac{2}{\gamma_2}} + \left| \frac{u_y - l_y}{s_y} \right|^{\frac{2}{\gamma_2}} \right]^{\frac{\gamma_2}{\gamma_1}} + \left| \frac{u_z - l_z}{s_z} \right|^{\frac{2}{\gamma_1}} - 1 = 0 \tag{4}$$

This is a parametric model falling into category 2. The vector $s = [s_x, s_y, s_z]$ can be interpreted as the size of the superellipsoid, the vector $[\gamma] = [\gamma_1, \gamma_2]$ governs the shape of the superellipsoid, and $l = [l_x, l_y, l_z]$ is a location in space. Figure 1 shows a sampling of the forms possible by varying the shape (γ_1 and γ_2) parameters.

For a fixed set of model parameters, the expression above defines a set of points forming a closed surface in space. Conversely, given a set of points, a parameter vector is consistent with those points if (4) holds for each point up to sensing error. Similar constraints can be derived to relate surface normals to superellipsoid model parameters (see Appendix A). Expression (4) and the normal constraint comprise the data constraints conforming to (2).

Superquadrics will be recovered from surface points observed using a range scanner. A range observation is a quantized, noise-contaminated version of the true range to a surface. Let $D(x, y)$ denote the function that returns the perpendicular distance from the scanner to a surface at the point (x, y) in the imaging plane, and let s denote a scaling factor. A nominal description of the sensor (category 1) is given by the expression:

$$z = s\,(D(x, y) + v), \quad v \in [-t, t]. \tag{5}$$

In other words, the observed value of depth is a scaled version of the true depth at image

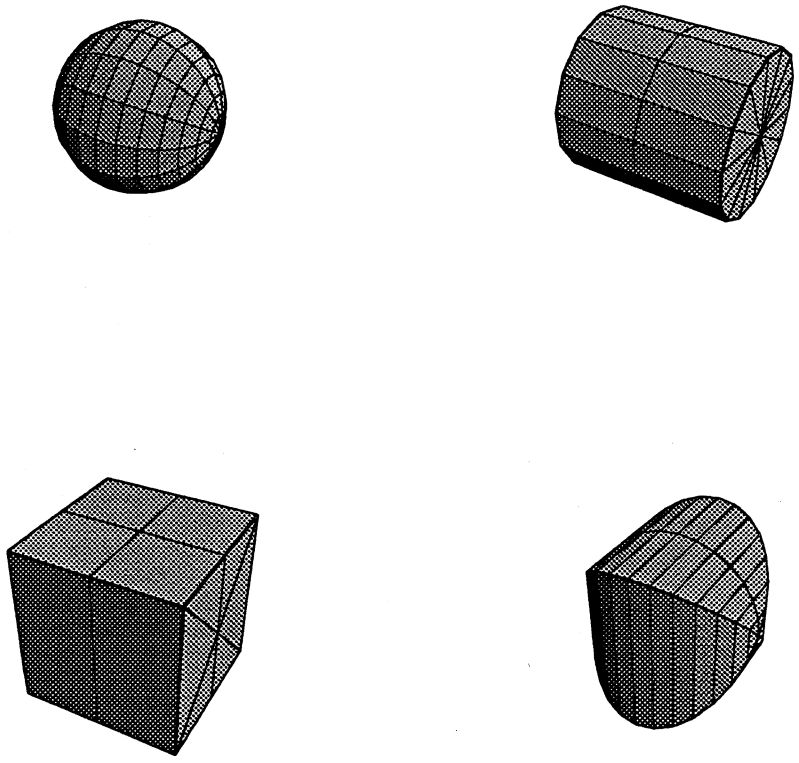


Figure 1. Some paradigmatic superellipsoids. Upper left, a sphere ($\gamma_1 = \gamma_2 = 1.0$); upper right a cylinder ($\gamma_1 = 0.1$, $\gamma_2 = 1.0$); lower left, a box ($\gamma_1 = \gamma_2 = 0.1$); and lower right, an unusual shape ($\gamma_1 = 1.0$, $\gamma_2 = 0.1$.).

location (x, y) that may be up to $s\,t$ units in error. Note that this expression *does not* describe the effects of occlusion or "drop outs."

In this case, it is possible to solve (5) for $D(x, y)$:

$$D(x, y) = z/s - v$$

and to substitute the result into (4). Hence, a given parameter vector, p, is consistent with an observation, $u = (x, y, z/s - v)^T$ if p and u jointly satisfy (4) for some $v \in [-t, t]$.

Finally, there are other constraints falling into category 3. Range images of objects are taken on a platform at a constant distance, B, from the scanner. By sorting points into platform points and non-platform points, B can be estimated from the data (using yet another constraint of type (2) above) and superquadric parameters can forced to obey the constraint:

$$l_z = s_z + B. \tag{6}$$

As noted above, the nominal sensor description does not include the effects of occlusion. However, occlusion properties can form strong constraint. For example, scanning an object with a flat top and vertical sides yields a range image comprised of two parallel planes. Since (4) neglects data from the scanning platform, it is severely underconstrained in this case: there are an infinite variety of models with flat tops that conform to the data. Solina [28] deals with this problem by adding a term to the optimization that prefers the smallest model that accounts for the data. A more physically plausible constraint is based on the observation that range points from the scanning platform are not occluded by the object and must therefore lie outside its projection onto the scanning platform. This constraint can be expressed (for a superquadric in standard position) as:

$$g(s, \gamma, l; u) = \left|\frac{u_x - l_x}{s_x}\right|^{\frac{2}{\gamma_2}} + \left|\frac{u_y - l_y}{s_y}\right|^{\frac{2}{\gamma_2}} - 1 \geq 0 \tag{7}$$

where u must come from the scanning platform. This constraint forces the object to be small enough to fit into its occluding contour.

2.2.2 Multiple Components and Multiple Objects

Objects with multiple components are described as the union of the component surfaces or, equivalently, by the disjunction of the constraints describing each component. This may be accomplished by multiplying the homogeneous forms of the constraints: the multiplied constraints are satisfied (equal zero) only when at least one of the constraints is satisfied. This approach could be used in the previous example to combine the superquadric model with the model of the imaging platform.

The properties of multi-component objects will be illustrated using gear flanges observed by a camera. The contours that would be extracted from such flanges (see Figure 2) are the outer toothed surface, and the inner hub. These contours in nominal position can first be defined as two explicit functions of angle:

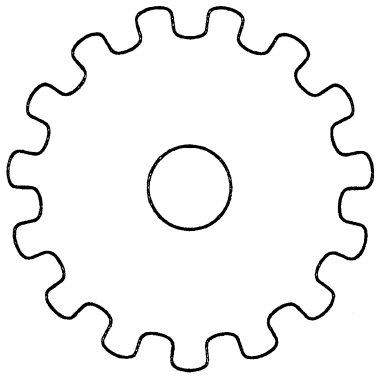


Figure 2. Picture of a gear

$$u_1(\theta) = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} (s + c\sin(K\theta + \theta_0))$$

$$u_2(\theta) = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} h$$

With some manipulation, it is possible to derive an implicit form of the constraint that does not require any extra variables. Adding location parameters and taking the union of the two curves yields the following constraint describing the contours:

$$\left[(x-l_x)^2 + (y-l_y)^2 - (s + c\sin(\theta_0 + K\,\mathrm{atan}(\frac{y-l_y}{x-l_x})))^2\right]\left[(x-l_x)^2 + (y-l_y)^2 - h^2\right] = 0. \quad (8)$$

Here, x and y are points on a contour and the model parameters are location (l_x, l_y), size s, gear cut c, hub size h, orientation θ_0, and number of teeth K. Note that the expressions for the individual contours share the same set of location parameters and that parameter K is restricted to integer values.

Multiple objects in a scene are an extreme case of multi-component objects. The only difference is that the components do not share any parameters or constraints. Consequently, by disjoining several data models, it is possible to automatically classify or *segment* data during the modeling process assuming the correct number and type of models is known beforehand. This type of multiple object example will be demonstrated in Section 4.

2.3 Decision Constraints

As noted in the introduction, decision constraints can be thought of as defining a proposition which, when satisfied, indicates that the set of model parameters support the associated decision. In most cases, there are several distinct decisions, each with a corresponding decision criteria. For example, classifying a superellipsoid model based on its size or shape parameters leads to distinct categories, each characterized by some range of sizes and shapes. The goal is to determine if a model of the observed object falls uniquely into any one of these categories. Let the decision criteria for a decision d be represented by a proposition a_d on $\mathcal{P}$, and let $\mathcal{A}$ represent a collection of decision predicates. The disjunction of the members of $\mathcal{A}$ leads to a logical condition which, when satisfied, indicates that one or more decisions can be made. These ideas are illustrated in the following examples:

Example 2.1 Suppose that the task is to determine whether an object (modeled as a superellipsoid) can be grasped by a gripper of known size, w. Then, assuming that the axis of the gripper is aligned with the axis of the superellipsoid, a "graspable" model must satisfy one of the following three conditions:

$$\begin{aligned} gr_x(s_x) &:= s_x - w < 0 \\ gr_y(s_y) &:= s_y - w < 0 \\ gr_z(s_z) &:= s_z - w < 0 \end{aligned} \tag{9}$$

In this example, $\mathcal{A} = \{gr_x, gr_y, gr_z\}$.

Example 2.2 Suppose that the goal is to categorize objects based on shape. Then, define predicates

$$\begin{aligned} \texttt{rnd}(\gamma) &:= \gamma \geq 0.7, \\ \texttt{sqr}(\gamma) &:= \gamma \leq 0.3, \\ \texttt{thn}(s) &:= s \leq 10. \end{aligned}$$

Using these predicates, four conditions describing composite classifications of superellipsoids in standard position can be defined as:

$$\begin{aligned} \texttt{flat}(p) &:= \texttt{thn}(s_z), \\ \texttt{tube}(p) &:= \neg\texttt{flat}(p) \wedge (\texttt{rnd}(\gamma_2) \wedge \texttt{sqr}(\gamma_1)), \\ \texttt{carton}(p) &:= \neg\texttt{flat}(p) \wedge (\texttt{sqr}(\gamma_1) \wedge \texttt{sqr}(\gamma_2)), \\ \texttt{parcel}(p) &:= \neg\texttt{flat}(p) \wedge \neg\texttt{tube}(p) \wedge \neg\texttt{carton}(p). \end{aligned}$$

In this example, $\mathcal{A} = \{\texttt{flat}, \texttt{carton}, \texttt{tube}, \texttt{parcel}\}$.

Example 2.3 Suppose that an assembly operation requires analyzing an assortment of gears to find the gear with the largest hub diameter. Let h_i represent the hub diameter parameter for gear i of a set of n gears. The decision predicate for gear i is a conjunction of the form:

$$a_i(h_1, \ldots, h_n) := \bigwedge_{j=1, j\neq i}^{n} h_i \geq h_j.$$

That is each gear model is tested to see if it is larger than all other gears. There are n of these predicates, one for each gear so $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$.

It is important to note that, while decision constraints themselves may include disjunctions, there is a difference between a disjunction *within* a decision constraint, and a disjunction *of* decision constraints. For example, the graspability condition of Example 2.1 combined into a single constraint is:

$$a(s) := \bigvee_{i\in\{x,y,z\}} (w - s_i) > 0 \tag{10}$$

This statement is logically equivalent to

$$\forall s_x\, s_y\, s_z\ (a(s_x) \vee a(s_y) \vee a(s_z)).$$

whereas the statement of Example 2.1 is equivalent to

$$\forall s_x\, a(s_x) \vee \forall s_y\, a(s_y) \vee \forall s_z\, a(s_z)$$

The latter requires that some single axis is determined to be graspable. The former is true if all parameter values describe a graspable object, but no single specific axis is proved graspable. In other words, the former statement is implied by the latter.

2.4 A Canonical Form for the Decision-Making Problem

The systems of constraints described above are essentially logical statements. These statements can be rewritten in conjunctive normal form (a conjunct of disjuncts). As noted earlier, the disjunction of two equalities in homogeneous form can be expressed algebraically by multiplying the expressions. Likewise, a disjunction that contains inequalities can also be combined into a single expression, although the algebraic form is somewhat more complicated [25]. All equalities can be written as two inequalities of the form $g(\cdot) \leq 0$. By this means, any system of constraints can be reduced to a conjunction of parametric inequalities. Letting the vector U represent the "giant" vector of all observed data, the final conjunction of constraints can be expressed in the form $g^*(p, U) \leq 0$ where $\leq$ is interpreted to apply componentwise. Finally, recall that each element of U has an associated error bound. Let $\mathcal{U}$ be the (bounded) set of vectors containing all values of U when the components are allowed to vary within their respective tolerance intervals.

A parameter vector p is *consistent* with a set of constraints g^* and data vector U if and only if $g^*(p, U) \leq 0$. If this condition does not hold, the parameter vector is *inconsistent* with U and g^*. The set of all consistent parameter vectors for a given set of constraints and data,

$$\mathcal{P}^*(g^*, \mathcal{U}) = \{p \in \mathcal{P} \mid g^*(p, U) \leq 0 \text{ for some } U \in \mathcal{U}\},$$

Other comparisons are completely analogous. Interval equality is defined as

$$\mathbf{x} = \mathbf{u} \text{ iff } \underline{\mathbf{x}} = \underline{\mathbf{u}} \wedge \overline{\mathbf{x}} = \overline{\mathbf{u}}.$$

The *interval extension* of a (vector) function $H : \Re^s \rightarrow \Re^m$ with component functions $h_i : \Re^s \rightarrow \Re$, $i = 1, \ldots, m$ is defined as

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

Given interval versions of the basic algebraic and trigonometric operators, the interval extension of a function can be computed by replacing all point operators and functions with their corresponding interval equivalents. With the addition of interval comparison operators, general nonlinear constraints can also be implemented by replacing all point operators and comparisons by their interval equivalents.

3.2 Interval Bisection

As noted previously, observation errors are assumed to be bounded. Consequently, a data vector u can be represented as an interval $\mathbf{u} = [u-t, u+t]$ where t is a known error bound. Thus, observations can be viewed as intervals. Given an interval of parameters, interval arithmetic operations can be used to prune away inconsistent parameters. By way of illustration, suppose that the only constraints are of the form $g(x, y, r; u, v) = (x-u)^2 + (y-v)^2 - r^2 = 0$ where (x, y, r) are the model parameters (location and size) while (u, v) is a generic data point. Given a parameter interval $\mathbf{p} = (\mathbf{x}, \mathbf{y}, \mathbf{r}) = ([-1, 1], [-1, 1], [5, 6])$ and data intervals $\mathbf{z}_1 = ([5, 6], [-0.5, 0.5])$, $\mathbf{z}_2 = ([5, 6], [6, 7])$, and $\mathbf{z}_3 = ([7, 8], [-0.5, 0.5])$, the following values of g can be calculated using interval arithmetic:

$$\mathbf{g}(\mathbf{p}, \mathbf{z}_1) = [-20, 26.25], \quad \mathbf{g}(\mathbf{p}, \mathbf{z}_2) = [5, 88], \quad \mathbf{g}(\mathbf{p}, \mathbf{z}_3) = [0, 58.25].$$

The point z_2 yields only positive values is therefore inconsistent with the parameter interval. That is, there are no parameters in $\mathbf{p}$ describing a circle that passes within the error envelope of z_2. The point z_3 is nearly inconsistent. It can only be made consistent by pushing the parameter and data error values to their extremes. The point z_1 is far from being inconsistent. If z_1, z_2, and z_3 all come from the same data set, then the parameter vector $\mathbf{p}$ would be declared inconsistent with this set of data by virtue of z_2. It is worth noting that although the size of the parameter and data intervals was constant, the size of the constraint intervals varies by nearly a factor of 2. In general, the propagation of uncertain quantities through nonlinear functions leads to wide, parameter dependent variations in the uncertainty of function values.

More generally, let $\mathbf{U}$ represent the conjunction of all data interval vectors. Suppose that the interval extension of g^*, $\mathbf{g}^*$, is evaluated on a parameter interval vector $\mathbf{n}$ and data interval vector $\mathbf{U}$. If $\mathbf{g}^*(\mathbf{n}, \mathbf{U}) \leq 0$, then every point in $\mathbf{n}$ is consistent with U. Conversely, if, for some component k of $\mathbf{g}^*$, $\mathbf{g}^*_k(\mathbf{n}, \mathbf{U}) > 0$, then every point in $\mathbf{n}$ is inconsistent with U . If neither case holds, then no decision can be made. Using these ideas, an interval *reduce*() operator can be defined as:

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

1. For each component i, $i = 1, \ldots, s$, trisect[2] $\mathbf{n}$ in i, yielding intervals $\mathbf{n}_{1,1}, \mathbf{n}_{1,2}, \ldots, \mathbf{n}_{s,2}, \mathbf{n}_{s,3}$.

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

2. For each $\mathbf{n}_{i,j}$, if $\mathbf{g}_k^*(\mathbf{n}_{i,j}, \mathbf{U}) > 0$ for some component k, then $\mathbf{n}_{i,j} := \emptyset$.

3. $\mathbf{n} := \bigcap_{1 \le i \le s} \bigcup_{1 \le j \le 3} \mathbf{n}_{i,j}$

Intuitively, *reduce*() tests each component of an interval parameter vector to see if an inconsistent subinterval can be "sliced off." This procedure is linearly convergent. It is also possible to define interval operations using derivatives that have asymptotically quadratic convergence. The use of these and other modified versions of *reduce*() is discussed in [16].

In what follows, a function *bisect*$(\mathbf{n}, d)$ computes the set of subintervals resulting from bisecting component d of $\mathbf{n}$. $\mathcal{A} = \{a_1, \ldots, a_n\}$ represents a set of decision predicates, and `test` represents a boolean procedure to be defined later. $\mathcal{Q}$ and $\mathcal{L}$ represent sets of interval vectors.

An algorithm to compute a decision is:

Algorithm 3.1

generalized-bisection$(\mathbf{n}, \mathbf{g}^*, \mathbf{U}, \mathcal{A})$:

1. *(Initialization)*
   (a) $\mathcal{Q} := \{\mathbf{n}\}$.
   (b) $\mathcal{L} := \emptyset$.
2. *Termination Test*
   (a) If $\mathcal{Q} = \mathcal{L} = \emptyset$, stop with **inconsistency.**
   (b) If `test`$(\mathcal{L}, \mathcal{Q}, \mathcal{A})$ stop with **success.**
   (c) If $\mathcal{Q} = \emptyset$, $\mathcal{L} \neq \emptyset$, and `test` fails, stop with **no decision.**
3. *(Reduction)*
   (a) Remove an interval $\mathbf{x}$ from $\mathcal{Q}$.
   (b) Compute *reduce*$(\mathbf{x}, \mathbf{g}^*, \mathbf{U})$.
   (c) If $\mathbf{x} = \emptyset$, go to step 2.
   (d) If $\mathbf{g}^*(\mathbf{x}, \mathbf{U}) \le 0$, then $\mathcal{L} := \mathcal{L} \cup \{\mathbf{x}\}$. Go to step 2.
4. *(Bisection)*
   (a) Choose a dimension $1 \le d \le s$ with $\mathrm{w}(\mathbf{x}_d) \ge 0$.
   (b) $\mathcal{Q} := \mathcal{Q} \cup$ *bisect*$(\mathbf{x}, d)$.
   (c) Go to step 3.

The acceptance test is defined as:

$$\texttt{test}(\mathcal{Q}, \mathcal{L}, \mathcal{A}) := \mathcal{L} \neq \emptyset \wedge \left(\bigvee_{a \in \mathcal{A}} \bigwedge_{\mathbf{n} \in \mathcal{P}} a(\mathbf{n}) \right)$$

In other words, `test` is true if *some* partition element has been proved consistent with observed data, and *all* active or accepted partition elements satisfy the acceptance criteria of some decision.

The former condition is needed to ensure that the criteria are not vacuously satisfied by the empty set. This condition can be dropped if the models used are expected to be physically correct. The latter condition ensures that the criteria for the decision hold for all points that may prove consistent with the data. If all model constraints are satisfied, but no single decision criteria is satisfied, the procedure indicates that no decision can be reached. If all parameter intervals are exhausted, then the algorithm indicates that the model constraints are inconsistent.

This algorithm illustrates the interplay between between data and decision constraints. Intuitively, the algorithm can be thought of as incrementally "classifying" portions of the parameter space as inside or outside the solution set. At every iteration, the set of intervals $\mathcal{Q} \cup \mathcal{L}$ is a partitioning of the current best approximation to the solution set. The algorithm terminates with success if the current set of consistent and undecided intervals satisfies the conditions of some decision. Consequently, the algorithm will terminate most quickly when the decision constraints are very weak (the supporting set is large) and the model constraints are very strong (the solution set is small).

3.3 Making the Algorithm Deterministic

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

Bisection Choices Ideally, bisection should enhance the ability of *reduce*() to perform constraint solving. To illustrate, suppose that $\mathbf{g}(\mathbf{x}) = \mathbf{x}_1 + 2 * \mathbf{x}_2$ with $\mathbf{x}_1 = \mathbf{x}_2 = [-1, 1]$. Bisecting $\mathbf{x}_1$ leads to values for $\mathbf{g}$ of $[-3, 2]$ and $[-2, 3]$ whereas bisecting $\mathbf{x}_2$ yields $[-3, 1]$ and $[-1, 3]$. The latter bisection has less overlap between the computed intervals and hence does a better job of distinguishing between the two interval vectors. Thus, a good quantity to base bisection on is the change in the widths of the values of $\mathbf{g}^*(\cdot)$ relative to changes in the widths of the elements of an interval parameter vector. This is estimated by computing a sensitivity value as follows: for an interval vector $\mathbf{n}$, let $\mathbf{n}_i^c$ be $\mathbf{n}$ with component i replaced by the center of the interval $\mathbf{n}_i$. Then

$$s_{i,k} = 1 - \frac{w(\mathbf{g}_k^*(\mathbf{n}_i^c, \mathbf{U}))}{w(\mathbf{g}_k^*(\mathbf{n}, \mathbf{U}))}.$$

Each $s_{i,k}$ is a value between 0 and 1 with values near 1 indicating that the the ith parameter has a large effect on the kth constraint.

Bisection is performed in the parameter component i that has maximum sensitivity value over all data values. One caveat, however: the constraints used to compute sensitivity values must depend on more than one model parameter. If a constraint involves only one parameter, then the constraint will always receive a sensitivity value of 1 for that parameter and bisection will never be performed on any other parameters.

Interval Selection The criteria describing a decision partition the parameter space into two sets: the set of parameters satisfying the stated criteria (hereafter referred to as the *supporting set* for the decision), and those that do not. Using interval-based constraint testing, any interval can be classified as supporting a decision, not supporting it, or neither. Define the *work* required to reach a decision to be the number of nonsupporting (undecided and refuting) intervals that are present on the active queue. The algorithm used to choose the interval to work on is as follows:

1. Choose the decision that has the least remaining work.
2. If there are intervals that are undecided with respect to this decision, choose the largest (by volume measure) to work on; otherwise
3. Choose the largest refuting interval to work on.

Recall that the supporting sets for decisions cover the entire parameter space. Thus, the above algorithm will always find *some* undecided or positive decision to work on.

Under reasonable assumptions this method leads to a convergent algorithm. In most cases, it leads to efficient generation of decisions since the work values for the most strongly supported decision tend to remain small compared to the work values for all other decisions. Once focussed on that decision, the algorithm concentrates effort into reaching it. Obviously, there are also scenarios that lead this method to perform badly. An interesting open question is to find search ordering algorithms that perform in an optimal or near-optimal fashion.

3.4 Further Considerations

Correctness and Convergence Assuming operators are properly defined and satisfy (11), it can be shown that Algorithm 3.1 terminates with **success** only if $\mathcal{P}^*(\mathbf{g}^*, U) \subseteq \mathcal{T}(a)$ for some $a \in \mathcal{A}$. Hence, the algorithm is *correct.* If, in addition, the constraint functions obey a Lipschitz condition, variables representing data only appear once in interval expressions, and certain fairness properties hold when choosing nodes from the active queue and choosing components to bisect, it can be shown that the interval bisection procedure converges to the solution set of a system of constraints. Consequently, the bisection procedure is also *complete*: under the stated assumptions Algorithm 3.1 will eventually terminate with **success** if $\mathcal{P}^*(\mathbf{g}^*, U) \subseteq \mathcal{T}(a)$ for some $a \in \mathcal{A}$. The proofs of these results can be found in [16].

Data Outliers Within the paradigm of this paper, a data outlier is an observation that exceeds the error bound used to formulate the system of constraints. Such observations can lead to incorrect decisions and potentially render the system of constraints inconsistent. There is a straightforward approach to handling outliers in data. If it is suspected that c percent of the data is outlier data, then no interval is rejected until it is inconsistent with $c'n$ constraints, where n is the total number of constraints in the system and $c' > c$. For large data sets this has proved to be a very workable approach to robustness, and can also be proved statistically convergent under reasonable assumptions.

Most applications tested to date employ a two-stage elimination of outlier data. The first, low-level stage checks for local consistency of data. The goal of this stage is to eliminate practically all of the outliers, even if some non-outlier data is also discarded. The second stage is the robust *reduce*() algorithm described above applied with a low value of c'. A concrete example of this approach is described in Section 4.

Data Selection The *reduce*() procedure often requires only a small portion of the data vector to work efficiently. A data selection procedure has been implemented and tested and has been found to work quite well. The processing required for data selection is significantly less that that required for *reduce*(), so the use of data selection usually yields computational performance improvements of 5 to 10 fold with little change in the number of iterations required to reach a decision.

Disjunctions As mentioned in the previous section, disjunctions (and also conjunctions) of constraints can be represented analytically. This form has several advantages: straightforward mathematical treatment, and simple implementation. However, explicit representation and computation of disjunctive statements often avoids the need to evaluate all disjuncts to test the joint constraint and also makes it possible to determine which data items satisfy which constraints (and hence which component of a multi-component model the data items belong to). Furthermore, convergence is often slowed due to the mixing of variables when using the analytic form. Consequently, explicit computation of disjunctive and conjunctive expressions is, from a computational perspective, usually the best approach.

Integer Values Integer-valued parameters are supported in essentially the same manner as real-valued parameters. The only difference occurs when an interval is split (either bisection or trisection). In this case, the lower-values of each interval are rounded up to the next integer value, and the upper-values of each interval are rounded down to next lowest integer value.

4 Experimental Results

The interval bisection algorithm is relatively simple to implement in a language that allows operator overloading. The version of the algorithm used to obtain the results in this section was written in C++ based on an in-house developed interval arithmetic package. Using this system, the implementing and solving systems of constraints is does not involve much more than writing procedures that compute the individual constraints. More details on this system can be found in [16].

4.1 Data Acquisition

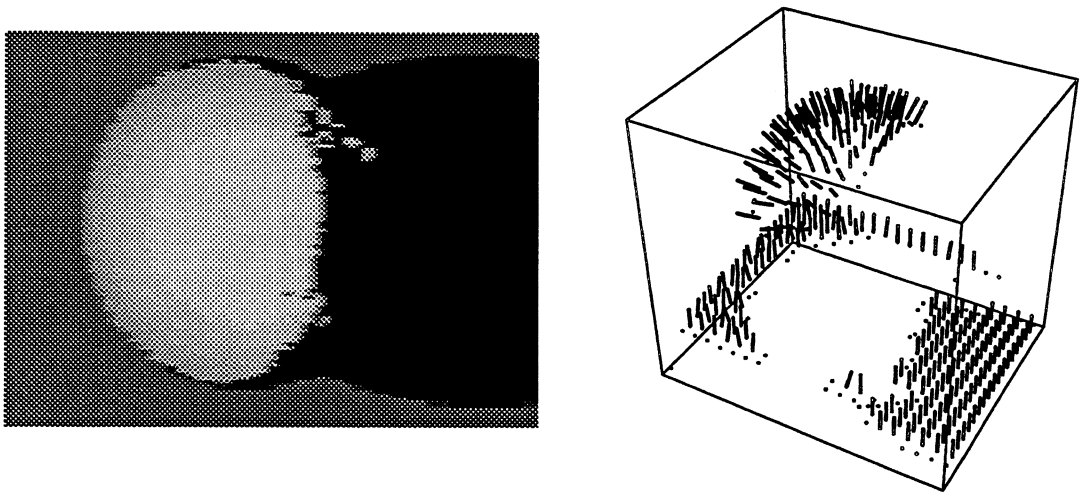


Figure 3. Data taken from a softball: raw data (encoded as an intensity image) and subsampled data with computed normals.

Range data is acquired by shining a laser on a surface and computing the distance to the surface via triangulation with a fixed camera [31]. The system has a minimum resolution of 1.5 mm[3] in the z (vertical) direction, and modeling of the sensor has shown that it typically returns values correct to one quantization level. It occasionally makes an error of two levels, but has never exceeded this bound except at object boundaries. Simulated range images are constructed by computing an artificial range image of a synthetic object and adding noise comparable to the real scanner. These images provide a basis for comparison to real data, and are used for debugging. Both real and simulated images are treated identically in all later stages of processing. All results described in this section used a uniform sample of 400 range points arrayed in a 20 by 20 grid.

Although the bisection algorithm can be made robust to outliers, its performance is best when the number of outliers is quite small. As noted above, the range data used here tends to have outliers at transitions between surfaces. However, objects are expected to be locally smooth, so a small neighborhood about a point should fit a quadratic curve up to sensing error. For a single row of data, a quadratic fit about a range point $u_i = (x_i, y_i, z_i)$ can be computed using the following set of constraints for a value of $k \neq 0$:

$$\begin{aligned} d_k &= x_{k+i} - x_i \\ b_k &= \frac{z_{i+k} - z_{i-k}}{d_k - d_{(-k)}} \\ a_k &= \frac{z_{i+k} - z_i - b_k d_k}{d_k^2} \\ c_k &= z_{k+i} - a_k d_k^2 - b_k d_k \end{aligned}$$

Substituting $\mathbf{z}_i = [z_i - 1.5, z_i + 1.5]$ for z_i and using interval arithmetic to evaluate the above

[3]Unless otherwise noted, all positions and sizes in this section are in millimeters.

expressions leads to interval values of a, b, and c for each value of k. The evaluation is repeated for $k = -5$ to $k = 5$ (skipping $k = 0$) and the results are combined via intersection. The process is performed for the corresponding column (substituting y for x in the above equations). If all intersections are nonempty, the point is deemed consistent and the values of b for the two orthogonal directions are used to compute the normal vector at the point. Heretofore, the value of k has been chosen empirically by examining the results of data rejection in simulation and on representative data sets. On the average, 0% to 5% of the data is rejected by this test. The remaining data values are assumed to be in error by no more than 1.5 mm.

Figure 3 shows the raw range image (with range encoded as grey value) and sampled range points with normal vectors for a softball. Distance from the imaging device is encoded as brightness: lighter areas are closer and darker areas are further away. The completely dark areas in the images indicate *shadows*—areas where the camera cannot see the light cast by the laser. No data is available from those areas.

4.2 Constraints

During fitting the algorithm dynamically classifies range observations as points on the platform, points on the object, or points falling in shadow. There are three data-dependent constraints applied to data points from the object (two of which were presented in Section 2): the inside-outside function applied to observed surface points, a surface normal constraint applied to normals computed from the raw range data, and a projection constraint applied to platform points. There is also a constraint used to recover the height of the scanning platform from the range data. As was discussed in Section 2, the position and size of the object is coupled to the height of the platform. The full parametric form of these constraints is given in Appendix A.

Each iteration of the algorithm takes from 1 to 20 seconds on a Sun SparcStation 2[4]. Rough calculations suggest that more efficient coding would reduce this time by a factor of 5 to 10. Adding parallelism could further reduce computation time to the point that most examples in this section could be run in a matter of seconds (see Section 5 for more discussion).

4.3 Experiments

Experiments are carried out in simulation and on real objects. The major difference between simulated data and experimental data is the lack of range shadows in the former. Consequently, the simulations usually provide better results, and as such provide an upper bound on the performance of the algorithm. That is, if the algorithm fails in simulation, there is no reason to expect it to work on real data.

In the following, all values are in units of millimeters. Accuracies are reported as the half width of the interval on the measured or computed parameter. The parameters of the test objects used are as follows:

[4]Sun is a trademark of Sun Inc.

Source	s_1	s_2	s_3	γ_1	γ_2
soda can	35	35	61	0.1	1.0
cd case	12	70	62	0.1	0.1
softball	50	50	50	1.0	1.0
lacrosse ball	33	33	33	1.0	1.0
catfood can	44	44	20	0.1	1.0

Absolute Accuracy The following table shows the accuracy of the superellipsoid size and shape parameters after 1500 iterations of fitting a model to simulated and real data:

Data Source		Size			Shape	
		s_1	s_2	s_3	γ_1	γ_2
softball	sim.	3.2	1.1	2.5	0.08	0.08
	real	9.8	8.3	13.2	0.30	0.46
lacrosse ball	sim.	4.9	2.0	1.8	0.02	0.1
	real	7.3	2.9	7.5	0.22	0.46
cd case	sim.	2.2	8.9	10.5	0.20	0.30
	real	9.9	11.9	15.6	0.27	0.46
soda can	sim.	2.2	1.2	5.6	0.13	0.20
	real	7.3	1.3	2.2	0.18	0.46
cat food can	sim.	0.5	1.1	2.0	0.1	0.06
	real	11.0	13.3	39.5	0.46	0.46

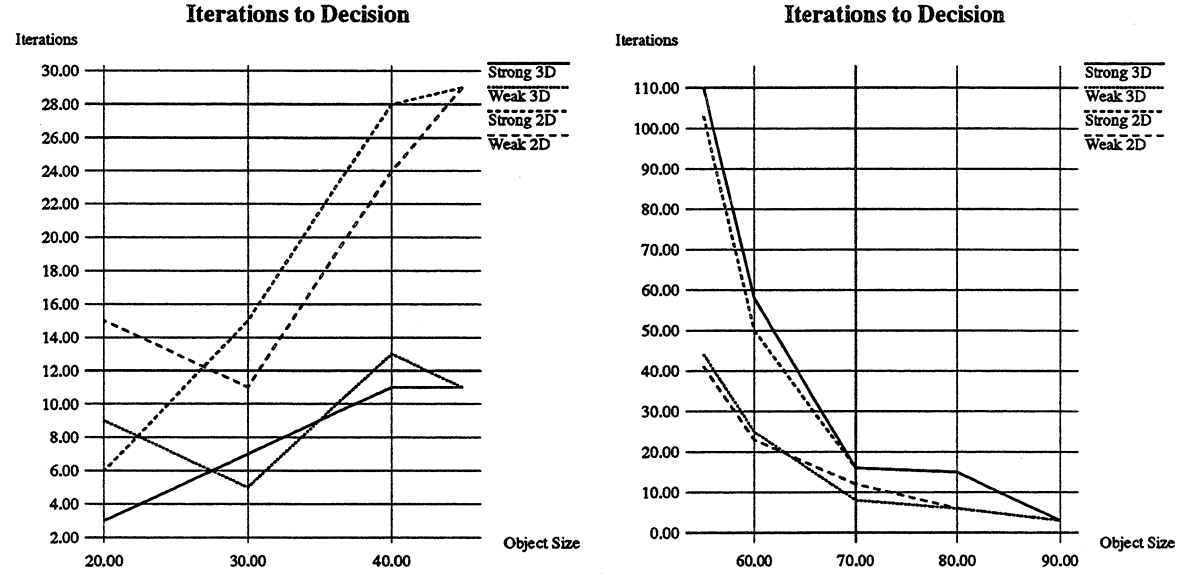
Although parameter bounds permit only rough comparison, it is clear that the fitting results on real data are not nearly as accurate as the results in simulation. As noted previously, the primary difference between real and simulated data are self-occlusions that eliminate major portions of the data. For this reason, real data is often inadequate to fully determine the model parameters, and the method accurately reflects this fact. Also, objects with a flat top are more difficult to recover since there is no data from the sides. The length and width of the object can only be inferred indirectly from the shadows that it projects onto the platform.

To illustrate the effect of range shadows, the same fitting process was applied to range scans taken from different directions. The following table shows the results of 1000 iterations of fitting a model to three range images of the softball and lacrosse ball spaced at angles of 0, 45, and 60 degrees:

Object	s_1	s_2	s_3	γ_1	γ_2
softball	5.3	2.3	3.5	0.13	0.13
lacrosse ball	2.2	4.4	2.2	0.14	0.2

The marked improvement in the accuracy of the recovered parameters is directly attributable to the additional model constraint achieved by using data from three views. In general, with three range images the results are as good as (sometimes better!) than in simulation.

Grasping Decisions The algorithm has been tested on the grasping problem described in Example 2.1. Both versions of the grasping criteria—the strong version of (9) and the weaker version given in (10)—were implemented. Furthermore, both of these versions were tested when considering the sizes of all three axis of the object, and also when the axis upon which the object rests is eliminated from consideration. The former will be referred to as the 3D decision, and the latter as the 2D decision. The gripper radius was set at 50mm. The following graphs show the number of iterations required to determine if simulated objects of various sizes would fit into a gripper:



The left graph shows the number of iterations to compute a positive decision, while the right graph shows the number of iterations required to refute a decision.

As theory predicts, the algorithm requires more computation as the size of the object approaches that of the gripper. Furthermore, theory would predict that 3D decisions are simpler to compute than 2D decisions, and that refuting 3D decisions is harder than refuting 2D decisions. This is generally true in the above data. Likewise, "weak" versions of the problem are easier to prove and harder to refute than the corresponding "strong" version; this is also apparent in the results.

The following results were obtained from real range images of the test objects (in this table "–" indicates no decision after 500 iterations):

	Soda Can	CD case	softball	lacrosse ball	catfood can
Strong 3D	Yes, 15	Yes, 10	–	Yes, 11	–
Strong 2D	Yes, 22	No, 22	–	Yes, 29	–
Weak 3D	Yes, 5	Yes, 7	Yes, 47	Yes, 3	Yes, 266
Weak 2D	Yes, 18	No, 22	–	Yes, 18	–

This illustrates what is probably the most important result of this paper: adding task-specific constraints to a problem can lead to efficient decision making even when the general recovery problem is underdetermined by the available data! Task-directed algorithms can be expected to be robust to occlusion or other factors leading to missing data. They will continue to operate

effectively as long as the available data suffices to reach a decision.

The algorithm had difficulty with the softball due to the fact that it is nearly the size of the gripper. The catfood can caused difficulties due to the large amount of missing data. These two cases illustrate the interplay between decision constraints and data constraints. That is, a very strong constraint on the size along the z axis comes from the constraint linking the height of the platform to the size and position of the object. When this dimension is included in the "weak" decision, there is just enough information in the range image to reach a decision for both the softball and the catfood can.

Classification Shape parameters are generally more difficult to determine than size parameters. Hence, determining a unique classification for objects can be quite difficult for the algorithm, particularly if data is missing. For comparison, classification tests were also carried out for the two balls using three range images. The following results were obtained (in this table, a "–" indicates undecided after 1000) iterations:

Object	Simulation	One View	Three Views
softball	47	682	233
lacrosse ball	45	–	327
cd case	296	897	n/a
soda can	81	222	n/a
catfood can	708	–	n/a

It is important important to note that in cases where the data does not support a unique classification, one solution is usually much more "probable" than any of the others. In particular, the lacrosse ball is not classified because the percentage of missing data is much larger than any other object (the softball is the nearest competitor). Although the object was not classified, the only remaining choices were "sphere" or "unclassifiable." The latter had very low probability.

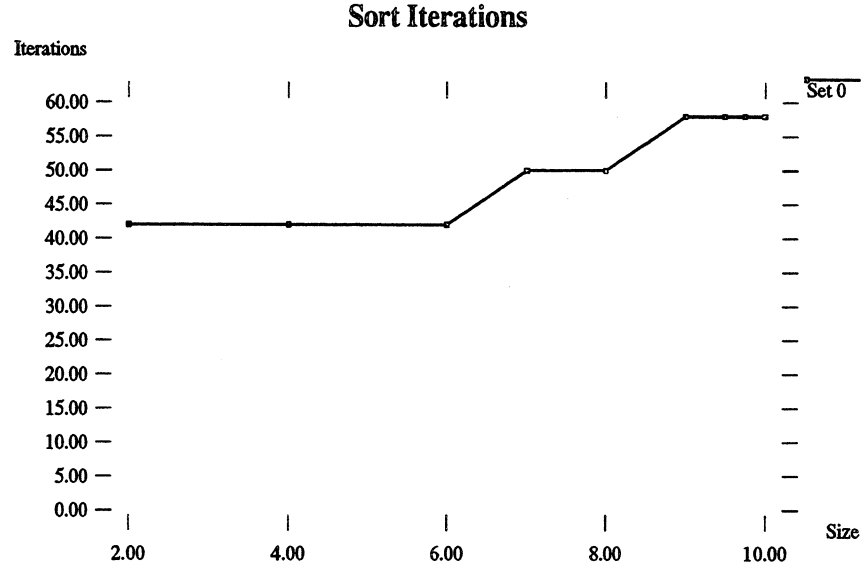
Sorting When comparisons among multiple objects are involved, the decision-making algorithm can be altered slightly to take advantage of the special structure of the problem. In all that follows, it is assumed that segmentation problem has been solved, and that the parameters of each observed object are independent of other objects.

Recall that the goal of Example 2.3 is find the gear with the largest hub diameter from a given collection. The decision criteria for gear i of a set of n gears is a conjunction of the form:

$$\bigwedge_{j=1, j\neq i}^{n} h_i \geq h_j.$$

That is each gear model is tested to see if it is larger than all other gears. Disproving that a gear is largest involves showing that *any* other gear is larger. Since the parameters of each gear are independent of other gears, the processing for each gear can proceed in parallel. Processing on a specific gear can stop when it is shown to be smaller than some other gear.

The bisection algorithm discussed previously is easily extended to support this type of decision making. Algorithm performance on multiple objects is illustrated using Example 2.3. These trials were carried out using simulated contour data of gears with an error bound of one pixel. The program was given presegmented data from 9 gears with the task of choosing that with the largest hub diameter. The program noted when processing was stopped on each gear. The following graph shows the number of bisection iterations devoted to each gear as a function of hub diameter:



As is expected, the closer the size of a gear to the largest gear (with hub size 10), the more iterations it takes to disambiguate it. The stepwise nature of the graph is due to quantization effects from the *reduce*() procedure.

4.4 Comments

In addition to the experiments described above the bisection algorithm has been run on a variety of other problems and data sets. In most situations the bisection algorithm performs extremely well. It appears to be quite robust to data outliers, and usually computes a decision (if one can be computed) with surprisingly little computational effort.

Over these tests, several aspects of this algorithm and the interval-based paradigm in general have become apparent. As illustrated above, when parameters are poorly determined by the data, model recovery convergence becomes very slow. This is exacerbated by the need to reject outliers. This is not completely unexpected—when parameters are poorly determined, the effect of data errors increases and more bisection must take place before the *reduce*() procedure becomes effective. However, decision-constraints often provide enough structure to offset convergence problems. There is only a moderate increase in the amount of computation needed to reach a decision, unless the parameters needed to reach a decision are exactly those that are difficult to determine from the data.

There is a clear advantage to building redundant systems of constraints, particularly when some

constraints apply to a subset of the entire parameter vector. In particular, algorithm performance can be improved greatly by performing grouping and clustering of low-level structures into medium level structures. For instance, grouping points into lines and curves permits more direct recovery of parameters such as orientation and radius. However, this grouping must be done carefully so that the property of bounded error can be preserved. For example, the most troublesome aspect of fitting superquadrics is filtering and computing range normals that are correct up to a specified error tolerance.

The use of trisection in the *reduce*() procedure leads to quantization problems. The algorithm will sometimes spend a large number of iterations "refining" a parameter until it is recognized as supporting or not supporting a decision, even when the constraints involved are quite simple. Thus, a small change in parameters decision parameters can cause a relatively large change in algorithm behavior. One approach to solving this problem is to use a second-order procedure, or some other procedure more intelligent than trisection. These issues are currently being investigated.

5 Extensions and Discussion

The basic paradigm described in the previous sections has a number of interesting extensions that permit it to perform more efficiently, address a larger variety of tasks, and weaken its reliance on strong parametric assumptions. This sections details some of these issues.

5.1 Using Comparisons Among Multiple Objects

In an assembly task, it is reasonable to assume that the sizes of the relevant parts are known. Hence a sensing task could could be expressed as "Find a gear with a 20mm hub" or "Find a 20mm diameter shaft." However, it may also be possible to describe a task using relative comparisons, *eg*. "Find a gear with the same size hub as some shaft." The former requires fitting to an absolute accuracy bound, while the latter requires simultaneous fitting of shafts and gears until two can be proved equal. Relative comparisons have the advantage that they do not require *calibration to an absolute coordinate system.* In general, problems that do not rely on an absolute scale or coordinate system will be more robust and simpler to build than systems that do.

5.2 Conditional Execution and Multiple Decisions

Imagine planning a grasp of an object based on three different pieces of information: the object is known to be graspable, the object is known to be graspable in a specific orientation, and the object is known to be graspable by servoing to a specific point in space. The first statement provides enough information to ensure that further reasoning will lead to successful grasp, but nothing more. The second confirms a particular strategy, and only requires the monitoring required to carry out the grasp. The final statement provides enough information to carry out the operation open-loop. Conversely the first expression would be the simplest to compute from sensor data, and the final expression would be the most complex.

If there are two decision predicate, a_1 and a_2 such that $\mathcal{T}(a_1) \subset \mathcal{T}(a_2)$, then whenever $\mathcal{A}^*$ includes a_1, it will by definition include a_2. As noted before, the decision criteria expressed in (9) and (10) are two such predicates. Consequently, by formulating a sequence of incrementally more precise decisions, it would be possible to condition actions on the detail of the information that can be computed from sensor data. If the data is poor, $\mathcal{D}^*$ will only contain very general decisions, whereas very good data may cause $\mathcal{D}^*$ to contain decisions based on very accurate or detailed information. The number and type of decisions that are satisfied, refuted, and undecided may give some indication of the quality of the observed data.

A planner or execution module could make use of this property to optimize task performance. That is, if it is relatively simple to compute a strong decision, then task execution can be speeded up, or performed using fewer resources. Conversely, if it is difficult to make even simple decisions, then it may be more efficient to perform the task with less information or more resources than to compute more information from the given data. Thus, the execution of a task can be conditioned on the ease and availability of sensor information.

5.3 Weakening Reliance on Global Model Structure

There are two means for reducing reliance on specific, global model structures. One approach is to retain the notion of a global model, but define the model with inequalities. This describes an enclosed or enclosing volume (depending on the sense of the inequality). Consequently, an observed surface will be bounded insteading of being forced to conform to a particular parametric form. Decision criteria generally form opposing inequality conditions, and an acceptable solution requires showing the existence of a model structure that satisfies decision criteria. In other words, the problem becomes one of demonstrating feasibility. For example, the problem of graspability (Example 2.1) can be posed by changing the superquadric equation to an inequality describing enclosing volumes. Then an object is graspable if a graspable enclosing superquadric volume for the object can be found.

A surface-based representation of objects can be retained if the task can be reformulated using *local* characteristics of the surface. For example, graspability can be defined by specifying a system of constraints on local surface normals. Deciding about graspability corresponds to globally searching for local surface patches that satisfy the decision constraints.

5.4 Increasing Performance

There are three means of increasing the performance of the interval-based algorithms: using more powerful interval solution methods, exploiting parallelism, and reducing the amount of data that is processed. Initial investigations suggest that the first alternative, while attractive, does not appear to lead to improved performance in many cases. In fact the method we have described performs better than many algorithms that use Jacobian information.

The last suggestion has been implemented and tested [16] and improves performance significantly. The basic idea is to choose the data that most strongly affect interval reduction. It can be shown (details can be found in [16]) that *reduce*() applied to an interval vector of n parameters

needs at most $2n$ data values. In the linear case, these data values can be chosen based on the structure of the linear system. In the nonlinear case, the system Jacobian evaluated at the center of an interval provides sufficient information to make acceptable data choices. The Jacobian computation on a data item is much faster than applying reduce (which requires $3n$ function evaluations per data item), so the combined selection-reduction algorithm operates several time faster than applying reduction to all data items.

Parallelism can be exploited in several areas. At the lower level, most algebraic interval computations can be implemented by performing four arithmetic operations and four compares. By pipelining and parallel computation, interval arithmetic could be made to be nearly as efficient as normal point-based arithmetic. With selection, *reduce*() requires $6n^2$ identical operations. These operations can be carried out entirely in parallel. Moreover, data selection and *reduce*() can be pipelined to operate in parallel. At the highest level, *reduce*() itself can applied in parallel to interval vectors in the queue. Also, if there are several objects, then computation can be spread across the different objects. Some initial experiments with the latter type of parallelism using the Linda [3] system have led to algorithms with nearly linear increase with respect to number of processors. Further investigation of these ideas is under way.

6 Conclusions

The central idea in this paper is the computation of incrementally improved *sets* of models for data observed with bounded error. Given this capability, it is possible to guide the solution refinement process based on task-specific decision criteria, and to thereby improve its decision-making reliability and performance. This methodology has several advantages:

- It explicitly incorporates the notion of *decision making*, and, compared to conventional methods, it both reduces the amount of work required to reach a decision and increases the likelihood of reaching the correct decision with sparse or incomplete data.
- It can be proven correct and convergent under very weak structural and statistical assumptions.
- Problems are straightforward to implement, test, and debug.
- The approach lends itself naturally to parallelism.
- Decision-making with multiple objects and multiple decision choices can also be implemented easily and efficiently.

In addition, the approach supports systems containing both equalities and inequalities, making it an extremely general approach to the problem of sensor data fusion and sensor-based decision making.

The specific techniques presented in this paper suffer from two major weaknesses. First, they are only suitable for problems that can be characterized with bounded observation error, known

[32] H. S. Witsenhausen. Sets of possible states of linear systems given perturbed observations. *IEEE Journal on Automatic Control*, AC-13(5):556–558, October 1968.