Experiments with a Network-Based Geometric Reasoning Engine

Robert B. Fisher Mark J. L. Orr
Dept. of Artificial Intelligence, University of Edinburgh, Scotland
5 Forrest Hill, Edinburgh EH1 2QL, Scotland, United Kingdom

Abstract

Fisher and Orr (1988) described a geometric reasoning engine based on a value passing constraint network. This paper reports on more recent results from this research: several experiments on detecting inconsistency between constraints, how to bind degrees of freedom in a revolute joint, some results on propagation of error in the networks, and performance on a large three dimensional reasoning problem involving three revolute joints.

Keywords: geometric reasoning, three dimensional scene analysis

1 Introduction

Fisher and Orr (1988) described a geometric reasoning engine based on value passing constraint networks. The networks are created based on an underlying algebraic description of three-dimensional geometric relations and transformations. These networks implement the five key visual geometric reasoning functions: locate, predict, transform, inverse and merge (Orr and Fisher 1987). The networks also produce tighter bounds than simply using symbolic algebra and are have greatly improved efficiency (over, e.g. ACRONYM (Brooks 1981)).

Moreover, we observed that the forms of the algebraic constraints tended to be few and repeated often, and hence standard subnetwork modules could be developed, with instances allocated as new geometric constraints were identified during model matching. For example, network modules exist for constraints like: "a model vector is transformed to a data vector" and "a data boundary must lie inside a transformed model boundary".

This paper describes and discusses some recent experiments and results obtained using the networks. In particular, we describe:

- experiments with detecting inconsistency between constraints,
- how one can bind degrees of freedom in a revolute joint,
- some results on propagation of error in the networks, and
- performance of a large network with three revolute joints.

These are described individually in sections 4 through 7. We start with a brief review of the main concepts behind the network-based geometric reasoner:

1. evaluating algebraic constraints in a value-passing network,
2. specifying the geometric problem as algebraic constraints and
3. partitioning the networks into prototypical modules.

Acknowledgements

This work was funded by the University of Edinburgh and Alvey Grant GR/D/1740.3. I'd like to thank J. Hallam and M. Cameron-Jones for help with the text and work.

2 Numerical Constraint Networks

The computation implemented by an algebraic network is given by a set of algebraic equalities and inequalities (determined by the problem being solved). Because we are interested in estimating upper (SUP) and lower (INF) bounds on all variables, the arithmetic relationships are reformulated into interval form (Alefeld and Herzberger 1983).

We must then estimate the bounds on the variables, given the algebraic relationships in which they play a part. ACRONYM's constraint manipulation system (Brooks 1981) used a symbolic algebra technique to estimate bounds on all quantities. Because this symbolic algebra method was slow and did not always give tight bounds, a new network approach was designed. The construction and evaluation of these networks was described in Fisher and Orr (1988), but a quick overview is given now.

The basis of the network was the propagation of updated bounds, through functional units linked according to the algebraic problem specification. A simple example is based on the inequality:

$$A \leq B - C$$

By the SUP/INF calculus (e.g. Brooks 1981), the upper bound of A is constrained by:

$$SUP(A) \leq SUP(B) - INF(C)$$

as are the lower bound of B and upper bound of C:

$$INF(B) \geq INF(A) + INF(C) \qquad SUP(C) \leq SUP(B) - INF(A)$$

Thus, one can use the value of $SUP(B) - INF(C)$ as an estimated upper bound for A, etc. These relationships are used to create the network for this single inequality, which is shown in Figure 1. As new bounds on B are computed, perhaps from other relationships, they propagate through the network to help compute new bounds on A and C.

There are two advantages to the network structure. First, because values propagate, local improvements in estimates propagate to help constrain other values elsewhere. Even for just local problems, continued propagation until convergence produces better results than the symbolic methods of ACRONYM. Second, these networks have a natural wide-scale parallel structure (e.g. 1000+) that might eventually lead to extremely fast evaluation. One disadvantage is that all bounding relationships must be pre-computed as the network is compiled, whereas the symbolic approach can be opportunistic when a fortuitous set of constraints is encountered.

For a given problem, the networks can be complicated, particularly since there may be both exact and heuristic bounding relationships. For example, the network expressing the composition of two 3D reference frame transformations to give a third contains about 2000 function nodes (of types '+', '-', '*', '/', 'sqrt', 'max', '≥', 'if', etc.). Though the network is formulated for parallel evaluation, simulated serial evaluation is also fast, because small changes are truncated to prevent trivial propagations, unlike in other constraint propagation approaches (see Davis 1987). Convergence is guaranteed (or inconsistency detected) because bounds can only tighten (or cross) and there must be a minimum change for propagation to occur.

3 Geometric Reasoning Networks

The key geometric reasoning data type for high-level computer vision is the position, that represents the relative spatial relationship between two visual features (e.g. world-to-camera, camera-to-model, or model-to-subcomponent). A position consists of a 3-vector representing relative translation and a unit quaternion (4-vector) representing relative orientation (of the form *(cos(0/2),sin(0/2)w)* for a rotation of *0* about the axis w).

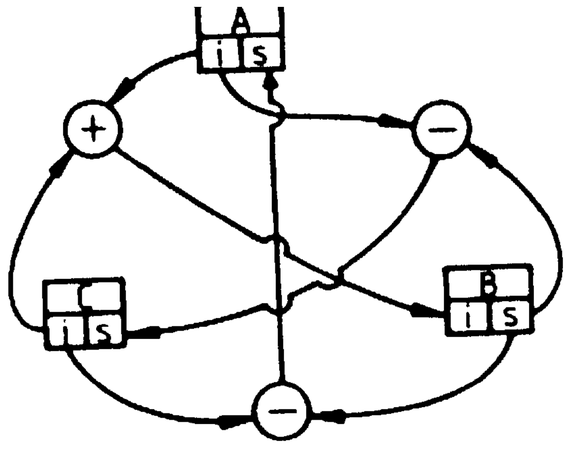

Figure 1: Small Constraint Network Example

The key geometric relationships concern relative position and have two forms – exact and partially constrained. An example of an exact form is: let object A be at global position $T_{g-A} = (\vec{r}_A, \vec{t}_A)$, (rotation $\vec{r}_A$ and translation $\vec{t}_A$) and object B be at $T_{A-B} = (\vec{r}_{AB}, \vec{t}_{AB})$ relative to A. Then, the global position of B is:

$$T_{g-B} = T_{g-A} * T_{A-B} = (r_{AB} * r_A, r_{AB} * t_A * r'_{AB} + t_{AB})$$

where * is the quaternion multiplication operator and " ' " is the quaternion inverse operator.

A partially constrained position is given by an inequality constraint:

$$t_{Az} \geq 50$$

which means that the z component of A's global position is at least 50. Such a constraint might arise from some *a priori* scene knowledge, or from observation of a fragment of a surface.

Other relationships concern vectors or points linked by a common transformation, as in $T\vec{v}_1 = \vec{v}_2$, or the proximity of points or vectors:

$$|\vec{p}_1 - \vec{p}_2| < \epsilon$$

Sets of these constraints are generated as recognition proceeds.

Next, we would like to estimate the values of the constrained quantities (e.g. object position) from the known model and data values and their relationships from a set of constraints. Alternatively, we would like to determine that the set of constraints is inconsistent, so that the hypothesis can be rejected.

A complication is that each data measurement may have some error or uncertainly, and hence the estimated values may also have these. Or, a variable may be only partially constrained in the model or by o *priori* scene information. Hence, each numerical quantity is represented by an interval and is bounded using the network methodology described in section 2.

The creation of the networks is time-consuming, requiring a symbolic analysis of the algebraic inequalities. Fortunately, there is a natural modular structure arising from the types of problems encountered during scene analysis, where most geometric constraints are of the type described above. Hence, it is possible to pre-compile network modules for each relationship, and merely connect a new instance of the module into the network as scene analysis proceeds. To date, we have identified and implemented network modules for:

SS - two scalers are close in value
PP - two points are close in location
VV - two vectors point in nearly the same direction
DOTS - the dot product of two 3-vectors is above a scaler

UNIT3 - enforcing a unit 3-vector constraint
UNIT4 - enforcing a unit 4-vector constraint
TP - a transformation links a pair of points
TV - a transformation links a pair of vectors
TV2 - a transformation links two pairs of vectors
TT - a transformation maps from one position to a second by a third relative position
P2V - a vector can be defined by two points
QWθ - a quaternion is equivalent to an axis and an angle

For example, the UNIT3 module is defined by the following constraints between the components of a vector (p_x, p_y, p_z):

$$p_i = (1 - p_j^2 - p_k^2)/p_i$$

$$|p_i| \leq \sqrt{1 - p_j^2 - p_k^2}$$
$$|p_i| \leq 1.7321 - |p_j| - |p_k|$$
$$|p_i| \leq 1.4143 - |p_j|$$
$$|p_i| \leq 1$$

We now include a simple example of network use. Suppose subcomponents B and C are rigidly connected to form object A. Given the estimated positions of the subcomponents in the global coordinate system, T_{g-B} and T_{g-C}, and the transformations between the object and subcomponent coordinate systems, *TA-D* and T_{A-C}, then these can be used to estimate the global object position, T_{G-A} using two instances of the "TT" module listed above. Figure 2 shows this network. Notice that each subcomponent gives an independent estimate of T_{G-A}, SO that the network keeps the tightest bounds on each component of the position. Any tighter resulting bounds then propagate back through the modules to refine the subcomponent position estimates.

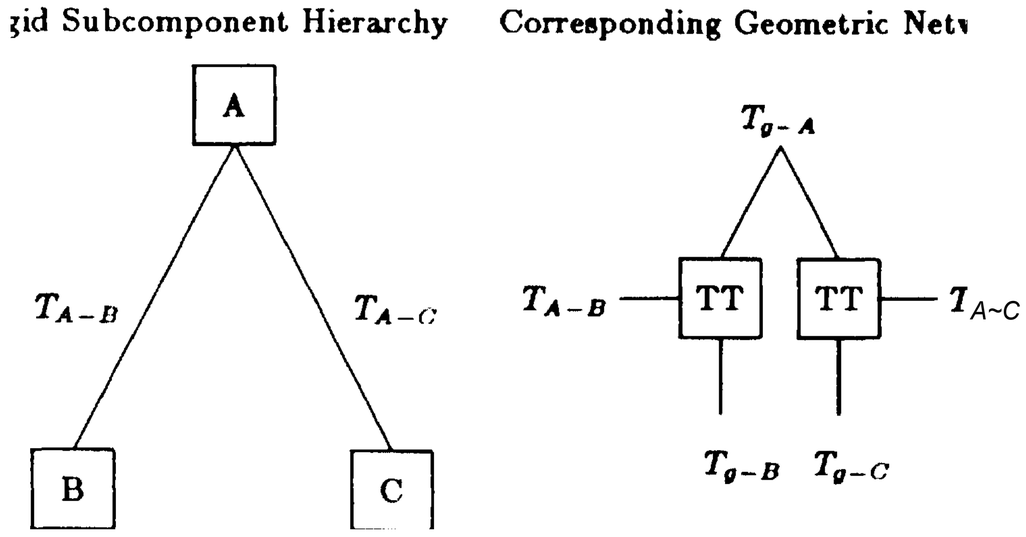

Figure 2: *A Simple Geometric Reasoning Network*

4 Binding Degrees of Freedom

Binding degrees-of-freedom is a typical visual geometric reasoning problem, such as finding the positional parameters, as above. Another example is estimating an intrinsic linear degree-of-freedom, as in a prismatic robot joint. Its underconstrained position can be modeled by using a variable in the reference frame translation. Then, bounds on this variable can be estimated by using the network method to compute its reference frame relationship relative to its main object.

The more difficult case is a rotational degree-of-freedom, as in a robot revolute joint, which we now consider in more detail. The problem with this case is that there is no element of the rotation specification that exactly corresponds to the degree-of-freedom (unlike the translation case). Fortunately, the cos(0/2) component of a quaternion can be related to a joint's rotation. Figure 3 shows how to exploit this *cos(6/2)* relationship in a subnetwork that solves the problem.

The key subcomponents are *B* and *D,* with their estimated global positions *TG-B* and T_{G-D} From the model, we know which vector in each of their local reference frames is the axis of rotation *(axisb* and *axisd).* Also, from the model we have reference vectors, re/6 and *refd,* such that when they are aligned, the rotation is zero. These two

positions and four vectors are inputs into the joint angle subnetwork, at the top of figure 3.

The two TV2 modules at the top map each pair of model vectors to a corresponding pair of scene vectors. Two of these map to the common axis vector. From these we can estimate the joint rotation in quaternion rotation form. This is done by the third TV2 module in the middle of the network. It rotates the global *outrefb* vector to the global *outrefd* vector, while maintaining the direction of the global axis vector *jointaxis*.

Finally, to extract the joint angle itself, we use a QWθ module that maps a rotation quaternion to a $(\theta, \vec{\omega})$ axis-angle form. Here, vector $\vec{\omega}$ is also the same as the global rotation axis *jointaxis*, so this helps the module. (The angular output is given in the form $cos(\theta)$).

The following is an example of the joint network:

$$axisb = axisd = (0, -1, 0)$$
$$refb = refd = (1, 0, 0)$$
$$T_{g-B} = ((0.20, -0.61, -0.45, -0.61), (0, 0, 0))$$
$$T_{g-D} = ((0.31, -0.43, -0.39, -0.75), (0, 0, 0))$$

with very small intervals (0.00001 width) correctly gives an estimated rotation: $cos(\theta) = [0.863, 0.864]$.

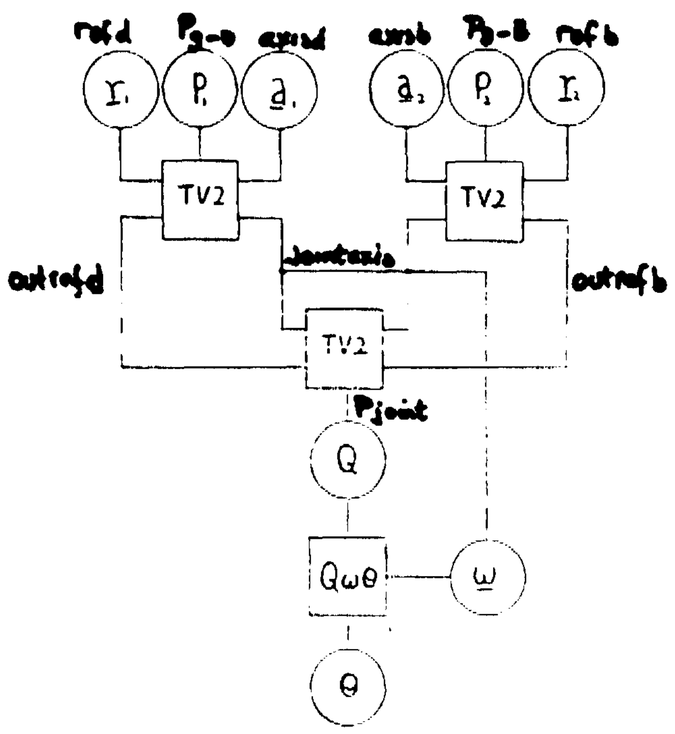

Figure 3: Revolute Joint Subnetwork

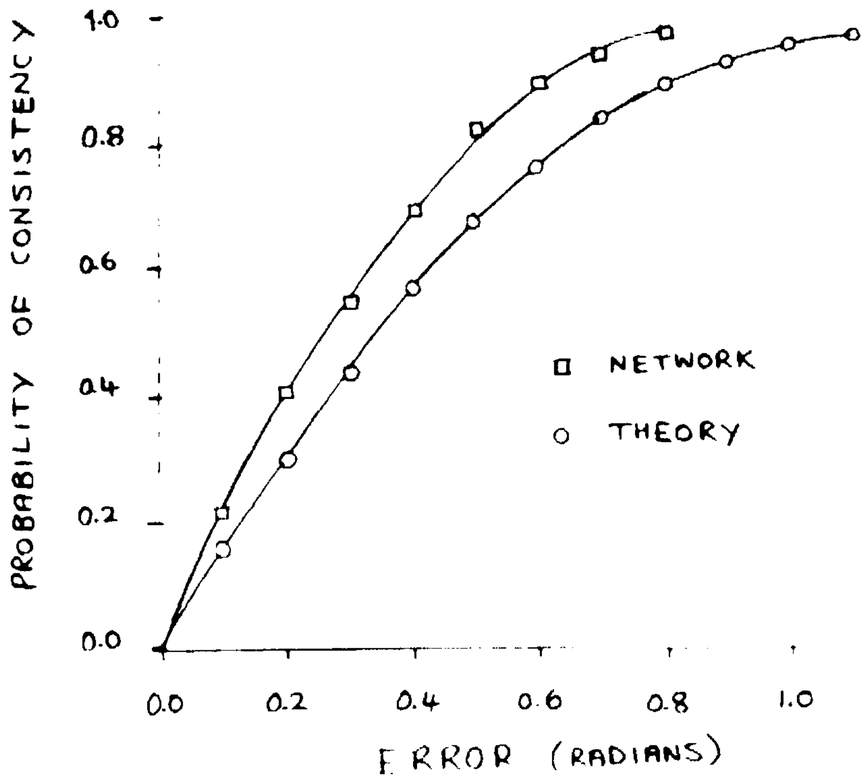

Figure 4: The probability of consistency as a function of ϵ

5 Detecting Inconsistency

The bound estimation algorithm is an incomplete decision procedure. If a network is inconsistent ($INF(V) > SUP(V)$ for any variable V) then there is definitely no solution. However, a consistent network does not guarantee that a solution exists. This is a problem because the network's ability to detect inconsistent sets of constraints relates to the vision program's ability to reject incorrect hypotheses. We would therefore like to investigate two questions about sets of constraints that have no solution but that nevertheless lead to consistent networks: (1) how likely are such problems and (2) typically "how close to being soluble" are they.

As our test case we take the problem of finding the rotation that maps a pair of 3D model direction vectors into a corresponding pair of 3D image vectors. This is not only a typical problem in geometric reasoning but also a basic one in that other constraints involving multiple matched pairs of directions and locations can be solved by decomposing them into one or more problems of this sort. For our present purposes we assume that the model vectors are known precisely while the image vectors are known only within certain errors. We further assume the errors are isotropic, that is, a nominal vector $\vec{v}$ is enclosed within an error cone of internal angle ϵ to yield the interval vector $< \vec{v}, \epsilon >_I$.

A solution exists only if the model vectors can be rotated to lie within the error cones of their corresponding image vectors. Let $\vec{v}_1$ and $\vec{v}_2$ be the (exact) model vectors and $< \vec{v}'_1, \epsilon >_I$ and $< \vec{v}'_2, \epsilon >_I$ be the (inexact) image vectors with isotropic error ϵ radians. Then a solution exists if:

$$cos^{-1}(\vec{v}'_1 \circ \vec{v}'_2) - 2\epsilon \leq cos^{-1}(\vec{v}_1 \circ \vec{v}_2) \leq cos^{-1}(\vec{v}'_1 \circ \vec{v}'_2) + 2\epsilon$$

For different values of ϵ we estimate the probabilities that randomly chosen vectors, will result in a soluble problem and consistent network by performing many trials. The results are shown in the graph of figure 4. The squares show the probability of network convergence while the dots show the probability of generating a soluble problem. The latter can be calculated analytically and is shown by the curve in figure 4 which is the function:

$$p(\epsilon) = \frac{1}{4}(2 - 2cos(2\epsilon) + (\pi - 2\epsilon)sin(2\epsilon))$$

The probability of the network being inconsistent given that the problem has a solution is zero for any value of ϵ, but the probability of the network being consistent when no solution exists is significant as figure 4 shows.

We now define:

$$s = \frac{|\theta - \theta'|}{2\epsilon}$$

as a measure of "distance from solution". The problem is soluble iff $s \leq 1$. For values of s larger than but close to 1 the problem is "only just" insoluble. By again performing many trials (for fixed $\epsilon = 0.1$) we compiled histograms of values for s in trials that were predicted by the above to have no solution and led to a network state that was (a) inconsistent or (b) consistent. The results are shown in figure 5.

Figure 5 part (a) shows that there is a wide distribution of s values between $s = 1$ and $s = 12$ for insoluble problems correctly detected by the network. On the other hand, part (b) shows that insoluble problems that the network could not detect are concentrated mostly between $s = 1$ and $s = 2$ and are consequently close to being soluble. We therefore conclude that the network will not give misleading results when problems are far from solution and that in practice the incompleteness problem has only a limited effect.

6 Propagation of Uncertainty

Because each numerical quantity is represented as an interval, and because most geometric calculations involve several arithmetic functions, it is possible for error intervals to grow as they propagate through the network. Hence, we examine some of the numerical performance of the network modules. This section discusses results for two key modules, TT and TP, and the joint angle estimation network.

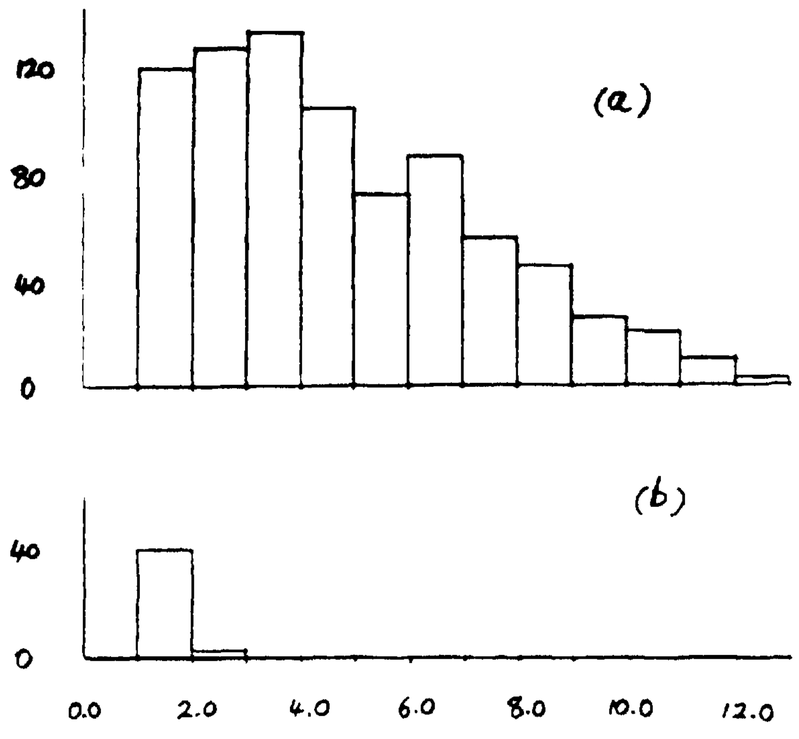

Figure 5: The distribution of s for insoluble problems with (a) inconsistent and (b) consistent networks

By examining the algebraic relationships involved in the composition of reference frame module (TT), it can be shown analytically that if each orientation component of a random position T_{a-b} has error δ and each orientation component of a random position T_{b-c} has error ϵ, then each component of the resulting position $T_{a-c} = T_{a-b} * T_{b-c}$ has (statistical) error:

$$\frac{16(\delta + \epsilon)}{3\pi}$$

assuming that both δ and ϵ are small. This result has been verified by simulation. Figure 6 shows the empirical results, which taper off as the maximum error volume is reached. In essence, the results mean that the output error width is about 1.7 times the input error width.

A similar analysis of the transformation of a location (TP module) shows that each translation component has output error:

$$\frac{16L\epsilon}{\pi}$$

where ϵ is the input rotation quaternion error on each component, and L is the distance of the input point from the origin. This is sensible, because small rotation errors amplify along a long baseline, to produce larger translation errors. This result has also been verified empirically.

We have not found an analytic form for the error in joint angle estimation, as a function of input position error, but empirical results

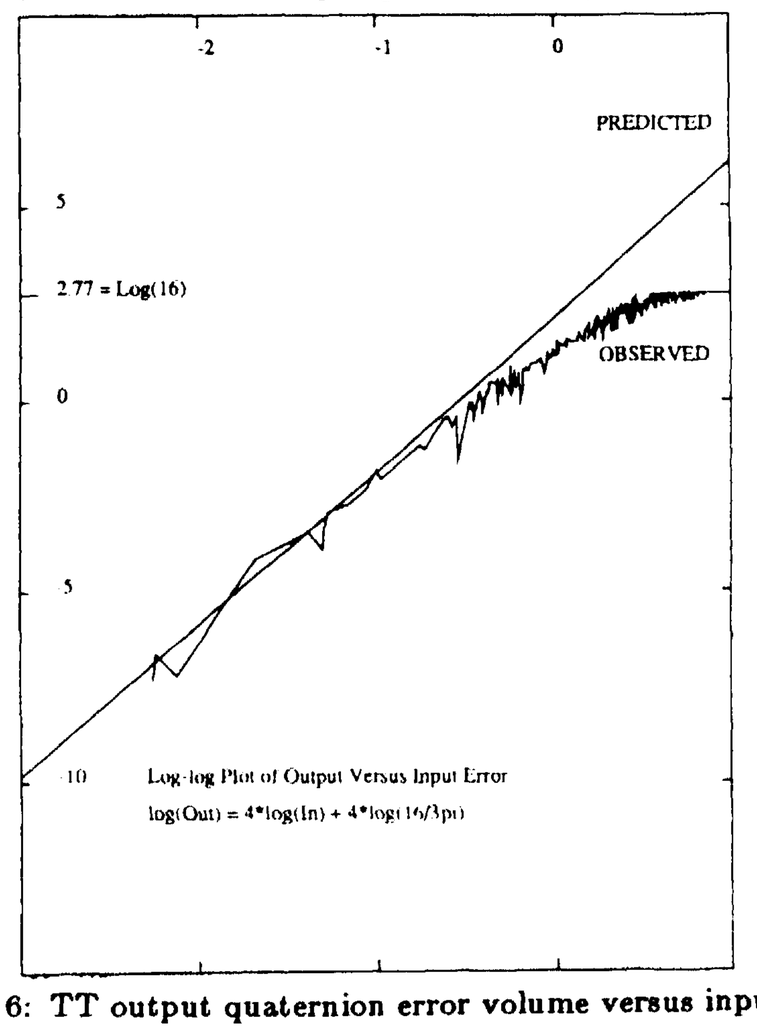

Figure 6: TT output quaternion error volume versus input error

show that the output error varies approximately linearly with input error and has magnitude about 5.5 times as big. The results were generated by perturbing each component of T_{g-B} and T_{g-B} by an error e for a randomly chosen rotation, and then looking at the output error of the *cos{6)* rotation estimate. The results also show that the error reaches a maximum when the input error reaches about e = 0.18, so the input positions need to be somewhat constrained for joint angle estimation to work.

Figure 6 also show that there is some variability in the results (besides the underlying trend) which is undesirable. One contributor to the problem is several special cases to the TV2 module that solve for the rotation using a different set of bounds from the main case, and hence have different numerical properties.

7 A Large Network Example

Figure 8 shows the full network generated for analysing the position of a robot in a test scene. As before, the boxes represent transformations, but there are more types used here. The TPn boxes stand for n instances *of* a TP module. The circular "Jn" boxes represent three identical instances of subnetworks allocated for transformations involving joint angles, which are omitted to simplify the diagram (each contains 7 network modules). The core of the subnetworks was shown in figure 3, but some additional modules are added to align reference frames. The relative positions of objects are given by the *T* structures, such as $T_{g-R,}$ which represents the position of the robot in the global reference frame. These are linked by the various transformations. Links to model or data vectors or points are represented by the unconnected segments exiting from some boxes.

The top position T_{g-c} *is* the position of the camera in the global coordinate system, and the subnetwork to the left and below relates features in the camera frame to corresponding ones in the global coordinate system. Below and right is the position T_{G-R} of the robot in the global coordinate system and the position *TC-R,* of the robot in the camera coordinate system, all linked by a *TT* position transformation module. Next, to the bottom left is the subnetwork for the cylindrical robot body T_{g-B}. The "J1" node connects the robot position to the rest ("link") on the right, whose position is T_{G-LK}. Its left subcomponent is the rigid shoulder ASSEMBLY (SH) with its subcomponents, the shoulder body (SB) and the small shoulder patch (SO). To the right, the "J2" node connects to the "armasm" ASSEMBLY (UA), linking the upper arm (U) to the lower arm (L), again via another joint angle (J3). At the bottom are the modules that link model vectors and points to observed surface normals, cylindrical axis vectors, and central points, etc. Altogether, there are 61 network modules containing about 96,000 function nodes.

The network structure closely resembles the model subcomponent hierarchy, and only the bottom level is data-dependent. There, new nodes are added whenever new model-to-data pairings are made, producing new constraints on feature positions.

Evaluating the complete network from the raw data requires about 1,000,000 node evaluations in 800 "clock-periods" (thus implying over 1000-way parallelism). Given the simplicity of operations in a node evaluation, a future machine should be able to support easily a 1 microsecond cycle time. This suggests that an approximate answer to this complicated problem could be achieved in about one millisecond.

Because the resulting intervals are not tight, confidence that the mean interval value is the best estimate is reduced, though the bounds are correct, and the mean interval values provide useful position estimates. To tighten estimates, a post-processing phase progressively shrinks the bounds on the intervals. Each position variable was examined to see if its interval was tight. If not, it was reduced in size by 10% and the new bounds were then allowed to propagate through the network. For the robot example, this required an additional 12,000 cycles, implying a total solution time of about 13 milliseconds on our hypothetical parallel machine.

Using the geometric reasoning network, the numerical results for the whole robot in the test scene are summarised in Table 1. Here, the values are given in the global reference frame rather than in the camera reference frame.

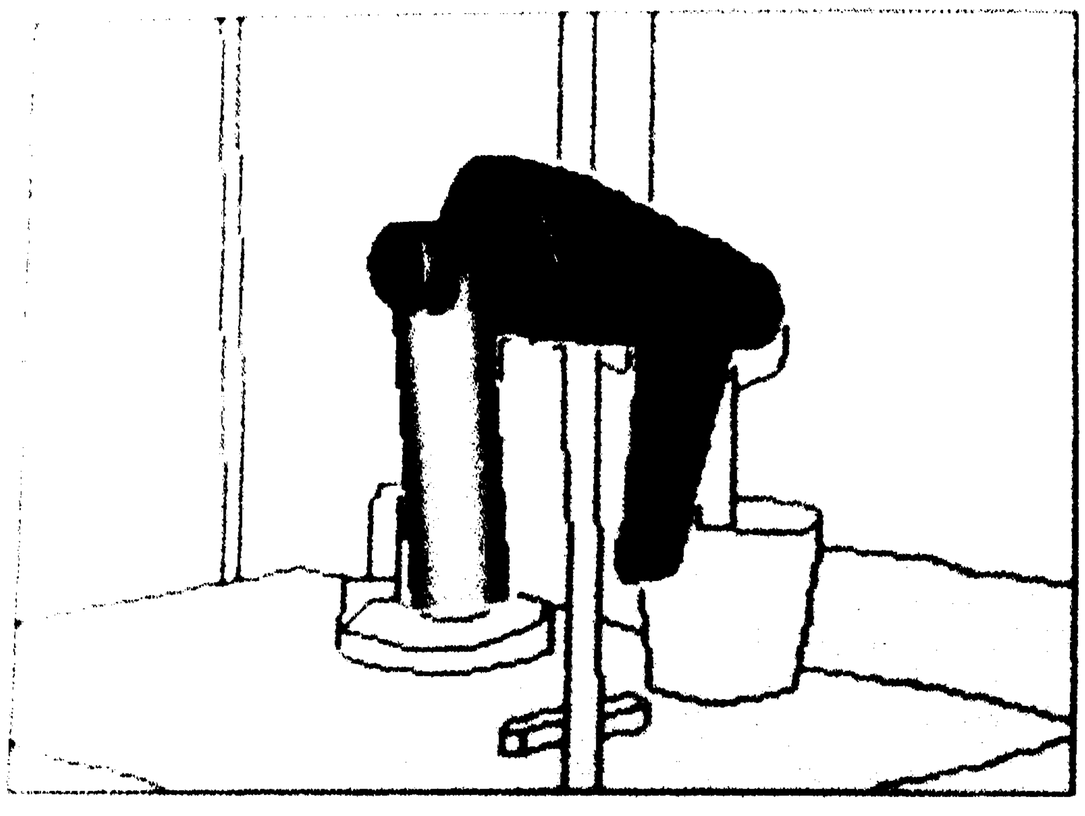

Figure 7: Recognized Complete Robot Using Network Method

Table 1: Measured And Estimated Spatial Parameters

PARAMETER	MEASURED	ESTIMATED
X	488 (cm)	487 (cm)
Y	89 (cm)	87 (cm)
Z	554 (cm)	550 (cm)
Rotation	0.0 (rad)	0.038 (rad)
Slant	0.793 (rad)	0.702(rad)
Tilt	3.14 (rad)	2.97 (rad)
Joint 1	2.24 (rad)	2.21 (rad)
Joint 2	2.82 (rad)	2.88 (rad)
Joint 3	4.94 (rad)	4.57 (rad)

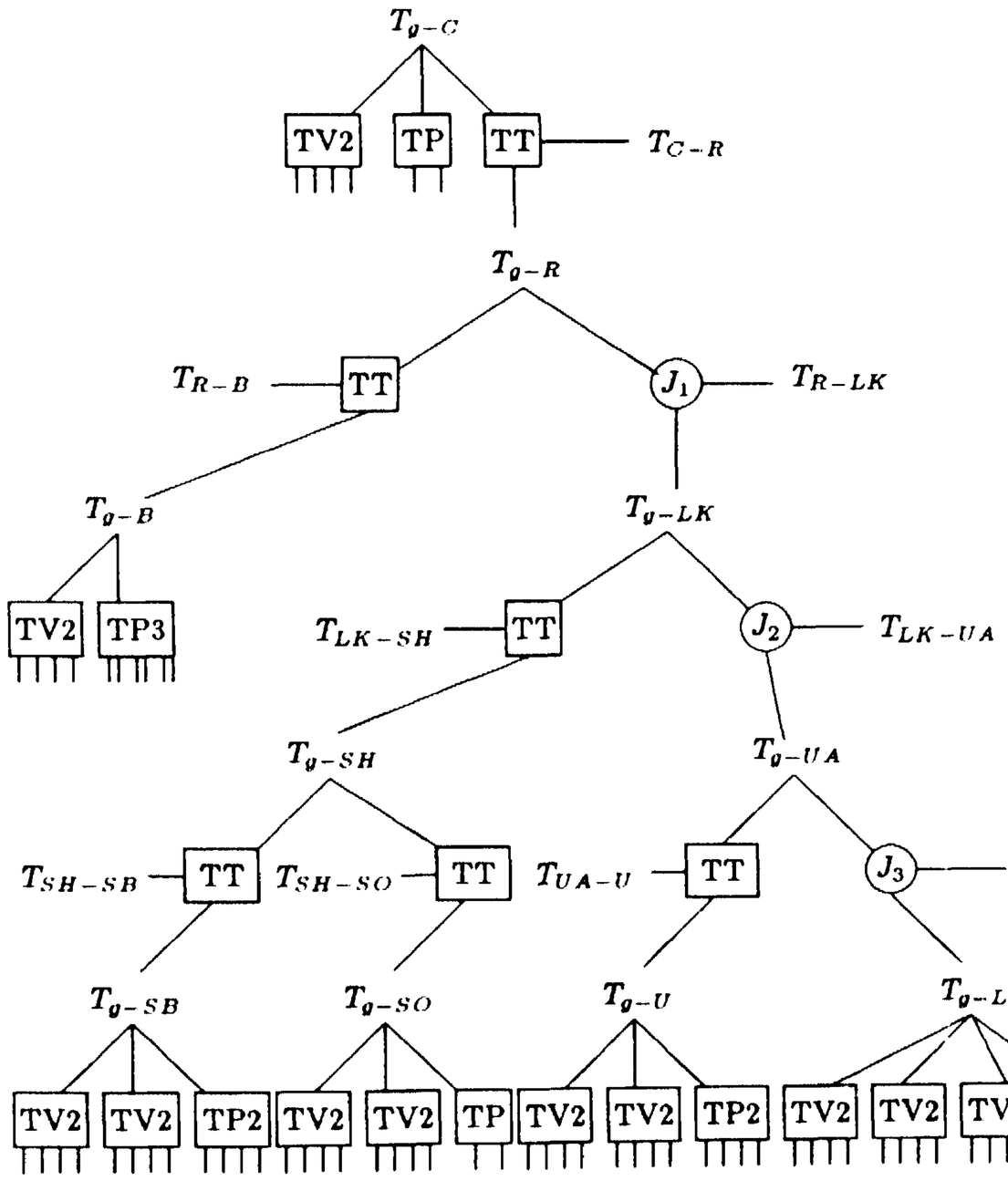

Figure 8: *Robot Scene Geometric Reasoning Network*

The results of the position estimation can been seen more clearly if we look at Figure 7, that shows the estimated robot position overlaying the original scene. Now, we are using the camera position of the whole robot assembly plus the estimated joint angles from above. While the positions of the individual components are reasonable, the accumulated errors in the position and joint angle estimates cause the predicted position of the gripper to drift somewhat from the true position. The iterative bounds tightening procedure described above produced this result, which was slightly better than the one-pass method.

8 Discussion

We have seen how the network methods can detect inconsistency and estimate degrees-of-freedom. From the examples given, it is obvious that this network approach can solve complex geometric reasoning problems, has an incremental modular structure and has potential for wide-scale parallelism. Unfortunately, after experience with use of the network approach, it has become obvious that there are also problems, particularly when working with noisy data.

One inherent problem concerns the ambiguity in the quaternion rotation representation, as the rotation *(qo, q1*, q2, q3) is equivalent to the rotation *(—qo, — q1* — q2, *—q3)*- When coupled with the interval arithmetic, sometimes the bounds on terms cannot choose between alternatives, producing unnecessarily large bounds. For example, the q_0 term may acquire a bound [-a - e, + a + c], when the bound (+a - c, + c] may be equally satisfactory. It is not possible to require that *qo > 0* always, because the composition of some rotations satisfying this condition may produce a rotation that violates this (using the standard quaternion rotation formulation). A related problem also occurs with a tolerance c about qo terms near one, creating intervals [l — e, l] and [-1, -1 + c].

These problems suggest that the model-based reasoning program may have to analyse the results and occasionally split or introduce alternative cases, given an understanding of the representation problems.

While the error analysis given in section 6 showed that errors usually do not grow too quickly, when we look at the robot network (Figure 8), we still find a problem. In particular, it turns out that the chain of reference frame transformations from the robot in the global reference frame $(T_{g\text{-}R})$ to the data vectors involved in the lower arm position (below $T_{g\text{-}.L}$) involve ten TV2 and three TT modules. Assuming each has an output interval width of 1.7 times the input interval width means that the combined contribution of the lowerarm evidence to the robot has an error interval width of $(1.7)^{13}$ = about 1000 times the input error interval. As the maximum output interval can be only [-1,+1) for the rotation, this implies that the input uncertainly can be at most about 0.01, which is unreasonably strict.

However, this error analysis must also consider that many of the reference frame transformations are the identity transformation, which create little error (and need not even be introduced into the network). Further, though the estimates may not propagate all the way to the robot global position node, they will help constrain more closely linked structures.

It should be noted that geometric reasoning involving a multiple joint articulated object is a genuinely complicated problem, whereas most object recognition problems have a much flatter model hierarchy, involving fewer compound degrees-of-freedom, if any at all.

A final problem posed by the interval approach is that of finding the "best" estimate for a parameter, when the intervals are large, particularly with rotation parameters. While integrating enough randomly distributed errors will cause the widths of the intervals to converge to a "correct" answer, we seldom have more than a few measured values (e.g. surface normals, curvature axes). Consequently, a statistical technique (e.g. Durrant-Whyte 1987) or a final least-squared error position refinement step also seems attractive, though they might have problems handling rotational degrees-of-freedom.

Though research on the efficient use of these networks is continuing, problems overcome by the new technique included the bounding of transformed parameter estimates and partially constrained variables and effective detection of geometric inconsistencies. The network module decomposition means that networks can be constructed incrementally as scene analysis proceeds. The network also has the potential for large scale parallel evaluation. This is important because a considerable portion of the processing time in three dimensional scene analysis is spent doing geometric reasoning.

Bibliography

1. Alefeld, G. and Hersberger, J., 1983, Introduction to Interval Computations, Academic Press.

2. Brooks, R.A., 1981, "Symbolic reasoning among 3-D models and 2-D images", Artificial Intelligence, Vol 17, p285.

3. Davis, E., 1987, "Constraint Propagation with Interval Labels", Artificial Intelligence, Vol 32, p281.

4. Durrant-Whyte, H.F., 1987, "Uncertain geometry in robotics", Proceedings of the IEEE Conference on Robotics and Automation, vol.2, p851.

5. Fisher, R.B., 1989, From Surfaces to Objects. Computer Vision and Three-Dimensional Scene Analysis, John Wiley and Sons, London, 1989.

6. Fisher, R. B., Orr, M. J. L., 1988, "Solving Geometric Constraints in a Parallel Network", Image and Vision Computing, Vol 6, No 2.

7. Orr, M.J.L. and Fisher, R.B., 1987, "Geometric Reasoning for Computer Vision", Image and Vision Computing, Vol 5, p233.