CS615
Term Project

Richard Wagner

rwagner@usc.edu

28 February, 1994
PROJ1.DOC

A+ : outstanding

Extending Polygon Pose Sensing by Inscription to Three Dimensions

Abstract

Optical angle scanners can be used to sense the pose of a prismatic part in the plane. The planar case uses sensing of the part boundaries to define an inscribing cone for the part, and based on the angles from two or more sensor positions, a unique pose (location and orientation) for the part is determined. An analog of this technique in three dimensions uses a single binary image (black and white) video camera to sense the pose of a polyhedron resting on a table. The vertices in the camera's image define a three dimensional cone in the workspace. With appropriate preprocessing, the polyhedron's pose can be found from the cone angle information in time complexity of $O(n^4)$ for an average case polyhedron with n edges.

This paper defines the problem of finding the polyhedral pose from image information and describes algorithms for both the necessary preprocessing and real-time execution.

1.0 Introduction

Algorithms exist to extract edges from the silhouette of a polyhedron in the bitmap image produced by a video camera. If a video camera is fixed in a known position over a work table, with the right choice of lighting and background (table top) color, a convex polyhedral part will produce a convex polygonal image (see figure 1) at the focal plane of the camera. The polyhedron is mapped into the camera space in "perspective," which is the physical analog of a homogeneous transform widely used in the computer graphics field.

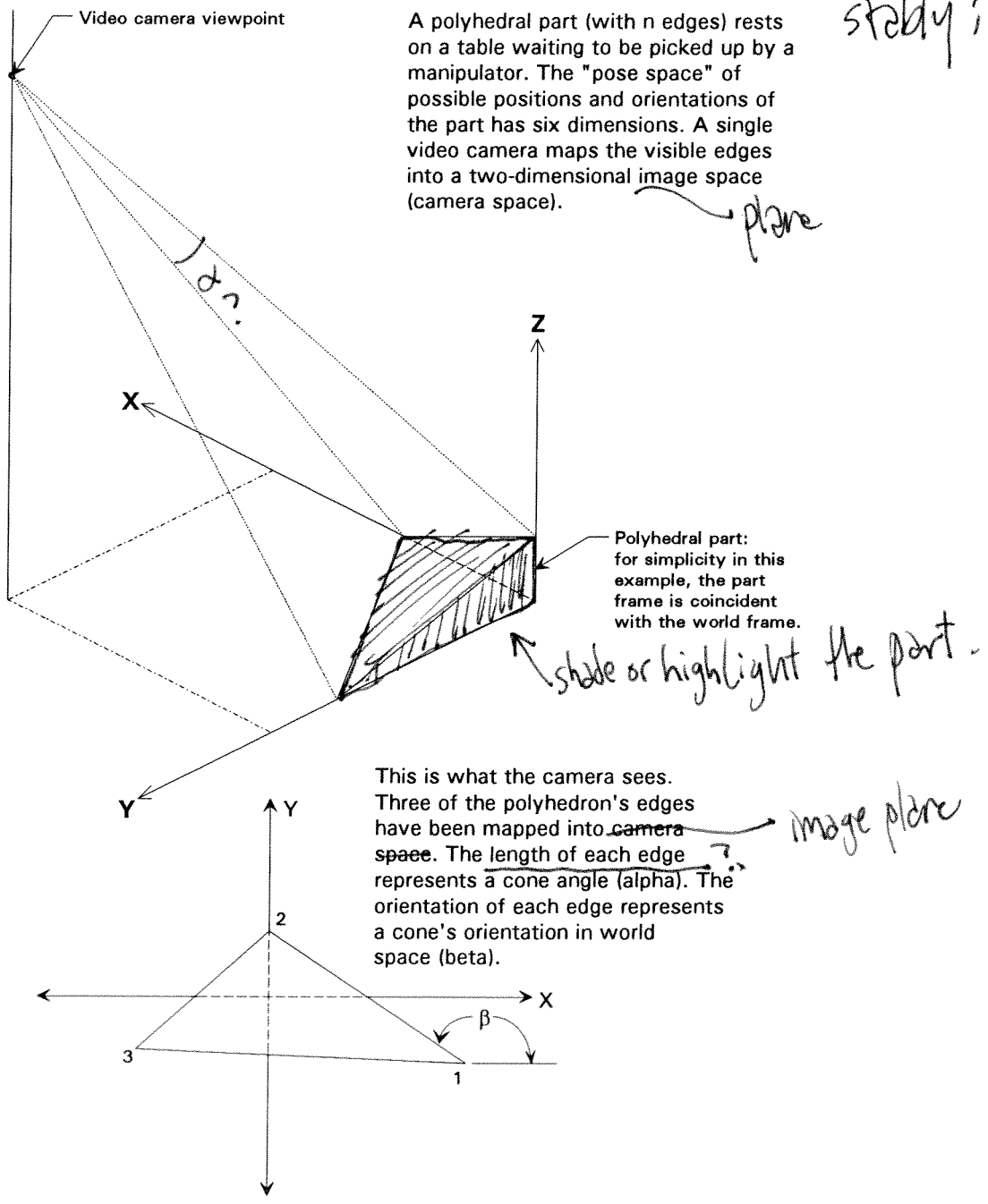

Figure 1

A subset of the polyhedron's edges (I call this set of edges a "ring" of the polyhedron[1]) will be mapped into the camera's image as a polygon.[2] Because the camera's position with respect to world space and the camera's focal length are known, the lengths of the edges in camera space are related to the angles of the cones formed by the view point and the polyhedron's edges. Each edge of the known polyhedron that is mapped into camera space is "inscribed" in its corresponding cone. This situation is analogous to the planar pose sensing by conic inscription described in "Sensing Polygon Poses by Inscription" by Jia and Erdmann [4].

As described in that paper, each triangle inscribed in a cone has a range of possible poses described by a piecewise elliptical continuous curve in a 3d pose space. For the case of the polyhedral edge, this reduces to the degenerate case of a 2-gon inscribed in the cone. Each edge in camera space might be any one of n edges of the polyhedron. Finding the conic pose equations for all n edges in all camera space edges has an upper bound on complexity that is exponential in n. However, by applying knowledge of a physical constraint (the polyhedron must be in a stable pose resting on one facet), preprocessing can be done to reduce the time complexity to $O(n^4)$. By assigning polyhedron ring edges to the camera space edges, we can construct a set of candidate configurations and then eliminate those configurations for which there is no solution to the conic inscription equations.

For a polyhedron that has unique edges (as in figure 1), only one candidate configuration can be valid. More generally, multiple solutions can be disambiguated by using edge orientation information (angle Beta in figure 1).

This project is closely related to the field of machine vision. I have surveyed this field and results indicate that my approach to this problem has been overlooked.[3] Portions of the algorithm have been implemented to demonstrate feasibility.

---

[1]For general solids, Shafer [6] calls this the "occluding contour."

[2]The degenerate case where the camera axis is vertical and centered (looking straight down at the polyhedron) is trivial because the silhouette of the polyhedron is invariant with rotation about the vertical axis.

[3]According to *Robot Vision* by ~~Klaus and~~ Horn [2], page 7, "It is known, for example, that binary image techniques are useful only when possible changes in the attitude of object are confined to rotations in a plane parallel to the image plane." A binary image is a two-level bitmap such as I suggest using.

2.0 Problem Statement

Consider a robot task involving the manipulation of a polyhedral part. Suppose the robot knows that the part is resting on a table. The robot knows *what* the part is but it does not know exactly *where* it is. This is where machine vision can enable the robot to determine the position and orientation ("pose") of the part so that the part may be grasped without error and positioned exactly as required by the task at hand.

In keeping with the RISC (Reduced Intricacy Sensing and Control) [2] paradigm, we wish to use a robust form of machine vision. The use of a binary state bit map ("black and white" image) will reduce image preprocessing time and will be relatively insensitive to photo interference.

I am defining this problem for a convex polyhedron for simplicity for this initial formulation. I see no reason why this approach might not be extensible to the whole class of simple polyhedra[4].

Using appropriate lighting and camera gain and contrast adjustments, a polyhedron resting on a table will appear on the camera's image plane as a silhouette, the outline of which forms a polygon. Each polygon edge can be easily defined, related to a cone angle based on camera properties, and stored in an array record along with orientation (Beta, in figure 1) information. The set of 2d cones (α_i) defined by the image edges forms a 3d cone. Because the polyhedron will touch each of the rays without penetrating the planes defined by the 2d cones, we can say that the polyhedron is *inscribed* in the 3d cone.

Given a polyhedron resting on a table, and a single silhouette image of that polyhedron formed by a video camera with a known position in world space and with known optical properties, the problem is to determine, first, if a unique pose for the part can be determined from the information, and second, determine what that pose is, i.e., what is the x-y position of the part on the table, what facet is it resting on, and what is its orientation about the vertical (z) axis.

---

[4]See "Future Work," section 6.0, below.

3.0 Related Work

As mentioned in the introduction above, Jia and Erdmann [4] applied conic inscription to polygons in the plane. Their 2d inscription algorithm has a time complexity of O(n).

Wallack and Canny [6] used crossbeam sensors to localize objects (find position and orientation) in the plane. Their algorithm produced similar results to the conic inscription approach, but used simpler sensors while relying on knowledge of a constant velocity of the part.

The related field in machine vision has been traditionally concerned with extracting line drawings of unknown polyhedral scenes from grayscale images [6]. Dohm reports a way to do this using three visible edges (not necessarily in the silhouette) of a known polyhedron [3]. The time complexity of that method was not reported, although a set of equations in the eighth power is involved.

4.0 Approach

In the discussions to follow, I will use the term "polyhedron" to mean the imaged polyhedron under consideration, and the term "polygon" to mean the image polygon. As mentioned above, a brute force approach has a time complexity that is exponential in n. More specifically, if the polygon has m edges and the polyhedron has n edges, there are n! ÷ (n-m)! combinations of possibilities to test.

Any polygon must correspond to an occluding ring of edges of the polyhedron. For brevity in the following discussions, I will use the term "ring" to mean any of the set of possible occluding rings of edges of the polyhedron[5]. The number of rings in a polyhedron is polynomial in n, and by using knowledge of the workspace configuration, the number of possible rings that correspond to the polygon in the camera image can be upper-bounded by n^2. There are not more than 2n/3 stable facets[6] that the polyhedron can be resting on [5], and for each of these stable facets there will be fewer than n distinct occluding contours available to the camera's view as the polyhedron rotates about a vertical axis.

We know further that the work table, while perhaps roomy for the task at hand, will constrain the part such that it is unlikely to show more than a small number of different silhouettes to the camera as it translates about the table (without vertical axis rotation with respect to the camera).

Using this knowledge, sets of possible rings can be constructed in a preprocessing operation and the polygon alphas can be compared to a subset of m-edged rings in real-time. The number of rings in the subset of possible rings is upper bounded by n^2. Thus, given an m-edged polygon, there will be an upper bound of mn^2 tests for a fit. Each polygon-to-ring comparison will have to be performed m times, one for each of the possible clocking positions of the ring relative to the polygon. This gives a total upper bound of m^2n^2 The number m^2n^2 is itself upper-bounded by n^4. Note that this number represents a worst case, such as for a 100 sided prism, where m will usually be close to n. A more common case will have a relationship approximating $m = sqr(n)$, resulting in a time complexity closer to $O(n^3)$.

The pose resolution flow can be summed up as follows:

1. Extract edges from the image silhouette and establish an array of alpha angles corresponding to the m edges.

2. For each of the set of m-edged possible rings, attempt to solve the m simultaneous inscription equations for each of m possible ring clockings. The rings for which solutions are found represent valid poses for the part.

---

[5]Rings of the polyhedron might not be unique in structure. I will discuss dealing with ambiguity later.

[6]"Stable facet" means that when the polyhedron rests on this facet, a normal projection of the part's CG onto the table will be within the boundary of that facet. By Euler's formula, the number of facets in a polyhedron is less than or equal to two thirds the number of edges.

3. For each valid pose, calculate the positions in the workspace of three of the ring points based on the cone leg lengths found in step (2), above, and the direction vectors of the cone rays.[7] By definition of a ring, the three points cannot lie on a line. These three points are used to find the transform to describe the positions of all the part vertices[8].

The key step in this flow is the solution of the m simultaneous inscription equations, and this is discussed in detail in the next section. As will be described, additional computations of order n are required, bringing the overall maximum time complexity to $O(n^5)$ with a nominal complexity proportional to n^4.

---

[7]While only three known points in the workspace are sufficient to define the pose of the part (because no three ring points can lie on one line), having multiple points available might be useful for overcoming problems associated with uncertainty.

[8]Single pose solutions will be the common case. Disambiguating multiple pose solutions is discussed in section 5.2.1.

5.0 Algorithm

This section describes the details of implementing the approach summarized above. I assume that the edge extraction and angle processing are done by a camera module and that the arrays of angles are available to the pose processor on demand.[9] In each of the algorithm discussions that follow, I will use the simple example of figure 1 to illustrate the geometry. While more complicated examples might reveal certain nuances, I feel that using a consistent simple example will best illustrate the fundamentals of the algorithm. To further define this example, the workspace constraints are based on the following:

The tetrahedron of figure 1 has its origin at the center of a square work table. The tetrahedron is one unit high, and the table is eight units on an edge. The camera is above one corner of the table and eight units above the table surface. The camera field of view is large enough to see all of the part in any position. The part will always be completely on the table.

5.1 Preprocessing

Preprocessing is done for each of the parts that are to be handled and is all done off-line. Knowledge of the part and workspace geometry is all that is required. This discussion describes the process for a single convex polyhedral part, but there is no reason that an actual system could not handle a large set of different parts, each with its own preprocessed dataset. All that would be required would be to have some way to notify the pose processor which type of part was being viewed.

The object of the preprocessing is to establish arrays of rings for the part. If a ring is not capable of being viewed considering the constraints of the workspace configuration, it is not included in the arrays. There will be one array for each of the edge-orders of rings. For the example tetrahedron there will be an array of 3-edged rings and an array of 4-edged rings.

For each of the stable facets (all four for the example tetrahedron), first translate the polyhedron (from part space) to the corner of the table that is farthest away from the camera (figure 2).

---

[9]Using a message-passing architecture will allow interchangeability of modules. For instance, going to a faster or higher resolution camera module would mean simply removing the old one and plugging in the new one to the local area network (LAN).

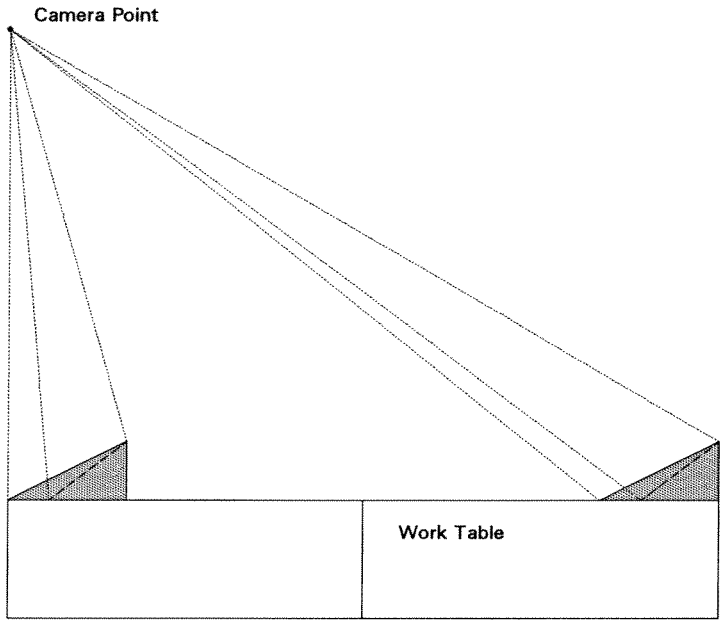

Figure 2

Taking the convex hull of the union of the set of part vertices and the camera point will establish the legs of the cones formed by the segments from the camera point to each of the ring vertices. Finding the convex hull can be done in O(n log n) time [5], and finding which edges have the camera point as one vertex takes time O(n). The ring is stored in the appropriate array, based on the number of edges in the ring.

Next the preprocessor moves the part (in its imagination only) to the opposite corner of the table, in its closest position to the camera, keeping the same orientation about the vertical axis of the part with respect to the camera. Then the convex hull is taken again, and if the resulting ring is the same as before, the preprocessor moves on to the next part orientation. If not, a binary search is initiated to find the transition point(s) to new rings. In the example, both positions of the part result in the same ring being defined. If that were not the case, the part would be moved half way to the camera and tested to see if there was a new ring, if so, it would moved half way again back, and if not, half way again forward. In this way, the transition point to a new ring is found (within some pre-specified accuracy). Once the (first) transition point is found, the remaining distance to camera proximity is similarly searched for remaining transition points. The resulting set of rings is also stored in their relevant arrays.

This completes the first orientation ring search for the first facet. Now, a finite set of part orientations is defined, and each of these orientations is searched for rings in a similar manner:

We can define a small orientation change increment by taking the shortest edge in the polyhedron and dividing it by the major diameter of the part. Taking the arc tangent of this ratio (mu) will give us a small enough angle to ensure that we can never skip a ring while rotating the part by this increment. Thus, the distance search described above is repeated for $2\pi/\mu$ increments, ensuring that all possible rings are captured for each stable facet of the part. This is repeated for each facet to complete the sets of rings.

5.2 Pose Calculation

Pose calculation is based on matching the camera polygon with the appropriate set of rings. In real-time, the pose processor is given a set of alphas corresponding to the image polygon edges. For the image polygon, some point (perhaps the lowest Y value) is chosen to be the starting vertex, and the vertices are numbered in counterclockwise order. For each edge, looking from inside the polygon outward, the vertex on the right corresponds to the right leg of its cone. Hence, each edge defines an angle alpha (α_i) and a right leg (r_i).

We know that one of m edges in one of the set of m-edged rings is inscribed in the cone of α_i. Drawing this relationship in two dimensions gives us figure 3:

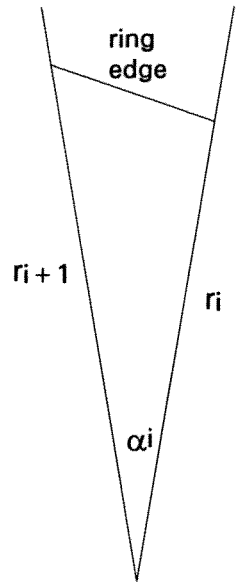

Figure 3

In this figure, r_i and r_{i+1} are unknown variables and the ring edge is one of m edges of some ring in the set of m-edged rings. If we let the length of the ring edge be symbolized by "d_i" and employing the Law of Cosines we have the following expression:

(1) $\quad d_i^2 = r_i^2 + r_{i+1}^2 - 2r_i r_{i+1}\cos(\alpha_i)$

This expression[10] represents a nonlinear system of m equations in m unknowns (given d_i) and I have not found an analytic solution for it. However, an iterative solution to this system

---

[10]For simplicity in the expression, the index "i+1" was used, but it should more properly read "(i mod m) +1" because of the circular nature of the geometric construct.

converges rather quickly[11] with a good first guess. Assuming that the two legs of the cone are equal, and using the distance from the center of the table to the camera point should be a good first guess to start off an iterative solution.

If within a preset number of iterations[12] there is no solution to the system of equations for the ring edges and clocking, the algorithm will reject that configuration as invalid, and evaluate the next one. If all clockings of a ring should be found to be invalid, that ring will be rejected, and the process repeated for all the rings in the set of m-edged rings. At the end of the process at least one valid ring configuration will be found, each representing a valid pose.

From the valid poses, the cone leg lengths found can then be used to find the ring vertex locations in the work space, based on the direction vectors of the legs (supplied by the camera processor).

5.2.1 Resolving Pose Ambiguities

A ring of a convex polyhedron is a structure of m edges connected at their vertices to form a circuit. The lengths of the edges and their relative orientations completely define the ring. It is very possible for a polyhedron to have two or more rings with the same structure. A good example is a cube, which contains two equivalence classes of four-edged rings and an equivalence class of six edged rings. Naturally, any polyhedron containing equivalent rings will generate several ambiguous poses, each corresponding to a ring of the valid equivalence class. Two rings need not have identical structure to generate two ambiguous poses: they need only have the same lengths of edges in the same order.

More generally, ambiguous poses may be generated by uncertainty, due to the inherent approximate nature of any iterative algorithm.

One approach to ambiguity in general, and in dealing with ambiguity due to uncertainty in particular, is to process multiple sets of data on the configuration in question. For example, adding a second camera most certainly will resolve the great majority of ambiguous cases.

Earlier, I suggested that the Beta angle information (see figure 1) could be useful in the case of ambiguous poses. This certainly seems to be a potentially rewarding approach, and I can see a number of ways to incorporate this feature. However, for the time being I am considering this to be out of scope for this project, and leave it for future work.

5.3 Uncertainty

As mentioned above, uncertainty (noise or low resolution in the camera image) can be responsible for ambiguous poses. A worse effect will result in the divergence of all ring tests so that no pose will be found. As above, adding a second (or more) camera can help a noisy

---

[11] A program to test the iterative approach was written. See the appendix.

[12] 20 iterations seems to be a reasonable number based on some tests with the example polyhedron.

situation.

Also mentioned above is the fact that the ring solution algorithm returns m points in the workspace, while only three (not in a line) are needed to define the pose of a solid. While actually finding and disambiguating the rings has a higher priority, this redundancy in ring points can help to reduce some uncertainty in the part transformation into the workspace.

One way to evaluate the effects of uncertainty on this algorithm is to run tests with increasingly rounded off input data (alphas). The inputs to the test program (see the appendix) were precise to six significant digits, which is probably more than will be available from the camera processor. This rounding off approach is a good subject for future work.

6.0 Future Work

As mentioned above, implementing ambiguity resolution using beta angle information (see figure 1) is the next logical step for this algorithm. Two different rings with the same structure will map to the camera with different beta angles for their edges. Using this knowledge to resolve pose ambiguities should be straightforward.

The algorithm needs to be tested with a wide range of polyhedra. The simple example used for illustration and for the iteration feasibility tests (see appendix) does not expose the algorithm to any potentially troubling cases.

As mentioned above, it should be very easy to find the input accuracy requirement for this algorithm by inputting successively rounded off alpha angles and seeing what precision is necessary to guarantee convergence. The required accuracy will most likely will be very dependent on the configuration of the polyhedron being viewed.

Most parts to be handled in an industrial setting are better approximated by simple (not neccessarily convex) polyhedra. I imagine that using convex hulls of parts might be a simple approach to adapting this algorithm to the more general case. The camera processor would merely report alpha angles based on the convex hull of the image it observes.

"Robotics" is distinguished from ordinary automation in industry by the key attribute of flexibility. It is within the realm of possibility that a robotic part handler might be called upon to view and handle many types of parts in its routine operations. Adapting this algorithm for a multi-part scenario could be done by simply pre-processing the parts in a common set. This approach extends the algorithm into the realm of part recognition, but this method may suffer from performance problems as the number of parts to be recognized grows.[13] A more efficient approach requires some way to notify the pose processor which part of a set of possible parts is being evaluated. Some gross feature of the part such as complexity[14] might suffice, but some additional feature to allow communication with a parts feeder might be a simple and reliable method.

The reader may observe that this algorithm is merely matching an ordered list of length functions (from the camera image) with a set of other ordered lists (from the preprocessed data), checking them all until all matches are found. This approach may seem to be too long and brutish to some (after all, it is $O(n^4)$ in performance). Pattern matching is a whole field of scientific investigation in itself, and it may be quite possible that there is some "smart" way to achieve the objectives of this algorithm in a much shorter time. A careful evaluation of the requirements of this algorithm in light of modern pattern matching practice may reveal such an efficient approach.

---

[13]On average, doubling the number of parts will double the number of edges (n), increasing processing time by a factor of 16 (because performance is proportional to n^4).

[14]A tetrahedon, for instance, can never show more than four vertices in profile and should always be distinguishable from a higher order solid such as an icosahedron.

After the above areas have been thoroughly investigated, a single-camera implementation with some real parts would be a good demonstration of the RISC approach in action. This robot algorithm module can be classified as a RISC approach because it uses binary images for simplicity and robustness. I expect that it shall prove to be both fast and reliable.

7.0 References

1. Batchelor, Bruce G. and Waltz, Frederick M., *Machine Vision Systems Integration*, SPIE Optical Engineering Press, 1991

2. Canny, John F. and Goldberg, Kenneth Y., "A 'RISC' Paradigm for Industrial Robotics," Technical Report ERSC 93-4, 1993

3. Dhome, M., et. al., "The Inverse Perspective Problem from a Single View for Polyhedra Location," Proceedings IEEE Computer Vision and Pattern Recognition, 1988

3. Jia, Yan-Bin, and Erdmann, Michael, "Sensing Polygonal Poses by Inscription," 1994 IEEE International Conference on Robotics and Automation

4. Klaus, Berthold and Horn, Paul, *Robot Vision*, MIT Press, 1986

5. Preparata, Franco P. and Shamos, Michael Ian, *Computational Geometry, An Introduction*, Springer-Verlag, 1985.

6. Shafer, Steven A., *Shadows and Silhouettes in Computer Vision*, Kluwer Academic Publishers, 1985

7. Wallack, Aaron S. and Canny, John F., "Object Localization Using Crossbeam Sensing," IEEE, 1993

Appendix: Ring Convergence Test Program

To test the feasibility of an iterative approach to solving the nonlinear system of equations, a small program was written that takes values from the example of figure 1. Subroutines were written to test for convergence for the correct ring for the image polygon clocked properly (sub TestRing), for an incorrect ring (sub FalseRing1 uses the base of the part), and for the correct ring clocked wrong (sub FalseRing2). The correct ring clocked properly converged to within 0.1% in 11 iterations.[15] The incorrect cases diverged rapidly. The code for the correct case is shown below. Comments are preceded by an apostrophe.

```
Sub TestRingButton_Click ()

  'Number of edges in this test image:
  m% = 3

  'First guess (distance from camera to center of table:
  Guess! = 9.79796

  ReDim d(m%) As Single              'Ring edges
  ReDim Alpha(m%) As Single          'Cone angles
  ReDim r(m%) As Single              'Right legs the test ring

  'The ring under convergence test:
  d(1) = 3.16228
  d(2) = 2.23607
  d(3) = 3.60555

  'The Alphas passed from the camera processor:
  Alpha(1) = .353917
  Alpha(2) = .246151
  Alpha(3) = .39922

  TestRing d(), Alpha(), r(), m%, Guess!

End Sub

Sub TestRing (d() As Single, Alpha() As Single, r() As Single, ByVal m As
Integer, ByVal Guess As Single)

  ReDim CosAlpha(m) As Single      'To store the cosines
  ReDim AdjFactor(m) As Single     'To store adjustment factors based on alphas

  'The correct answer for this run:
  'r(1) = 9
  'r(2) = 9
  'r(3) = 9.16515

  'Set the desired precision:
  ErrorBound! = .001        'One part in 1,000 for the ring segments

  'Set the maximum loops before bailing out:
  MaxLoops% = 20            'If it doesn't converge in 20, it never will
```

---

[15] Within 0.01% (one part in 10,000) in 24 iterations.

```
    For i% = 1 To m%
      'First guess (distance from camera to center of table:
      r(i%) = Guess
      'A little preprocessing to speed things up:
      CosAlpha(i%) = Cos(Alpha(i%))
      AdjFactor(i%) = (Cos(Alpha(i%)) / Sin(Alpha(i%))) / 2
    Next i%

    'Keep track of time for performance evaluation:
    Start! = Timer

    Do
      MaxError! = 0
      For i% = 1 To m%
        dCalc! = Sqr(r((i% Mod m%) + 1) ^ 2 + r(i%) ^ 2 - 2 * r((i% Mod m%) + 1) *
r(i%) * CosAlpha(i%))
        dError! = d(i%) - dCalc!
        ErrorFraction! = Abs(dError! / d(i%))
        If ErrorFraction! > MaxError! Then MaxError! = ErrorFraction!
        Adjust! = dError! * AdjFactor(i%)
        r(i%) = r(i%) + Adjust!
        r((i% Mod m%) + 1) = r((i% Mod m%) + 1) + Adjust!
      Next i%
      j% = j% + 1
      If MaxError! < ErrorBound! Then Exit Do
      If j% > MaxLoops% Then Exit Do
    Loop

    Finish! = Timer
    ElTime! = Finish! - Start!

    Print
    Print " Time: "; ElTime!
    Print " Loops: "; j%
    Print r(1); r(2); r(3)

End Sub
```

A sample output when the "Test Ring" button is pushed is shown below. The "Test Ring" button triggers the TestRingButton_click event procedure which calls the TestRing subroutine. The display shows an elapsed time of zero. Actually, there was some finite time used, but the amount is below the resolution of the display number format. The iterative solution converged to within part-per-thousand precision (in the measured parameter, edge length based on alpha) in 11 loops. The calculated cone leg lengths are also shown. An actual implementation would use these leg length numbers to find the locations in the workspace of the ring vertices.