Robotics and Artificial Intelligence

Edited by

Michael Brady

Senior Research Scientist, Artificial Intelligence Laboratory, Massachusetts Institute of Technology,
Cambridge Massachusetts

Lester A. Gerhardt

Chairman and Professor, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute
Troy, New York

Harold F. Davidson

Consultant, Department of the Army
Washington, D.C.

Springer-Verlag Berlin Heidelberg New York Tokyo 1984
Published in cooperation with NATO Scientific Affairs Division

CAD, Robot Programming and Ada[1]

by

Richard A. Volz[2]
Anthony C. Woo
Jan D. Wolter

Trevor N. Mudge
Jerry L. Turney
David A. Gal

1. Introduction

This paper addresses two topics which on the surface are unrelated, the use of CAD to assist robot and sensor programming, and the use of Ada as the basis for robot programming. The association between them arises from the fact that they are being combined in an experimental facility. The facility consists of an Intel iAPX 432 multiprocessing microcomputer system, a GE TN2500 camera, an ASEA RB 6 robot and a link to a VAX 11/780 off-line computer system. The facility is being used as a testbed for various robot programming and interface strategies, and to investigate the utility of object based systems as the computer foundation of manufacturing cells. Experimental verification of techniques using information extracted from CAD models to assist in robot programming and the use of Ada are important parts of the experiment.

1.1. The Use of CAD to Assist Robot & Sensor Programming

Computer aided design is rapidly becoming an integral part of design in many fields of engineering. The geometries contained in CAD descriptions can also be of use in several ways to assist the programming of robot and sensor systems and reduce or eliminate their on-line manual training thus increasing productivity. Two specific uses of CAD are considered here, the automatic determination of gripping positions and the use of model generated templates for an optimal template matching scheme for machine vision.

Grip position determination requires consideration of several basic issues: definition of what constitutes a good grip; specification of the geometrics involved; available knowledge of the operating procedures and environment; computational efficiency. Typical conditions for a "good grip" include *feasibility*, the reachability [1], [2] of the grip position by the robot without interference [3] between the robot and the object to be gripped, *stability*, [*1*], [*4*] the nonmovability of the object being grasped under external forces, particularly gravity, and the *area*[5]-[7] of contact between the gripper and the object surface. Paul [1] has suggested that the center of mass of the object being grasped lie on the axis between the fingers during the grip to minimize the torques applied to the part around the axis of the grip. Asada [5] has given a different definition for stability: a grip is stable if when a small relative displacement occurs between the part and the hand a restoring force is generated to bring the part back to its original situation. Asada has also suggested that a good heuristic for maintaining stability is that the perpendicular projection of the center of mass be near the intersection of gripper and the object [5]. Mason [8] has also suggested uncertainty reduction as an important criterion.

The determination of grasp positions cannot be accomplished based upon geometric considerations of the object alone. Grasp positions depend upon other objects in the vicinity which might interfere with a particular grasp point as well. Several authors have developed

[1] This work is supported in part by the Air Force Office of Scientific under contract number F49620-82-C-0089 and in part by the Zimmer Foundation.

[2] The authors are with the Robot Systems Division of the Center for Robotics and Integrated Manufacturing, College of Engineering, University of Michigan.

techniques for taking into account interfering objects in the vicinity of the initial or final object position [3], [4], [9]-[11].

This paper addresses both the question of what constitutes a good grip and the computational issue. In particular, the stability of a grip is viewed from two perspectives, slippage and twisting of the object in the gripper due to external forces and torques. The latter can also be heuristically related to uncertainty of positions during the grasping operation itself. Resilience to slippage is expressed in terms of friction effects of the surfaces involved, the shape of the contact between the gripper and the object and the usual distance of the grasp point from the center of mass of the object. A new performance measure is introduced to reflect resilience to twisting. In both cases it is found that area is not the critical parameter reflecting the quality of a proposed grip. A general grasp planning strategy is introduced which, is both run-time efficient and general in that it can be used in a broad range of applications. The problem is divided into three parts, one which depends only upon the geometry of the object to be grasped, one which takes into account a priori geometric constraints, and one which handles constraints unknown until run time.

The second application of CAD derived information is in the development of visual recognition algorithms for objects whose boundaries cannot be completely determined. Template matching of the boundary of the part with a model of the boundary has been suggested for dealing with this problem [12]: if a match with a large portion of the template is made in the image, it is concluded that the part has been found. Unfortunately, in its basic form, this procedure requires an excessive amount of computation and can produce a number of false matches resulting in unreliable recognition. Variations of template matching have been proposed, such as the generalized Hough transforms of Sklansky [13] and Ballard [14]. Ballard's approach does not permit an edge image point to be considered a match unless it agrees in angle of slope as well as in edge point. While this criterion places a local restriction on the matching condition, it is not restrictive enough. A typical correct match still produces a considerable number of large false peaks in the accumulator array recording the template edge matches. The approach presented here optimally weights the edges of the template to emphasize segments of the object boundary distinct from segments of boundaries of other objects which might be present. This greatly reduces the size of false peaks in a large number of cases.

1.2. Ada as the Basis for a Robot Programming Language

With the advent of robot-based manufacturing cells, the need for a standard implementation language to program these cells has grown in importance. The present practice of designing new robot languages for nearly every new robot may satisfy the particular programming needs of each robot, but it is counterproductive from the standpoint of developing integrated manufacturing cell technology. Standardization is clearly needed. Moreover, the current high level languages used to implement the real-time requirements of manufacturing systems lack some of the language tools, such as data abstraction, that facilitate programming in the large. Even the most sophisticated robot, numerically controlled (NC) tool, and related "manufacturing systems" programming languages presently in commercial use support neither data abstraction nor other features appropriate for large scale programming [15], [16]. They are, therefore, unsuitable at the cell integration level. However, the Department of Defense's (DoD's) future system implementation language, Ada[3], is an attempt to provide language constructs which can overcome most of these shortcomings.

Ada was originally developed at the instigation of the DoD [17] for programming embedded systems. Ada is based on Pascal. However, significant extensions make it the first

[3]Ada is a Registered Trademark of the Ada Joint Program Office--DoD.

practical language to bring together important features that include data abstraction, separate compilation, multitasking, exception handling, encapsulation, and program abstraction through generics and operator overloading. These extensions make Ada particularly appealing for programming large scale real-time embedded systems--a situation characteristic of robot-based manufacturing cells. Most importantly DoD's strong support of the language guarantees a large scale presence in the future. This, therefore, warrants a serious inquiry into the feasibility of using Ada as a standard implementation language for manufacturing cells.

2. Application of CAD to Robot and Sensor Programming

2.1. Automatic Determination of Gripping Positions

The approach being taken to determine gripping positions for an object is characterized by the following:

- Extraction of geometric information about the part from a Computer Aided Design (CAD) system.
- Use of extensive off-line preprocessing to generate a grip-list which can be used to select a grip at with minimal run-time computation.
- Development of quantifiable criteria for a good grip (which can be related to grip stability and configuration uncertainty) and use of the criteria to judge the relative quality of gripping positions.

Several necessary conditions and evaluation criteria of a good grip are identified below and algorithms for evaluating a possible grip according to these described. The general strategy for determining a set of potentially good grips is to use some of the necessary conditions as filters which eliminate infeasible geometries from consideration. Next a set of potential grips is generated and evaluated according to the remaining criteria. The grips are then ordered according to grip criteria. In a particular situation in which a grip is needed, the highest ranking grip which does not cause interference with the initial or final surroundings is chosen. The use of the grip list forms a basis for shifting computation between on line and off line according to available a priori knowledge of the system, thus minimizing the amount of on line computation.

2.1.1. Part and Gripper Geometries

A prerequisite for finding gripping positions is the availability of detailed geometric information about the part, the robot gripper and the surroundings. Part information is assumed to be polyhedral and is derived from a description generated on a Computer Aided Design system. This information takes the form of a topological and geometrical description of the surfaces of the object in terms of faces, vertices, and edges.

Possibly usable surface elements include faces, groups of edges, or vertices. Grips on sets of vertices only are discarded since such grips are highly subject to rotation when even small torques are applied. Similarly, grips on combinations of edges and vertices are discarded, as they are highly subject to twisting during a grip. Grips on a pair of faces, a face and a vertex, a face and an edge, or on pairs of coplanar edges may possibly yield good grips. For simplicity, however, only grips on pairs of faces are considered initially. Extension to the other cases is not expected to be difficult. In general, both outside and inside grips are possible. The algorithms developed initially are for outside grips only, but can be easily extended to handle inside grips as well.

A typical parallel gripper is considered. The "arm" is assumed to extend to infinity in a direction extending backwards from the gripper. This model is used for evaluation of intersections between the hand (and arm) and the part. Further, it is assumed that the gripper applies a constant force to the gripped object.

2.1.2. Off-Line vs. On-Line Processing

Grip position determination cannot be performed independent of the environment in which the part is to be found. Interference calculations must be performed to assure that a proposed grip causes neither interference between the gripper and the part nor interference between the gripper and any other objects in the vicinity of the initial and final positions. In a completely unstructured environment, all of the calculations for determining gripping positions must take place on-line at run time. The manufacturing environment, however, is seldom completely unstructured, and one can use whatever knowledge is available about the structure of the environment to allow some of the computations to be performed off-line to improve run time efficiency.

If one knows at least what parts are to be present, then one can determine off-line a set of feasible grip positions for each part and rate them in terms of their grippability (see below). Then at run time the top ranked grip position which does not cause interference can be selected. If one knows a priori that the part will be found on a flat table (albeit one that may be cluttered with other parts) one can find a set of feasible grip positions for each of the stable positions [18] which will not interfere with the known table location. This will be a smaller list than the complete list generated for the part as a whole. If one further knows that there are no other objects near either the initial or final position of the part, then a specific grip point can be selected off-line.

The approach taken then, is to generate a list of possible grip points for each stable position and rank each of the lists by quality of the grip (as measured by resilience to slippage and twisting, as indicated below).

2.1.3. Gripping Criteria

The key to operation of the grip determination algorithms are the criteria used to evaluate potential grips and the algorithms for performing the evaluation. Two obvious and familiar necessary conditions and three evaluation criteria are used.

Necessary Conditions

- The faces to be gripped must be nearly parallel, facing away from each other, separated by less than the opening of the gripper, and opposite each other.
- There must be no interference between the hand, the object to be grasped, the initial surrounding, or the final surroundings.

Evaluation Criteria

- A measure of the susceptibility of the grip to slippage of the object.
- A measure of the twistability of the object during gripping.
- A measure of the gravitational torque which might be applied to the grip.

The following paragraphs discuss the algorithms for each of these briefly.

The first necessary condition is easily enforced by requiring that the inner product of the outer normals of the faces under consideration be within epsilon of -1, and that the projection of one of the faces on the other not be null. The separation condition can be directly checked.

The obstacle evaluation process can be simplified by reducing the 3D intersection problem to a 2D intersection problem for each potential grasp point. The portion of the part or nearby objects which lies between the top plane of the finger (in the open position) and the top of the potential gripping surface is projected onto a gripping plan parallel to the fingers of the gripper. The result is a 2D figure representing the portion of the part blocking the top finger. (See Figure 1.) A similar projection is used to determine interference with bottom finger and the base of the hand. On the surface, the idea of using cutting planes corresponding to the hand base and fingers appears the same as that of Laugier and Pertin [3], but the algorithm used here to find the obstacles is completely different and more efficient as the obstacle checking does not involve an iterative procedure as does theirs. Each potential grip is partially specified by an approach vector to the object. To check for interference, the hand is simply slid toward the part along the approach vector until the projection of its fingers and hand on the grip surface touches the projected obstacles. This is a 2D test which can be completed quickly. The touch point defines the distance along the approach vector at which the grip should actually be made. Other interesting parameters of the grip (contact dimensions) can then be determined from the projection of the fingers on the gripping surface.

Two basic types of slippage (which is related to the stability of the grip) are considered, translational slippage and rotational slippage. The translational force required for slippage to occur is dependent upon the normal force and the coefficient of friction. The former is assumed constant and the latter is a function only of the materials involved. As neither of these is a function of the grip, translational slippage is not a factor in determining gripping positions.

The torque required to rotate a part between the fingers of the gripper depends upon the coefficient of static friction and the shape of the surface gripped. For example, if the surface of contact is a disk, the torque required for slippage is $\tau = (2/3)\mu NR$ where μ is the coefficient of friction, N is the normal force and R is the radius of the disk. (See [6] for more detail) It is worth noting that torque required for slippage is in some way proportional to the shape of the surface contact, not the area (as has been used in several other grip determination schemes). For a good grip one would like to contact dimensions to be large.

The gravitational torque applied which will tend to cause slippage, on the other hand, should be as small as possible. This torque depends upon the distance of the grip from the

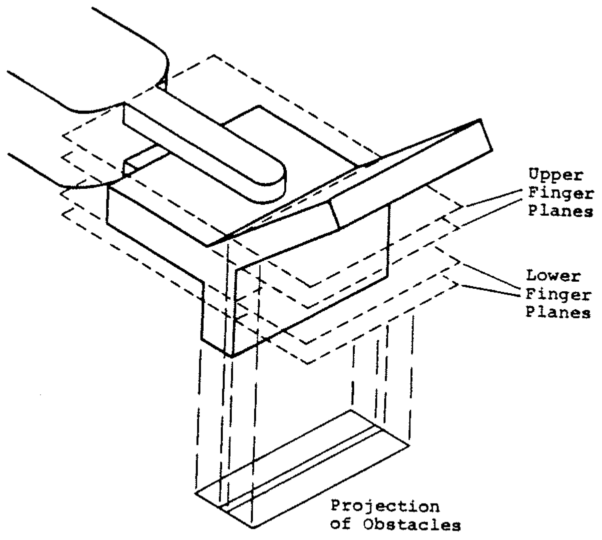


Figure 1. The portion of the object between planes P_1 and P_2 and between P_3 and P_4 are projected onto the gripping plane P_g.

centroid and the orientation of the part and gripper. Since the hand (and hence the part) will move through a variety of orientations during a move performed after the grip, the straight-line distance of the grip from the centroid is used as a worst case measure of gravitational torque. Torques applied as a consequence of the dynamic motion of the hand with the grasped object cannot be determined without knowledge of the planned trajectory of motion. They will also, however, in worst case orientation, depend upon the distance of the grip position from the centroid of the object.

The final criteria used is the susceptibility of the proposed grip position to the occurrence of twisting during the grip. If one attempts to grasp the object along the long narrow pair of faces of the T-bar in Figure 2, it is quite likely that the object will twist in the fingers of the gripper, particularly if a pneumatic type of grip actuation is used. Gripping at the T end of the bar would be likely to be more stable, even though it may have smaller area that the narrow face. The relevant measure in this case is the distance by which the fingers must be separated for the part to twist out of position. To calculate this measure, the convex hulls of the 2D contact areas of each finger with the surface of the part are determined. For each edge of one convex hull, the most distant vertex on the other is found. This is how far the hand would have to be open to tip over that edge. (See Figure 3). The minimum of these distances is found and the normal finger separation subtracted. The result is the distance the hand would have to open to allow the part to twist into another position.

A linear combination of the three measures given above is used to rate the potential grip points. The method has been fully implemented in PASCAL on VAX 11/780. A typical part with 46 faces, 132 edges and 86 vertices is handled in 35 cpu seconds. The complexity is of order N^2 where N is the number of faces.

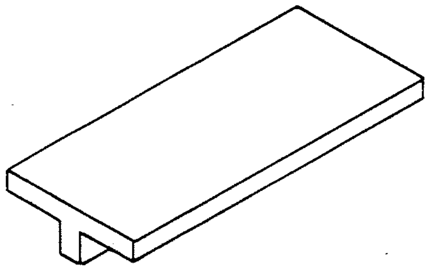


Figure 2. Grasping along the narrow pairs of faces is not as stable as grasping the T end.

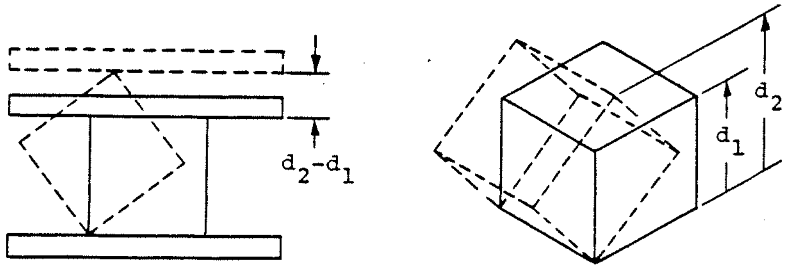


Figure 3. 3(a) shows an object rotated along one edge as would occur during twisting. 3(b) shows an end view of the same object with the gripper included. The difference d_2-d_1 is the gripper displacement needed to allow twisting.

2.2. Occluded Part Recognition

In a typical industrial scene, parts may be intermixed, partially occluded, and of unknown pose (position and orientation); however, the types of parts that will be present are almost always known *a priori*. From the part geometries in the database, boundary templates for every stable position of every part are determined, and a set of subtemplates which most differentiate the parts are identified. We refer to these as *salient* features. To illustrate consider Figure 4. Figure 4(a) shows the salient features when parts A and B comprise the set of parts, while Figure 4(b) shows the salient features when parts A and C comprise the set of parts (salient features are shown as heavy lines). Saliency is formally defined and given a continuous range. Then the pose of a part that is partially occluded can be accurately determined if the visible portion of the part matches some set of subtemplates with enough combined saliency.

2.2.1. Template Matching and the Hough Transform

The scene in which the parts appear will be referred to as the application scene. It is digitized into a two-dimensional array of pixels, I ($1 \le x \le M$, $1 \le y \le N$: x and y integer), which can take on values from a set of gray-levels. The boundary representation of the parts in the application scene is called the boundary image. Although its machine representation is usually a compact form, such as a linked list, the boundary image can be thought of as a function, $B(x,y)$, which is defined on the same domain as I and which has the value one at boundary pixels and zero elsewhere. It is the boundary image against which the templates are matched to locate the part in the application scene. A template, $T(x,y)$, can also be thought of as a binary valued function defined similarly to B.

Associated with each pixel in B is an accumulator whose purpose is to record template matches. This two-dimensional array of accumulators is assumed to be initially zero. A reference point is selected that is fixed with respect to T; we used T's centroid, but any appropriate reference point could be used. A template, $T^{*}(x,y)$, is formed by rotating T 180^{0} about

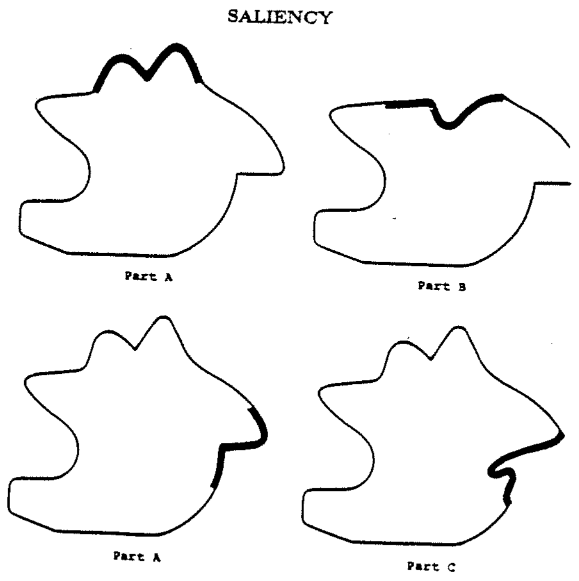


Figure 4 Salient Features

its centroid. The centroid of T^* is placed at a boundary pixel in B and every accumulator that coincides with a boundary pixel of T^* is incremented by one. This procedure is repeated for each boundary pixel in B. When all the boundary pixels have been visited, the accumulator with the largest value identifies the position of the centroid of template T that gives the best match.

As demonstrated by Sklansky in [13], the function produced in the accumulator array by the above procedure is the convolution of B with T^* which is simply the cross-correlation or template matching of T with B. Two points arise from this observation. First, the Hough transform [13] performs the function of a matched filter and therefore is the optimum filter when B is corrupted by additive white Gaussian noise under a wide variety of criteria. However, none of these criteria implies that it is optimal for recognizing one shape in the presence of others (see the discussion on normalized correlation in Rosenfeld and Kak [19]). Second, the function produced in the accumulator array by the Hough transform is the same as that produced by template matching. However, the order of operations are different.

Although the Hough transform is an efficient way to perform template matching, it nevertheless has many of the shortcomings associated with template matching, particularly in its ability to deal with variations in orientation and scale. For example, to account for variations in orientation the complete procedure (template matching or Hough transform) must be repeated for every orientation to be distinguished. In other words, if it is required to distinguish between orientations that are 1^0 apart, the procedure must be repeated 360 times, resulting in 360 accumulator arrays. The best match would then be identified by the accumulator with the largest value in all of the 360 arrays. A similar complexity arises if the scale of the parts are not known in advance—all sizes of T to be distinguished must be tried.

The next section presents a technique based on an extension of the Hough transform which uses subtemplates. In addition to improving the Hough transform by introducing more local restrictions that decrease the likelihood of false matches, this technique reduces the complexity that arises from the need to consider variations in orientation. The problem of variation in scale is not addressed in detail in this paper.

2.2.2. Subtemplate matching

A set of $|\tau|$ overlapping subtemplates, τ_i, are created from template T. Each subtemplate, τ_i, has an associated vector, $\mathbf{v}_i$, that points to the location of the centroid of T. The Hough transform can now be modified by requiring that a segment of the boundary image centered on a particular pixel, b_j, match τ_i before $\mathbf{v}_i$ can be originated at b_j. The degree to which τ_i matches the boundary segment is reflected by allowing the accumulator pointed to by $\mathbf{v}_i$ to be incremented by a fractional value (see below). The restriction of matching subtemplates significantly reduces the subset of vectors that originate from b_j. In our experiments each τ_i was 20 pixels in length and $|T|/2$ subtemplates were created for each template T where $|T|$ is the cardinality of pixels in T, i.e., each template pixel was part of 10 subtemplates. The choice of the length and the number of subtemplates is application dependent.

In the process of matching the subtemplates with the boundary it is important to place the subtemplate in the correct orientation. This can be achieved efficiently if the subtemplate and the boundary image segments are represented in their *intrinsic* coordinate systems in which the angle of slope, ϑ, and the arc length along the boundary, s, act as coordinates for pixels [20]. This $\vartheta-s$ representation allows subtemplates and boundary image segments to be characterized by functions of the form $\vartheta(s)$. An important property of these intrinsic functions is that a change in orientation, ϑ_c, of a boundary in $x-y$ space corresponds to simply adding ϑ_c to $\vartheta(s)$ in $\vartheta-s$ space.

the weighted subtemplate will respond poorly to the boundaries of other objects.

2.2.4. Results

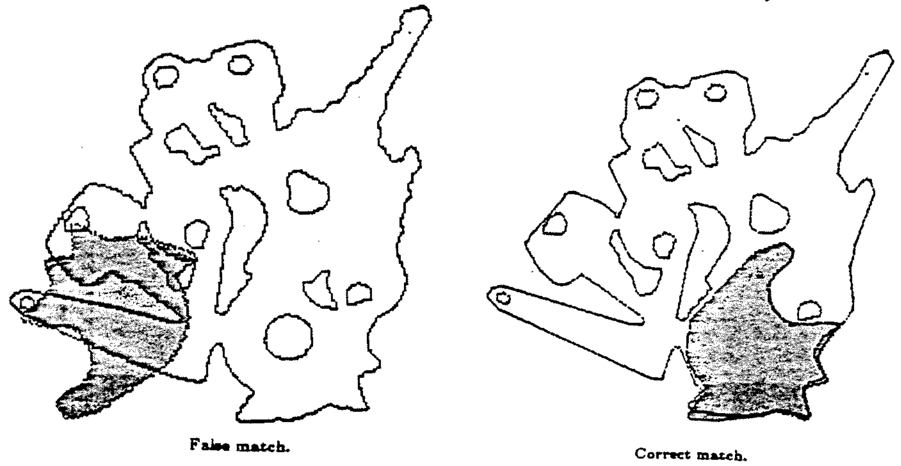
Figure 5 shows the results of applying this algorithm to a scene in which objects are overlapping. Parts are found without the need of rotating a template since the subtemplate matches are matched in a rotationally invariant space. The code is written is the language "C" and runs on the VAX 11/780. The average time to recognize an object is 45 CPU seconds. However, no effort has been made as yet to optimize the code.

3. Ada as a Basis for a Robot Programming Language

As indicated in the introduction two major concerns in the development of future robot programming systems are the complexity of the software to be developed and standardization. Ada was expressly designed to facilitate the development and maintenance of large software systems through partitioning and by DoD decree will to some extend be standard. Separate compilation while retaining type checking of parameters passed between modules allows a team of cell designers to work concurrently on the development of separate subsystems. It also allows subsystems to be easily modified without affecting the rest of the system--an important feature for maintaining the system. Ada relies on data and program abstraction to simplify the construction of subsystem interfaces. Most importantly for the current topic they allow the *language* to be tailored to a specific application area. Further, Ada provides multitasking and timing constructs, an essential ingredient in manufacturing cells where there are typically several computation tasks that need to be performed in realtime.

3.1. Features of Ada

The underlying philosophy of Ada is centered upon the use of *objects* for program design. An *object* is a data structure[4] having a unique identifier and an associated set of functions and procedures that can operate on it [23]. These "operators" are the only allowed means of



False match.

[Slope restricted Hough.]

Correct match.

[Subtemplates with salient weightings.]

Figure 5. (a) shows a false notch which occurred using the generalized Hough Transform of Ballard. (b) shows the correct notch determined by our method.

[4]The meaning of the term "object" is not universally agreed upon; our usage is fairly narrow. See [22] for a discussion of various viewpoints.

Matching starts by choosing a subtemplate, $\vartheta_{\tau_i}(s)$, from the template boundary and attempting to match it against an equal length segment of the boundary image, $\vartheta_{\beta_j}(s)$. β_j represents a boundary segment centered at pixel b_j. To place the subtemplate in the correct orientation before matching the average angles of orientation, $\overline{\vartheta}_{\tau_i}$, and $\overline{\vartheta}_{B_j}$ for the subtemplate τ_i and the boundary image segment β_j are determined by averaging the slope angles of the points in the subtemplate and boundary image segment respectively. The difference between the average slope angles, $\Delta\overline{\vartheta}_{ij} = \overline{\vartheta}_{\beta_j} - \overline{\vartheta}_{\tau_i}$ is a measure of the difference in orientation between the boundary image segment and the subtemplate. The subtemplate is then shifted $\Delta\overline{\vartheta}_{ij}$ to the same average angle as the boundary image segment before the two are compared. The shift in ϑ corresponds to an angular rotation of the subtemplate to the average orientation of the boundary image segment.

A measure of the mismatch between the rotated subtemplate, τ_i, and the boundary image segment, β_j, is given by the quantity, γ_{ij}, such that:

$$\gamma_{ij} = \sum_{p=1}^{h} [\vartheta_{\beta_j}(s_p) - \vartheta_{\tau_i}(s_p)]^2$$

where h is the number of pixels in τ_i and β_j. Using γ_{ij} a measure of the match between a subtemplate and a boundary image segment is given by the *matching* coefficient, c_{ij}, where:

$$c_{ij} = \frac{1}{1+\gamma_{ij}}$$

c_{ij} is used to adjust the contribution added to the accumulator. When a match occurs, the better the match, the larger c_{ij} and, therefore, the larger the contribution.

In order to locate the potential center for the part, the vector, bvi, associated with the subtemplate, τ_i, is rotated by the angle, $\Delta\overline{\vartheta}^{ij}$, and is attached to the edge image section at its center point. . Assume the cell pointed to by the rotated bvi, is in column, $u(i,j)$, row, $v(i,j)$. The value, $c_{ij}w_i$, is added to the accumulator at location $(u(i,j), v(i,j))$ where w_i is the weight associated with subtemplate τ_i. The angular difference, $\Delta\overline{\vartheta}^{ij}$, is stored in a linked list associated with the same cell as the accumulator, this angular information allows the orientation of the part to be determined by averaging the angles stored in this list if the cell is determined to be the template center.

2.2.3. Determining the Saliency of Subtemplates

Each template from the database is matched, via the subtemplate matching scheme described above, with every other template in the database of possible objects and stable positions. This results in sets of accumulator values which depend upon the weights of the subtemplates. These accumulator values are combined into a positive definite form quadratic in the weights. The weights are then constrained to lie in the region [0,1] and to sum to unity. The resultant constrained quadratic optimization problem can be solved using the "Nonnegative Least Squares" algorithm described in Lawson and Hanson [21].

The subtemplate weights computed are based on the dissimilarity of the subtemplates to sections of the boundary of the other objects in the database. Those subtemplates of the boundary which are dissimilar to other sections of other boundaries are weighted more heavily. A group of heavily weighted subtemplates makes a larger contribution to the accumulator array and therefore plays a more significant role in determining the potential location of the object when the boundary is fragmented. Weighting the subtemplates can also be viewed as a method of decorrelating the complete template of the object with the boundaries of other objects. Since more weight is given to those subtemplates which fail to match sections of the boundaries of other objects, the matching response of a template composed of manipulating the object. A number of advantages follow from this "object-based" programming methodology. Objects and their associated functions and procedures form natural boundaries along which to subdivide systems. In addition, because the structure of a data type is hidden from all but its associated operators, changes to the structure have a limited impact greatly simplifying program modification and maintenance. In effect, the data type can be abstracted and known only through operations on it. Thus, object-based programming provides a way to implement data abstraction, which, as a result of work in programming language design during the 1970's, has emerged as a major organizational concept in programming languages [24]. Several experimental programming languages have been implemented that were designed containing features to support abstract data types, but, with the possible exception of Modula-2 [25] and Concurrent Pascal [26], Ada is the first that is likely to see wide-spread use.

Ada provides a construct called a *package* that allows the programmer to encapsulate objects and their associated functions and procedures. In addition, it has *private* types and *limited private* types that further restrict encapsulation so that objects thus typed, while visible to program parts, can only be manipulated through procedure and function references. Together these features permit the programmer to hide data structure implementation and create abstract data types. The package definition consists of two parts, a *specification* part and a *body*. The specification part introduces the data types, variables and procedures visible to the user of the package. The body contains the implementation of the package and may be accessed only by the mechanism stated in the package specification.

Program abstraction is possible through the use of *generic* packages and subprograms, and operator overloading. These are a step in the direction of polymorphic function implementation [27], [28], and they allow, among other things, operations to be defined over a set of data types, thus providing a broader use of objects. As shall be discussed further, the object and package concepts address the management and portability of complex software.

Ada also provides a *task* construct which is a means of dividing a program into logically concurrent operations with possible synchronization between them [29]. In addition to forming the basis for real-time operations, tasks also provide a means of increasing processing efficiency in a parallel processor environment. Syntactically tasks bear a resemblance to packages in the sense that they both have a specification part and a body. However, the specification part of a task is used solely to declare the synchronization points or *entry* points to the task--the entry points indicate where messages are received/transmitted by a task.

3.2. A Strategy for Using Ada

The use of Ada for programming manufacturing cells begins with the definition of *objects* for the various physical and logical components in the cell and the interfaces to these objects. Among these are the problem oriented primatives one would like to have in a robot language. These objects are embedded in Ada packages. Various mechanisms can be easily implemented to (nearly) automatically make these objects available to the programmer. The robot programmer can then use these objects and interfaces as though they were part of the language specification.

To provide a concrete illustration of the use of Ada as the system implementation language for cell programming, a system consisting of a robot, a vision sensor and a link to a CAD database is considered in a simplified manner. Four basic object types are defined in this illustration:

• ROBOT	--	Provides the basic robot interface.
• POSITION	--	Provides a set of data abstractions

for part and robot locations.

- CAD_MODEL -- Provides CAD database access.
- VISION -- Provides an interface to the vision subsystem.

Each of these basic objects is associated with an Ada package. A view of a portion of the main programs and the specification for each of these basic objects gives an introduction to the object oriented design approach.

Figure 6 shows a portion of an example main program. The action part of the example shown is to find a part on the input conveyor, move the robot to it and pick it up. It is assumed that the set of parts that could potentially appear on the conveyor are assigned to the variable SET_OF_PARTS. The names of the parts are assigned from a terminal via a command interpreter (not shown). Details such as following a particular speed profile or the handling of exceptions are omitted since they would tend to obscure the example. It is assumed that the geometry of the part is available in a CAD database and that off-line utilities using the CAD information are available to provide recognition information to the vision system and the location on the part where it can be picked up (the grasp points are defined in a local coordinate system of the part itself).

The first part of this example identifies Ada packages which provide data types and services to the main program. The *with* and *use* clauses are the mechanism by which the robot environment is made available. (In the program parts shown words in lower case bold are Ada key words. The upper case words are user-defined, or predefined package, function, procedure, type or variable names.) The *with* clause tells the compiler that the programmer intends to use data types, procedures, and functions defined in the package named after the *with*. *The use* clause tells the compiler that the programmer wishes to reference the data types, procedures and functions defined in the package named after the *use* by the names given in the package definition without including the name of the package as a qualifier. In general, however, the user might not even have to enter these *with* and *use* clauses directly. The *use* and *with* clauses could be placed in a program template with which the user begins. Alternatively an *include pragmaa*[5] could be added to the compiler which would read a file of *with* and *use* clauses and include them in the program. In this way an environment of data types, and primitive operations tailored to a specific application in this case robots, can be provided to the user.

The second half of the main program shows use of the data types and functions provided by the Ada packages for the simple operation described above. The syntax used is similar to that found in several robot languages and the type and variable names are sufficiently mnemonic that one can follow the intent of the program with minimal reference to the supporting packages (see below). Note that comments are introduced by a preceding "--" and they can be placed anywhere in the text stream. In addition, the comments in Figure 6 include one or two numbers in parentheses that are the figure numbers of relevant packages.

FIND is a procedure in the VISION package that finds and identifies a part on an input conveyor and returns the part's name, a 4×4 homogeneous transformation giving the location of a coordinate frame for the part in terms of the robot's world coordinates and an index of the stable position in which the part was found. These three items of data are stored as components of a record X.

PICK_APP_POINT and PICK_POINT are functions which return (from the CAD database or utilities acting upon it) 4×4 homogeneous transformations which express the approach and grasp points in terms of the coordinate frame for the part. The "*" has been overloaded to mean multiplication of 4×4 matrices so that the result is the transformation of the

[5]A pragma is simply a compiler directive.

appropriate point in terms of the world coordinates of the robot. TARGET_LOC holds this transformation and is the argument of the MOVE function which actually causes robot motion.

Partial specifications for the packages referenced in Figure 6 are given in Figures 7 through 10.

3.3. Discussion of Ada Usage

From the viewpoints of managing complex software, providing an application specific programming environment to the user, and achieving language standardization, Ada provides a number of advantages. These include:

- The use of data abstraction and operator overloading to create well modularized application specific code helps usability, readability and maintainability.
- The resulting application package can create a reasonable application specific environment.
- The strong type checking significantly aids debugging.
- The separate compilation features in conjunction with the other features above aids flexibility and helps portability.
- The expressive power of the language is excellent.

Having these capabilities widely available in a standardized language is very significant as this can greatly aid in standardizing application specific "languages" and giving them portability. The portability can be inherited, to a large measure, from Ada.

There are also a number of concerns which have arisen which either are a detraction to some users or bear further investigation:

- The heavy use of data abstractions creates additional procedure calls and corresponding overhead which can cause difficulty in a real-time environment.
- Strong typing can get in the way of what one wants to do.
- How usable will Ada really be, even with good environment creation through special packages, to the noncomputer professional?
- The debugging of robot programs requires close interaction with the programmer. It is not clear this can happen with Ada.

In theory, the inefficiencies produced by excessive subroutine calls can be reduced with the *inline pragma* (a compiler directive which expands the subroutine source code in-line wherever called). Our early experiments with *inline* supports this theory. Drastic reduction in computation time has accrued through its use. The strong typing argument has raged for some time and is not specific to robot or manufacturing cell applications. We believe that as the size and complexity of a software project increase so does the importance of using strong typing.

We do not ever expect to see robots on manufacturing cells programmed in Ada by shop floor personnel. We expect that as more complex arrangements of machines are built and as better links with CAE/CAD are forged, shop floor personnel will cease to "program" robots. Rather they will interact with a program to identify what is to be done next. The actual programming will be done in a more generic fashion by a person who has a good mix of manufacturing and computer engineering/science in his/her background. A person with this type of training should be able to deal with a "roboticized Ada".

The debugging issue is one that requires considerable additional research. All Ada implementations in progress are based on a compile translation while almost all robot programming languages are based on interpretive translation. From the point of view of the programmer, however, the robot program may be a separately prepared and debugged entity. What is really necessary is a fast interactive translate/debug system. This does not preclude compile translation, particularly if used in conjunction with a simulator [30].

Recent programming language research has yielded a number of new concepts which will aid the program development process. A number of these are incorporated into Ada. Future languages will undoubtably encompass more of these concepts. However, at present, the considerable resources being put into the Ada effort by the DoD coupled with its orientation toward real-time embedded systems makes us believe it will be a significant factor in the future.

4. Summary

The sections above describe three aspects of work being incorporated into an experimental robot/vision/CAD system. At the present time the distributed control of the robot, a preliminary vision system based on standard SRI [31] techniques (however, trained from CAD generated data), and grip position determination are all complete and individually working. Integration into the complete system is expected to be accomplished by January of 1984. The occluded part recognition system is working standalone on a VAX 11/780. It has recently been discovered that the algorithms are amenable to parallel computation. This will be explored in the future as the occluded part recognition system in incorporated.

```
with POSITION; use POSITION;           -- Make the procedures, functions
with CAD_MODEL; use CAD_MODEL;         -- and data types defined in the named
with VISION; use VISION;               -- packages available to create a robot
with ROBOT; use ROBOT;                 -- environment for the programmer.
procedure MAIN is
      •
      •   N: INTEGER;                        -- Number in set of parts.
                                       -- Input from terminal.
   SET_OF_PARTS: PART_SET (1..N);      -- Set of parts that could potentially
                                       -- appear on the conveyor.
                                       -- Input from terminal (9).
   X: PART;                            -- Data about the part found (10).
   TARGET_LOC, PICK: FRAME;            -- Coordinate frames for the part
                                       -- and its grasp point (7).
      •
      •
begin    CALIBRATE;                        -- Calibrate the robot before starting (8).
      •
      •
   SET_SPEED (FAST);                   -- Set robot speed fast for motion
                                       -- to approach point (8).
   X:= FIND (DECISION_TREE (SET_OF_PARTS));
                                       -- Find and identify the part (9,10).
   PICK:= PICK_APP_POINT (X.NAME, X.STABLE_POS);
                                       -- Approach point from CAD d/base (9).
   TARGET_LOC:= X.LOCATION * PICK;     -- Express approach point
                                       -- in world coordinates (7).
   MOVE (TARGET_LOC);                  -- Move to approach point (8).
   SET_SPEED (SLOW);                   -- Set robot speed slow for final
                                       -- motion to grasp point (8).
   PICK:= PICK_POINT (X. NAME, X.STABLE_POS);
                                       -- Get grasp point from CAD database (9).
   TARGET_LOC:= X. LOCATION * PICK;    -- Put in world coordinates (7).
   MOVE (TARGET_LOC);                  -- Move to grasp point (8).
   CLOSE_GRIP;                         -- Grasp part (8).
end MAIN;
```

Figure 6. Outline of the Main Program Controlling the Robot.

```
package POSITION is
    type COORD is new FLOAT;
    type ANGLE is new FLOAT;
    -- COORD and ANGLE are declared "new" float|ing point types.
    -- This way they will not be confused with other FLOAT's.
    type FRAME is private;
    -- FRAME is the representation of one coordinate system in terms of another.
    function BUILD_FRAME(X,Y,Z: in COORD; R,S,T: in ANGLE) return FRAME;
    -- Allows FRAME's to be constructed from lower level primitives.  Necessary
    -- since FRAME is private and its structure cannot be directly accessed.
    function  "*" ( A, B : in FRAME ) return FRAME;
    -- This function expresses the coordinate frame represented by B in
    -- terms of the one in which A is  represented, i.e., it is a transformation.
private
    type FRAME is array (1..4,1..4) of FLOAT; -- A 4x4 homogeneous transformation.
end POSITION
```

Figure 7. Package Specification for Coordinate Frames and Related Operations.

```
with POSITION; use POSITION;
package ROBOT is
    SLOW: constant := 0.1;              -- Fine motion speed.
    FAST: constant := 1.0;              -- Approach speed.
    subtype SPEED is FLOAT range 0.1..1.0;
                                        -- Bound speed for safety check.
    procedure CALIBRATE;                -- Calibrate the robot arm prior to use.
    procedure MOVE(DESTINATION: in FRAME);
                                        -- Move to a point given by applying
                                        -- the transform represented by FRAME.
    procedure OPEN_GRIP;
    procedure CLOSE_GRIP;
    procedure SET_SPEED (SPD: in SPEED);
end ROBOT;
```

Figure 8. Package Specification for ROBOT.

```
with POSITION: use POSITION:
package CAD_MODEL is
    type PART_ID is private;
    type D_INFO is private;
    type DECISION_INFO is access D_INFO;
    type PART_SET is array (INTEGER range <>) of PART_ID;
    type STABLE_POSITION is private;
    type S_POS_SET is array (INTEGER range <> ) of STABLE_POSITION:
    function DECISION_TREE (S: in PART_SET) return DECISION_INFO;
    function STABLE_POS_SET (PART_NAME: in PART_ID) return S_POS_SET;
    function PICK_POINT (PART_NAME: in PART_ID;
            STABLE_POS: in STABLE_POSITION) return FRAME;
    function PICK_APP_POINT (PART_NAME: in PART_ID;
            STABLE_POS: in STABLE_POSITION) return FRAME;
private
        type PART_ID is new STRING (1..8);
                                            --Eight character part identifier
        type STABLE_POSITION is new INTEGER;
                                            --Index of stable positions.
        type D_INFO is                      -- Node in binary tree.
            record
                VALUE: FLOAT;
                LLINK: DECISION_INFO;
                RLINK: DECISION_INFO;
            end record;
end CAD_MODEL;
```

Figure 9. Package Specification for CAD_MODEL

```
with CAD_MODEL; use CAD_MODEL;
package VISION is
    type PART is
        record
            NAME: PART_ID;
            LOCATION: FRAME;
            STABLE_POS: STABLE_POSITION;
         end record;
    function FIND (D_T: in DECISION_INFO) return PART;
    -- Identifies the part, its location and the position it is in.
end VISION;
```

Figure 10. Package Specification for VISION.

References

[1] R. P. Paul, "Modelling, trajectory calculation, and servoing of a computer controlled arm," AIM 177 , Artificial Intelligence Lab., Stanford Univ. , Nov. 1972.

[2] R. H. Taylor, "The synthesis of manipulator control programs from task-level specifications," AIM-282, Artificial Intelligence Lab., Stanford Univ., 1976.

[3] C. Laugier and J. Pertin, *Automatic grasping: a case study in accessibility analysis.* Bazga, Italy : NATO-Advanced Studies Institute on Robotics & Artificial Intelligence, June 1983.

[4] P. Brou, *Implementation of high-level commands for robots (M.S. Thesis).* Dept. of Electrical Engrg. and Computer Science, MIT, Dec. 1980.

[5] H. Asada, *Studies in prehension and handling by robot hands with elastic finger.* Univ. of Kyoto, 1979.

[6] J. Wolter, T. C. Woo, and R. A. Volz, "Gripping position for 3D objects," *Proc. of the 1982 Meeting of the Industrial Applications Soc.* , pp. 1309-1314, Oct. 1982.

[7] C. Laugier, "Industrial robots," *Proc. 11th Int'l Symp.* , Oct. 1981.

[8] M. T. Mason, *Manipulator grasping and pushing operations (Ph.D. Thesis).* Dept. of Electrical Engrg. and Computer Science, MIT, 1982.

[9] T. Lozano-Perez, "The design of a mechanical assembly system," AI TR 397, Artificial Intelligence Lab., MIT, 1976.

[10] T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. SMC-11, no. 10, Oct. 1981.

[11] M. Wingham, *Planning how to grasp objects in a cluttered environment (M.Ph. Thesis).* Univ. of Edinburgh, 1977.

[12] W. A. Perkins, "A model-based language for manipulator control," *IEEE Trans. on Computers 27*, pp. 126-143, 1978.

[13] J. Sklansky, "On the Hough technique for curve detection," *IEEE Trans. on Computers*, vol. C-27, no. 10, pp. 923-926, Oct. 1978.

[14] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111-122, 1981.

[15] K. G. Shin, *Comparative study of robot programming languages.* Center for Research in Integrated Manufacturing, Univ. of Michigan, (to appear) 1983.

[16] W. A. Gruver, B. I. Soroka, J. J. Craig, and T. L. Turner, "Evaluation of commercially available robot programming languages," *Proc. of the 19th Int'l Symp. on Industrial Robots & Robots 7*, pp. 12-58 to 12-66, April 1983.

[17] *Ada Programming Language (ANSI/MIL-STD-1815A).* Washington, D.C. 20301: Ada Joint Programming Office, Dept. of Defense, OUSD (R&D), Jan. 1983.

[18] M. A. Wesley, "Construction and use of geometric models," *Computer Aided Design*, 1980.

[19] A. Rosenfeld and A. C. Kak, in *Digital Picture Processing.* New York, NY: Academic Press, 1976.

[20] H. G. Barrow and R. J. Popplestone, "Relational descriptions in picture processing,"

Machine Intelligence, vol. 6, pp. 377-396.

[21] C. L. Lawson and R. J. Hanson, in *Solving Least Squares Problems* Prentice-Hall, Inc. , 1974.

[22] T. Rentsch, "Object oriented programming," *Sigplan Notices*, vol. 17, no. 9, pp. 51-57, Sept. 1982.

[23] E. I. Organick, *A programmer's view of the Intel 432 system.* Santa Clara, CA 95051: Intel Corp., 1982.

[24] M. Shaw, "The impact of abstraction concerns on modern programming languages," *Proc. of the IEEE*, vol. 68, no. 9, pp. 1119-1130, Sep. 1980.

[25] N. Wirth, in *Programming in Modula-2 (2nd ed.).* Berlin, Germany: Springer-Verlag , 1982.

[26] P. B. Hansen, in *The Architecture of Concurrent Programs.* Englewood Cliffs, NJ: Prentice-Hall, 1977.

[27] R. Milner, "Theory of type polymorphism in programming," *Jour. of Computers and System Sciences*, vol. 17, pp. 348-375, 1978.

[28] D. I. Good and W. D. Young, "Generics and verification in Ada," *Sigplan Notices*, vol. 15, pp. 123-127, Nov. 1980.

[29] E. S. Roberts, A. Evans, Jr., C. R. Morgan, and E. M. Clarke, "Task management in Ada - a critical evaluation for real-time multiprocessors," *Software - Practice and Experience*, vol. 11, pp. 1019-1051, 1981.

[30] S. J. Kretch, "CAD/CAM for robotics," *Robot 7 Conf.*, March 1982.

[31] G. J. Gleason, "Vision module development," Ninth Report, NSF Grants APR75-13074 and DAR78-27128, SRI Projects 4391 and 8487, Stanford Research Institute, Menlo Park, CA, pp. 9-16, Aug. 1979.