Sensory feedback gives this integrated and knowledge-based assembly cell enhanced flexibility and lessens the need for high-priced parts feeders and precision fixturing.

A Knowledge-Based Robotic Assembly Cell

A. C. Kak, K. L. Boyer, C. H. Chen, R. J. Safranek, and H. S. Yang
Purdue University

For automation to be cost effective in small batch manufacturing, the use of special purpose parts handling equipment must be minimized by the use of adequate sensory feedback to enable a robot to interact with a random environment.[1] For this kind of manufacturing, an assembly cell should consist ideally of one or more robots, sensory elements (vision, tactile, torque/force), a motion control system, and supervisory intelligence; such a cell should be knowledge driven in the sense that it should "know" about the objects and plans that the assembly cell is supposed to deal with. This knowledge should also allow the robot to handle uncertainty in sensory data and to arbitrate between sensors in the event of conflicts.

The intent of this article is to provide an overview of one possible organization for an automated assembly cell that is under development in the Robot Vision Laboratory at Purdue University. We will present a knowledge-based system that consists of Supervisor, Global Knowledge Base, Current World Model, Motion Controller, and Sensory Subsystem modules. (Although the complete system is an ongoing project, significant contributions have been made in most of the system's components.)

We will also discuss the issue of object representation and mention our efforts on the slot-filler approach, which appears to work well for objects with distinctive landmarks; this approach can also be used for storing other items of information such as assembly instructions and any real-world constraints to be used in task planning.

For solid objects of high symmetry, not uncommon in the industrial world, we will review the Extended Gaussian Image concept for representation, and, in addition, present a simpler approach for generating the center coordinates of a tesselated Gaussian sphere.

For the Current World Model generator, we will discuss the radial-valued skeleton approach for objects that are primarily 2-D in nature and then show its extension into a novel method for solving the stereo problem symbolically by using information-theoretic image-to-image interprimitive distances and relational constraints. For generating the Current World information on 3-D solid objects, we will summarize some of our recent work on the creation of edge-vertex graphs from structured light data. We will then talk about the Motion Controller which consists of a flexible user-friendly manipulator interface.

Finally, some experimental results in parts mating using 3-D vision feedback will be shown.

System overview

Until recently, not much was published on the organization of sensor based assembly cells. In the past, manipulator systems have usually consisted of a robot with special-purpose fixturing; if sensory capability was required, a vision system (and, sometimes, other sensors) was grafted onto the configuration.[2,3] This approach presents a number of difficulties, the most significant of which is the lack of a truly integrated system-level design.

Recently, there has been some work at defining configurations for more general purpose, and better integrated, systems. Alami[4] describes a system being developed in France under the ARA (Automatisation et Robotique Avanceesa) program. This system uses a Lisp-based environment to integrate the sensory, manipulation, and decision processing required for research in automated assembly. Several communicating processes support two manipulators and a vision system.

0885-9000/86/0200-0063$01.00

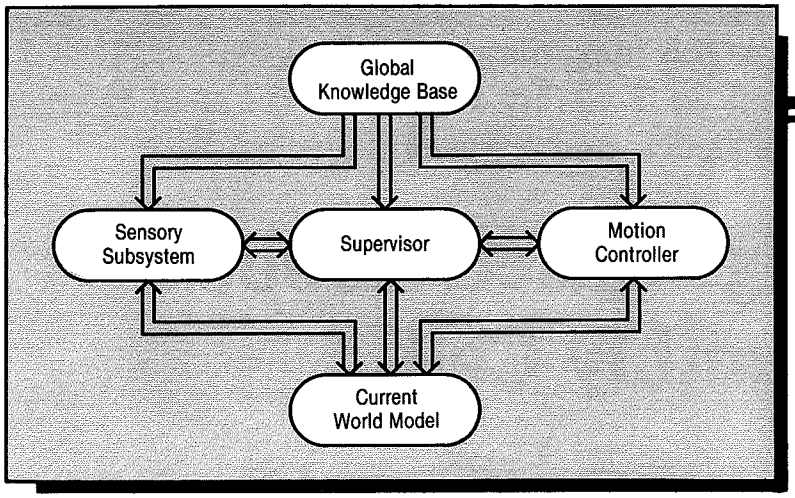


Figure 1. Control functions for the assembly cell distributed across several subsystems.

Lee and Goldwasser[5] have recently presented a distributed system that is being explored by the General Robotics and Active Sensory Processing Group at The University of Pennsylvania. This configuration consists of a network of several mini- and microcomputers, with sensory and manipulation tasks routed to the most appropriate computing resource. Their published work focuses on distributed communication and operating system issues. Shin and Epstein[6] have examined the communication problems inherent in a multiprocessing robotics environment. They propose a set of process-level communication primitives that can be used to coordinate the components in an assembly cell.

The system presented here is designed as a framework for research on sensory systems and sensor-guided motion. It has the following design goals:

- Flexibility: Since its first use will be as a research tool, the system should permit incorporation of new methodologies and programming paradigms, as they become available, for each of its major components.
- Modularity: To implement a system of this size, a broad spectrum of technologies must be integrated. To simplify system integration, general specifications for several independent subsystems were established. Each of these subsystems is a self-contained module performing a subset of the tasks required for the cell's operation. The modules communicate through a well-defined message-based network.
- Ease of use: This feature is probably the most important. People with vastly different backgrounds are cooperating to bring the project to fruition, each working in a particular area of expertise. However, since each must also interact with system components outside his or her area, the interfaces between components must be easy to use. Human-robot interaction, in particular, should occur at a sufficiently high level so that persons not skilled in robotics can use the system effectively.

A block diagram showing the component subsystems and their interconnections is given in Figure 1. The subsystems consist of Supervisor, Motion Controller, Global Knowledge Base, Current World Model, and Sensory Subsystem(s). An earlier version of this system is detailed elsewhere.[7]

The Supervisor. The Supervisor coordinates and controls the activation of the entire system and the interaction between the components. It is given assembly instructions for a product and, from them, determines the necessary sequence of operations to successfully complete the task. After the task plan is derived, requests for action are sent to the appropriate subsystems, their responses monitored, and adjustments in the task plans made continuously on the basis of the latest sensory information available to the Supervisor.

The Motion Controller. The Motion Controller accepts action requests from the Supervisor, determines their feasibility, and handles their execution. This module is in charge of controlling all electromechanical devices in the work cell; the devices could include one or more manipulators, grippers, parts handling machinery, and programmable work-piece positioners (Figure 2).

The actions of the Motion Controller are divided into planning and control functions. Planning functions consist of path planning and collision avoidance, while control involves the origination and verification of movement requests to the various motion execution units, or MEUs.

Each programmable device has an associated MEU. These modules take action requests from the Motion Controller, supervise their execution, and note any exceptional conditions. To accomplish this, each MEU maintains a model of the current state of the device under its control. As actions occur, this model is continually updated with information received from the device's control system. If an error condition occurs, execution is halted and the Motion Controller is notified of the nature and extent of the difficulty. Since the MEU is closely coupled to each device's control system, it is in charge of monitoring the fast sensory feedback loops required for force and contact sensing. For a manipulator, the MEU would be an augmented version of a conventional robot controller.

The Global Knowledge Base. The Global Knowledge Base contains everything the assembly cell knows about its world. This knowledge includes part and fixture models, assembly instructions, and any real world constraints used in task planning.

The GKB provides a central repository for all the long term knowledge required by the system. This approach has several advantages over distributing the system's knowledge among the subsystems. Since the knowledge is centralized, special-purpose knowledge representation and database languages

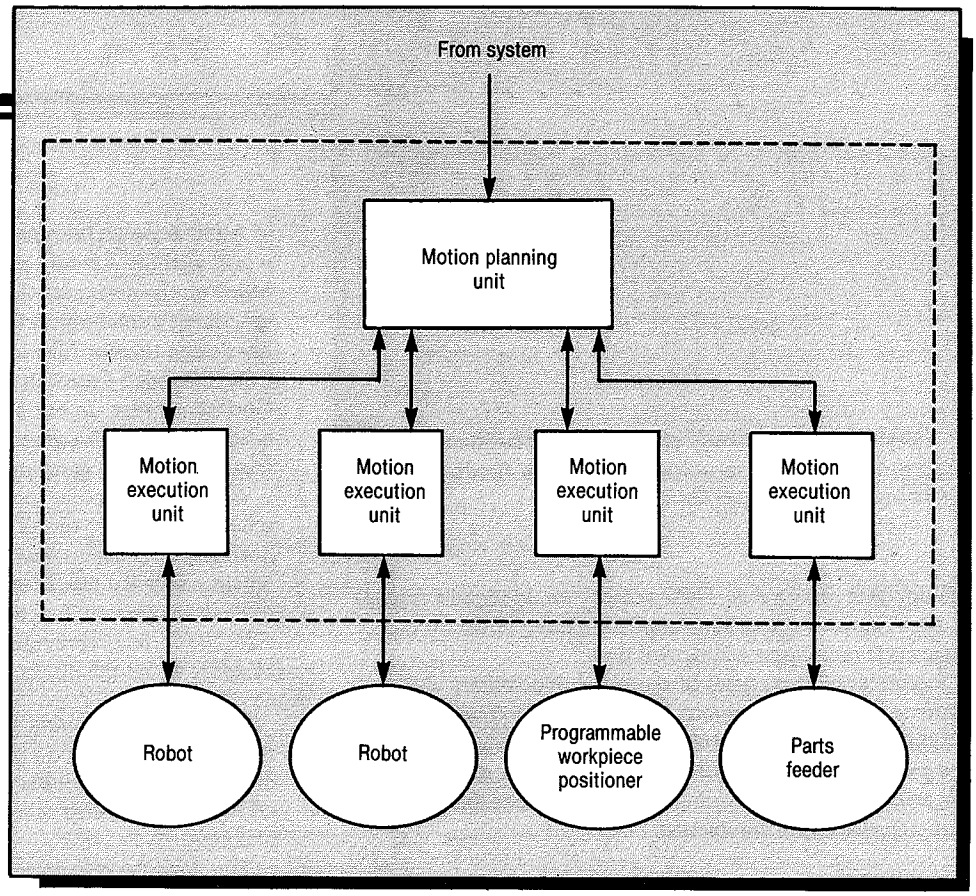


Figure 2. The Motion Controller. The Motion Controller coordinates the planning and execution of all movements within the cell.

that are relatively easy to implement can be used to ease the burden of depositing new information and modifying or retrieving existing information. If each subsystem maintained its own database, knowledge base additions and corrections could become major undertakings. Note that the information stored in the GKB might be useful to more than one subsystem in the cell. For example, the Motion Controller may need to know about the object models in the GKB for path planning, while the Sensory Subsystem may need to know about them for position verification and object recognition.

The Current World Model. The Current World Model maintains a description of each known object in the work cell, as well its location and and orientation. Also included in the CWM are the locations and orientations of the end-effectors on the manipulators. At any instant, the information contained in the CWM should be sufficient to reconstruct the current state of the cell.

Entries in the CWM are changed whenever a new object is identified, an object moved, two or more objects assembled, or one of the manipulators or positioners moved. Since access by the other components to the data housed in the CWM is asynchronous, care must be taken to make sure that at any given time the entries are valid. For example, assume that the current task is to put a cover on a box. The Supervisor requests the Motion Controller to move the manipulator holding the cover to a position over the box; it also requests the vision system to verify that the cover is in a position that will allow it to be installed. Since each subsystem operates asynchronously, there is no guarantee that the manipulator will have the cover in position before the vision system receives its request. Therefore, some sort of interlock is required on CWM entries while the manipulators are moving. The use of semaphores[8] is one way to ensure this type of data integrity; the *waitfor* primitive of Shin and Epstein[6] is another possible approach. With interlocks, the CWM entries for the cover and manipulator are locked at the start of the transfer movement and released when the motion has been completed. The vision system then waits until these entries are released before inspection. A time sequence for this scenario is given in Figure 3.

The Sensory Subsystem. The Sensory Subsystem consists of sensors requiring a processing cycle that is greater than the

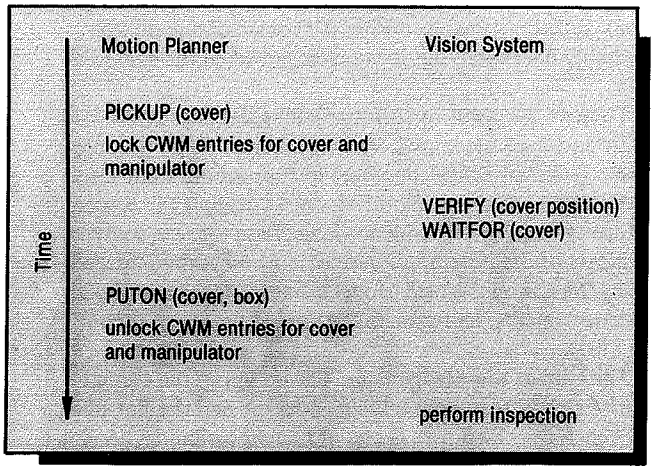


Figure 3. To ensure data integrity, data locking capability is necessary in a distributed assembly cell.

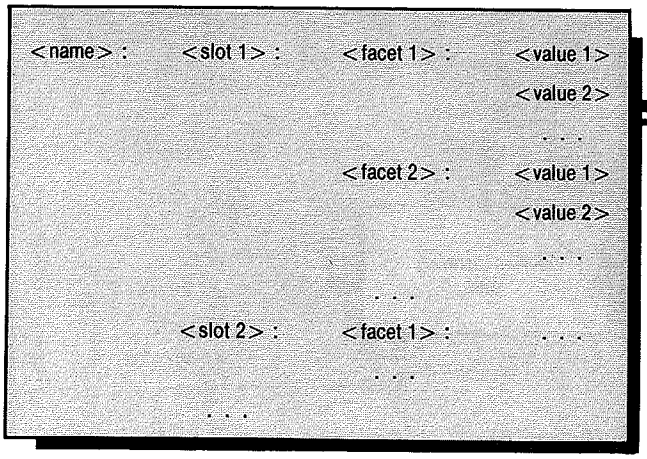


Figure 4. Slot-filler paradigm provides a flexible scheme for knowledge representation.

manipulator servo rates. These sensors would require special-purpose hardware and significant computing power to produce cycle rates approaching real time.

Examples of these types of sensors include binary vision systems,[9] 3-D vision systems,[10-13] and tactile arrays.[14,15] Common tasks for these sensory types include determining the identity and location of objects in the work area, inspecting assemblies, and providing the feedback necessary to ensure successful task completion.

Knowledge bases and representation schemes

In a complete robotic assembly cell, the types of knowledge required for system operation fall into two classes: static and dynamic. Static knowledge consists of knowledge that is unaffected by the system state; examples include object models, assembly instructions for a particular task, and rules of inference. Dynamic knowledge is knowledge of the environment, which must be updated as the system state changes. Dynamic knowledge reflects the current state of the world, accounting for all actions taken by the robot and the effect of external operators on the world state as sensed.

The term static does not mean that such information is never changed, altered, or updated. This type of information is not subject to change during the course of a complete assembly operation, but it can changed when the system is idle.

To meet the needs of software packages that require both static and dynamic knowledge, we maintain two knowledge bases: a Global Knowledge Base to support the static information and a Current World Model to support the dynamic information. The Global Knowledge Base is accessed by all elements of the system, but altered only by human intervention. The Current World Model is likewise accessible by all system elements, and can also be altered by any of them.

The Global Knowledge Base. The design of the Global Knowledge Base is a continuing endeavor and the slot-filler approach described here is one of many possibilities. A slot-filler form can be cast into a graph data structure, an advantage being that subgraph matching techniques can now be used for tasks such as object identification.

Another powerful representation strategy described here works well for 3-D objects with no special landmarks. It is based on the notion of Gaussian spheres. For example, describing a solid object like a dodecahedron would be very hard using the slot-filler approach, but rather easy using the EGI (Extended Gaussian Image) technique derived from the Gaussian sphere. In our discussion on the EGI, we will show a simple method for generating the center coordinates of the tesselations on a Gaussian sphere.

An important point to note is that no single representation scheme will suffice for all classes of objects that a robot will normally be asked to deal with; therefore, a repertoire of schemes is called for. Eventually, a top-level expert will have to be developed, its function would be to automatically invoke the correct representation strategy on the basis of partial evidence about the object under examination.

Slot-filler representation. The slot-filler form of knowledge representation is used to store (1) information on object models that can be characterized by distinctive landmarks and (2) information, such as assembly plans, required for task planning.

The slot-filler structure is a generalization of the frame concept proposed by Minsky.[16] Several special-purpose knowledge representation languages based on this scheme have now been developed; these include FRL (Frame Representation Language),[17] KRL (Knowledge Representation Language),[18] and PEARL (Package for Efficient Access to Representations in Lisp).[19]

An example of a generic frame is shown in Figure 4. We may think of this generic form as follows: We associate with each object a number of slots, an example being a parts-inventory slot. With each slot, we associate a number of facets. For example, in the part inventory slot of a wrench, we might have facets for "grip," "loop," and "lip," these being the features of a thinned version of a wrench image (Figure 5). Finally, with a facet we associate a value, if there is a value. For example, with the "grip" facet, we may associate a value of 1, meaning that there is only one grip in the image of a wrench. A slot-filler data structure is a generalization of the frame structure in the sense that an arbitrary degree of nesting is allowed. That is, we do not stop at the nesting implied by the slot-facet-value form. Instead, slots can occur within slots, and this recursive inclusion can proceed to an indefinite extent. (Our previous work on the use of the slot-filler representation is documented elsewhere.[7])

As Figure 5 illustrates, the frame for a complete object contains information describing how the object is composed of simpler parts, including inventory and interconnection information. Figure 6 is a frame for an item in the parts inventory of Figure 5. The inventory slot holds a list of all the simple parts that may appear in making up the complete object, and their respective quantities. The node-members slot contains parts connection information. Parts appearing together may be expected to share common connections in a structural description. Other object structure information may also appear, as appropriate.

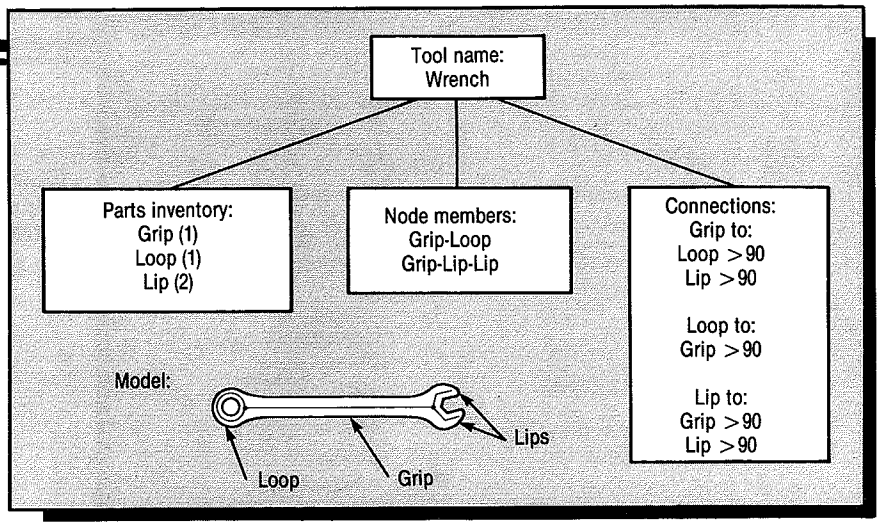


Figure 5. Object decomposition into parts and structural information.

A complete object is decomposed into its simple parts by separation into structural primitives. For example, pliers would be segmented into two handles and two jaws, while a screwdriver would be segmented into handle and shank.

The frames for object parts contain information about the geometrical features of extractable simple parts, uses for those parts, action to take if any of them appears alone, etc. These simple parts can be used as primitive elements in an object recognition system. Possible geometrical features used in these frames include aspect ratio of the bounding ribbon, straight (true or false), corner (true or false), cross-section (constant or variable), loop (true or false), curvature (constant or variable), axial inflections (number), and end characteristics (pointed or blunt). This information is treated as fuzzy, and variable weighting may be applied to each item depending on its importance in determining the geometry of a given part. As an example of the fuzziness, allowable values of aspect ratio are elongated, slightly elongated, square, and very elongated.

A frame for an object part also contains information for the application of that part, meaning a list of all the objects to which this part can belong. This list is used in an object recognition scheme wherein parts are recognized by their geometrical properties. These results then guide a search through the complete object frames.

Information to assist with task planning is also stored in slot-filler type structures (Figure 7). These representational units can be thought of as steps in a hierarchical set of assembly instructions. The *argument* field contains all global variables used by this stage of the plan. Substeps of the plan are specified in the *component-plans* field. If the plan is primitive—that is, a hard-coded procedure—this field is empty. Verifying correct execution is the responsibility of the *verify* field. If verification is desired, it contains the necessary parameters. Finally, if an error occurs during execution, the *error* field provides an error handler that can be invoked as needed.

To illustrate the use of this representation, Figure 8a shows in a slot-filler form and Figure 8b in a schematic form, the execution sequence for moving an object to a specified location. First, the plan for moving an object is retrieved from the GKB. This plan is found to consist of two subplans—

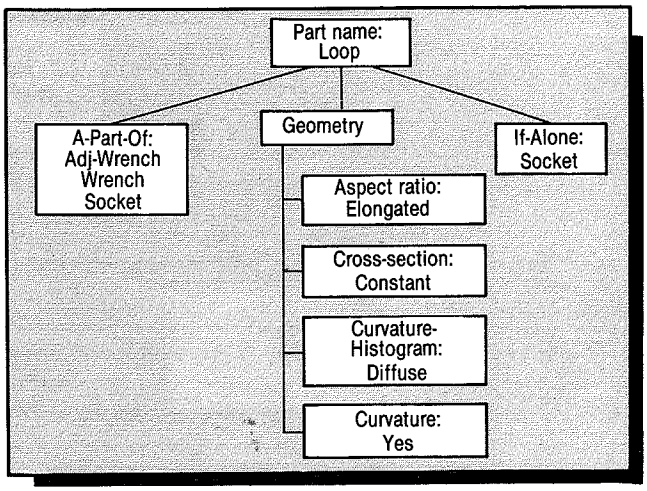


Figure 6. Object-part descriptions with geometric and contextual information.

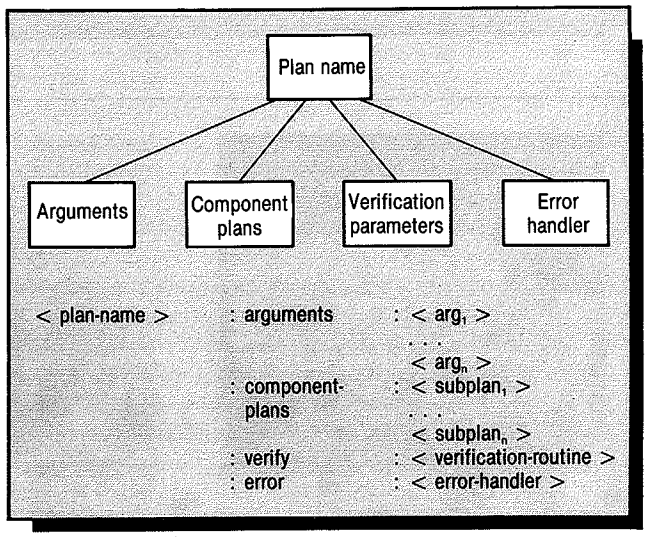


Figure 7. Assembly plans in slot-filler form (a) and a pictorial representation of a generic plan (b).

		: *object location*
		: (pickup-object, *object*) (putdown-object, *object location*)
move-object	: arguments	
	: component-plans	: position-of *object*
	: verify	: (object-position-error)
	: error	: *object*
pickup-object	: arguments	
	: component-plans	: (pickup-approach *object*) (grasp *object*) (depart *object*
	: verify	: (position-of *manipulator*)
	: error	: (manipulator-position-error)
pickup-approach	: arguments	: *object*
	: component-plans	: Ø
	: verify	: (position-of *manipulator*)
	: error	: (manipulator-position-error)

(a)

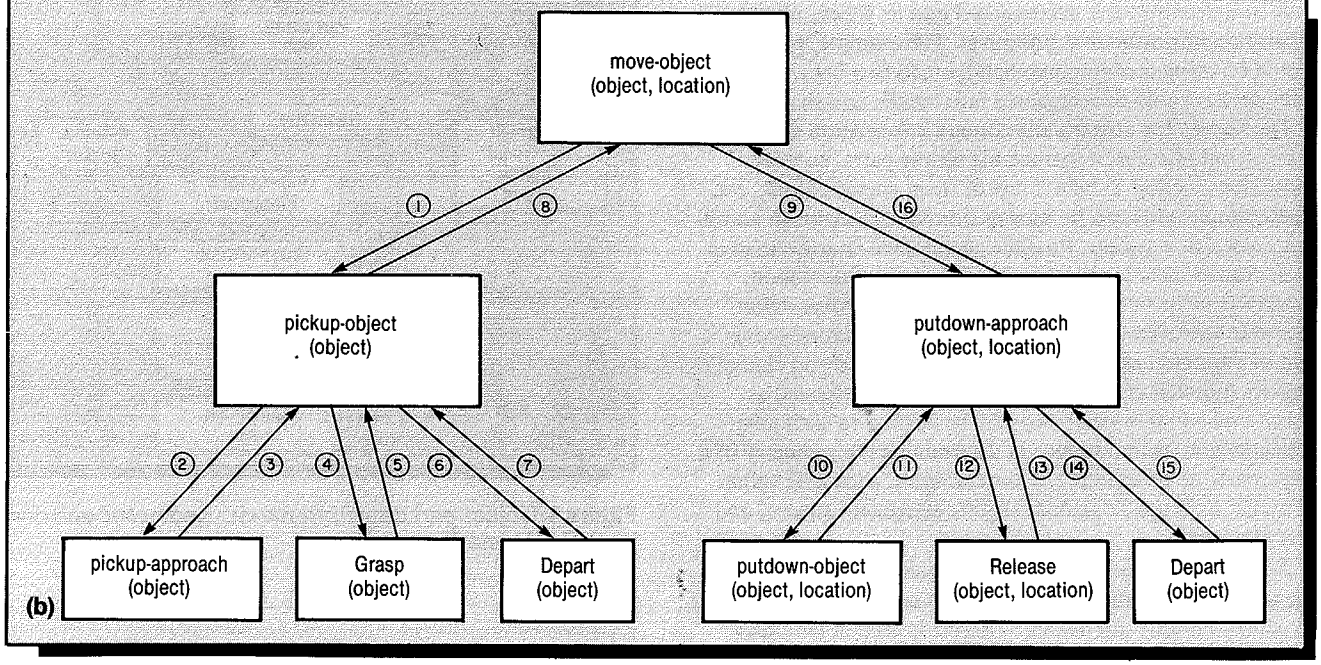


Figure 8. Representations of three simple plans (a) and a flow diagram showing the execution order for *move-object* (b).

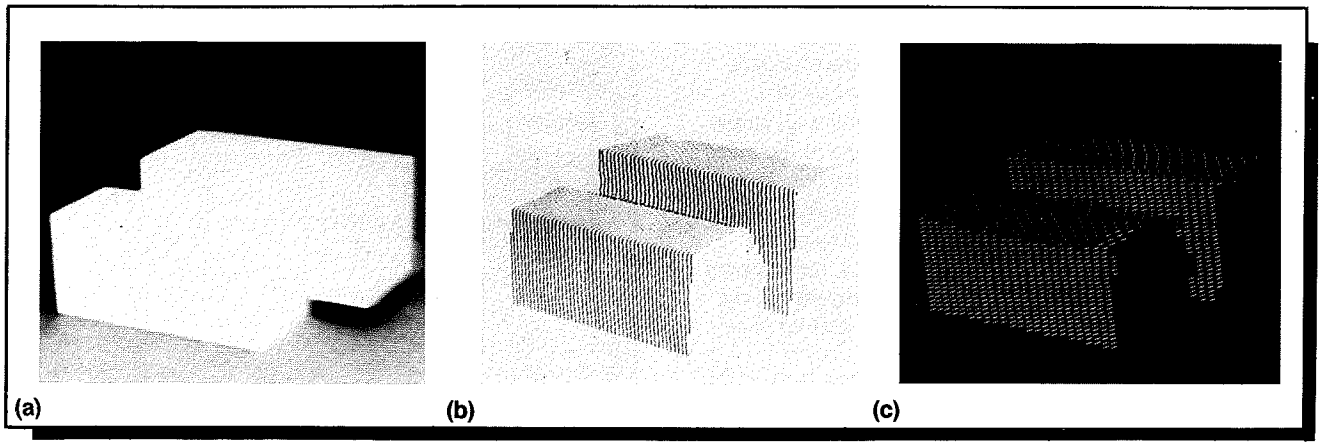


Figure 9. A scene consisting of two boxes (a), light stripes projected by a structured light sensor (b), and surface normals computed from the range map (c).

pickup-object and *putdown-object*. The arguments of *move-object* are unified with those of the subplans, and then the sub-plans are invoked in order. In this case, the plan for *pickup-object* is retrieved, and the plan-invocation procedure is repeated. At the next level down, *pickup-approach* is found to be a primitive plan, so its procedure is invoked. Control now shifts back to the *pickup-object* frame. The other subplans, *grasp* and *depart* are similarly invoked. As each plan is completed, the corresponding sensory verification is performed. For example, verification for the *grasp* routine determines if the gripper has actually closed on an object. At a higher level, the verification for *move-object* determines if the object is actually in the desired location.

At least for storing knowledge about objects, we can view the structural descriptions captured by a slot-filler representation as an attributed structural graph. Such graphs are useful for solving problems involving object identification and location via structural descriptions through the use of graph homomorphisms and subgraph isomorphisms.[20,21] Following Shapiro and Haralick,[20] we will give but a flavor of symbolism behind such graph-theoretic forms.
A structural description at the top level is a pair

$$D = (P,R)$$

where P is a set of object primitives and R is a set of named $n$ary relations over the primitives. An object primitive is characterized by a set of attribute-value pairs. So we have

$$P = \{p_1,\ldots,p_n\}$$

with each primitive defined in the product set

$$p_i \subseteq A \times V$$

where A is a set of attributes and V is a set of values for those attributes. As mentioned, R is a set of named $n$ary relations over P:

$$R = \{R_1\ldots,R_K\}$$

For $k = \{1,\ldots,K\}$ we have

$$R_k = (NR_k,R_k)$$

Where NR_k represents the name assigned to the kth relation and R_k is a set of M_k tuples of primitives having that relationship. For each k, there is an integer M_k such that

$$R_k \subseteq P^{M_k}$$

Thus, R_k is a set of M_k tuples drawn from the primitive set P. We refer to M_k as the cardinality of the relation NR_k.

The idea here is that, if necessary, an object recognizer would construct a structural description as an attributed graph from the frames stored in the GKB, and then compare this graph with those obtained from CWM. In practice, we do not want to get bogged down in a combinatorial explosion, so we must either use some form of a hypothesize-verify approach or order the primitives before invoking procedures such as subgraph isomorphism. A probabilistic method for ordering the primitives for this purpose is described elsewhere.[22]

The Extended Gaussian Image. Although the slot-filler approach is very flexible and rather forgiving of scene segmentation errors, its successful use requires that the object possess distinctive landmarks. For solid objects that are highly symmetrical—not an uncommon occurrence in industrial assembly—use of the slot-filler approach is difficult. One of the best alternatives, in our opinion, is based on the concept of the Extended Gaussian Image. For many solid objects of high symmetry, this representation can be used both for recognition and determination of orientation. In fact, for convex objects, such a representation is uniquely invertible, since only one representation is possible for a convex object, and from that representation, the object can be reconstructed. Horn gives a good introductory survey of the subject.[23]

At a simple level, the EGI, at least in its discrete form, can be viewed as an orientation histogram. The first step in obtaining the histogram is to divide the surface of a unit sphere into cells—a process called the tesselation of the spherical surface. The next step is to characterize each cell by the orientation of the surface normal at the center of the cell and then assign to the cell all the object surface normals that have the same orientation, under the assumption that all normals on the object surface represent equal elemental areas. (If the computed normals on the object surface do not represent equal elemental areas, they must be multiplied by suitable obliquity factors.) Note that such a representation is object centered, as opposed to viewpoint centered, and that we have assumed the availability of a discrete approximation for the object surface, as might be generated by a 3-D vision system. Figure 9 is an example of a discrete approximation to the surfaces in a scene; this approximation was generated from a range map obtained with a structured light scanner. Figure 9a shows the original scene consisting of two boxes, Figure 9b shows the light stripes, and Figure 9c the computed surface normals.

To construct a discrete approximation of the EGI by computer, the surface of the unit sphere must be divided into cells. Ideally, these cells should have the same area and shape, which should be regular (after all, in most cases, nobody wants to use a histogram with dissimilar cells). These conditions on cell area and shape can be satisfied by projecting a regular polyhedron onto a unit sphere after bringing their centers into coincidence. There are only five regular solids for this purpose: three of them bounded by equilateral triangles (tetrahedron, octahedron, icosahedron), one bounded by squares (cube), and one by regular pentagons (dodecahedron).

Although the tesselations generated by regular solids satisfy the area and shape constraints, even for an icosahedron, which gives us the largest number of cells, 20, the resulting sampling of the Gaussian sphere is simply too coarse. Finer subdivisions are obtained by splitting each face of a given tesselation into triangular facets. If we divide each face of a dodecahedron into five equal triangles, the resulting polyhedron is called the pentakis dodecahedron and consists of 60 cells. Further subdivisions are usually accomplished by employing the technique of geodesic tesselation, which consists of dividing each edge of the solid into sections. If we

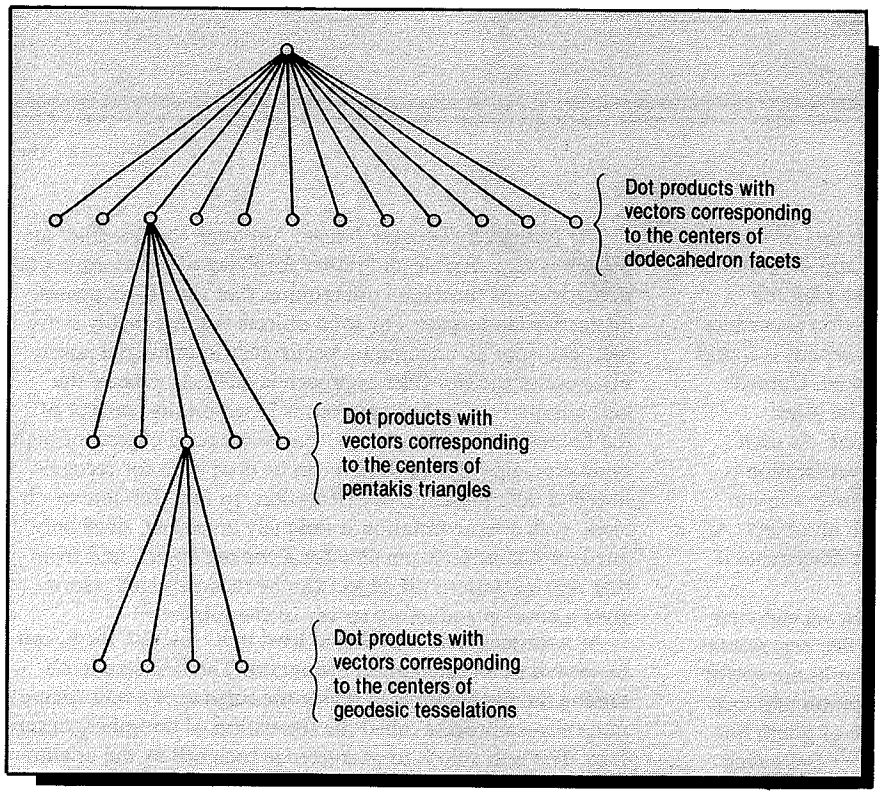


Figure 10. Locating cells via hierarchical descriptions. Given an object surface normal, for increased computational efficiency it is possible to locate the corresponding cell on the EGI by utilizing a hierarchical description of the tesselations on the Gaussian sphere.

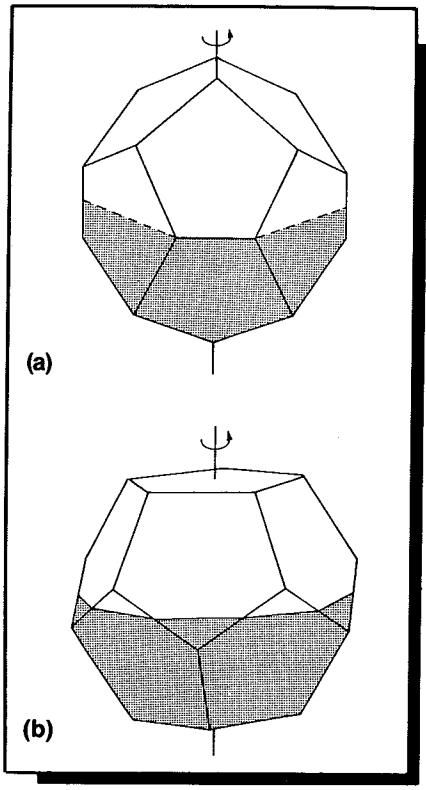


Figure 11. Axes of twofold (a) and fivefold (b) symmetry of a dodecahedron.

divide each edge into two sections, every cell of a pentakis dodecahedron would split into four smaller triangles, giving us a total of 240 cells.

To construct an orientation histogram from measured data, we must take the dot product of a measured unit vector with the vector corresponding to the center of each cell. The best candidate cell is then determined as the one that gives the largest dot-product. Depending on the number of surface patches used, this brute force approach to constructing histograms can be computationally demanding. However, for geodesic tesselations, this calculation can be performed hierarchically, thereby reducing the computational burden. As was mentioned by Horn,[23] to hierarchically assign a measured surface normal to a cell, we first take the dot-product of the measured normal, with unit vectors corresponding to the centers of the dodecahedron's facets. The measured normal is assigned to the facet yielding the maximum value. Next, we take dot-product of the measured normal with vectors corresponding to the centers of pentakis triangles for the dodecahedron facet. We now assign the measured normal to the triangle yielding the maximum value. This procedure is repeated with the geodesic tesselations of the pentakis triangles. Figure 10 is a pictorial depiction of this hierarchical search for the correct cell on the Gaussian sphere.

Clearly, for the above hierarchical procedure to work, we must first determine the center coordinates of the 12 pentagons of the dodecahedron, of the 60 larger pentakis triangles, and, finally, of the 240 smaller triangles obtained by geodesic tesselation.

What follows is a simple procedure for computing these center coordinates for frequency-two geodesic tesselations. (The 240-cell tesselation is also called a frequency-two geodesic dome, since each edge of the pentakis triangles is divided into two intervals.) For this procedure, we use the axis of twofold symmetry for a dodecahedron, which is shown in Figure 11a as the z-axis of a Cartesian coordinate system. We next write down expressions for the vertices of the dodecahedron in this coordinate system, first recognizing the following relationship between the radius, R_c, of the circumscribing sphere of the dodecahedron and the length, ℓ, of any of its edges:

$$R_c = 1.4013\ell_1,$$

Since the Gaussian sphere has unit radius, implying $R_c = 1$, the edge length ℓ_1 must equal

$$\ell_1 = \frac{1}{1.4013} = 0.7136.$$

The twofold symmetry for a dodecahedron shown in Figure 11a leads, in most cases by inspection, to the following expressions for the coordinates of the 12 vertices shown in Figure 12a:

$$P_1 = (-\frac{\ell_1}{2}, \ell_3, 0)$$

$$P_2 = (\frac{\ell_1}{2}, \ell_3, 0)$$

$$P_3 = (-\ell_3, 0, \frac{\ell_1}{2})$$

$$P_4 = (\ell_3, 0, \frac{\ell_1}{2})$$

$$P_5 = (0, \frac{\ell_1}{2}, \ell_3)$$

$$P_6 = (0, -\frac{\ell_1}{2}, \ell_3)$$
$$P_7 = (-\ell_4, \ell_4, \ell_4)$$
$$P_8 = (\ell_4, \ell_4, \ell_4)$$
$$P_9 = (-\ell_4, -\ell_4, \ell_4)$$
$$P_{10} = (\ell_4, -\ell_4, \ell_4)$$
$$P_{11} = (-\frac{\ell_1}{2}, -\ell_3, 0)$$
$$P_{12} = (\frac{\ell_1}{2} - \ell_3, 0)$$

where $\ell_1 (= 0.7136)$ is the side length of a dodecahedron inscribed in a unit sphere; ℓ_2 is the length of the line connecting a vertex and a midpoint of its opposite side of the pentagon (Figure 12b); ℓ_3 is the distance from the origin of coordinate system to the midpoint of the edge where two pentagons meet; and $\ell_4 = \ell_1 \cos 54°$. The other eight vertices not visible in Figure 12a are symmetric to P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10} with respect to the x-y plane, so their x and y coordinates are identical to their symmetric counterparts and only the z coordinates change sign.

Note that in the coordinate system used here, all 60 coordinate values (for 20 vertices) can be described in terms of four unique numbers (ℓ_1, ℓ_3, ℓ_4, and 0).

Once the coordinates of all 20 vertices are determined, the center of each spherical pentagon projected on the unit sphere can be calculated by first averaging the coordinates of the five vertices forming such a pentagon, and then normalizing the resulting vector by its magnitude to yield a unit vector. The same procedure is used to find unit vectors corresponding to the centers of the triangles of the pentakis dodecahedron, and, finally, those of the smaller triangles belonging to the geodesic tesselation.

If the coordinate system in which the z-axis corresponds to the axis of fivefold symmetry (Figure 11b) is preferred, the coordinates of all the vertices can be deduced from those given above by using an appropriate rotational transform.[12]

Note that the number of unique values for describing the vertex coordinates in the two systems shown in Figure 11 is not the same; Table 1 shows a comparison. It would appear that by using the system of Figure 11b, fewer unique values are required. In Table 1, Type 1 refers to the system in Figure 11a (twofold symmetry), and Type 2, to the system in Figure 11b (fivefold symmetry). The table shows that we need eight unique values for describing the center coordinates for all the pentagons in Figure 11b, whereas only three unique values are required in Figure 11a. The second row shows that 30 unique numbers are required for describing the center coordinates of all the pentakis triangles in Figure 11b, whereas only nine unique numbers are needed for the same purpose in Figure 11a, and so on. This implies that a look-up table for the centers of cells of the EGI would be much more efficient for the coordinate system of Figure 11a, than for the one in Figure 11b.

The Current World Model. The Current World Model maintains a representation of the current status of the robot's working environment. Examples of knowledge to be maintained include location and, if known, identity of objects in the work area; status of the robot manipulator; status of any environmental variable, such as special-purpose lighting, which is subject to change for completing the task at hand or sensing the world state. The techniques described here model the current world state as obtained from vision packages within the sensory subsystem.

Edge-vertex drawings from structured light images. To generate the CWM information on polyhedral objects from 3-D vision data as acquired with structured light techniques, we often use the edge-vertex drawing representation.[24,25] The edges in the scene are assigned labels according to type: con-

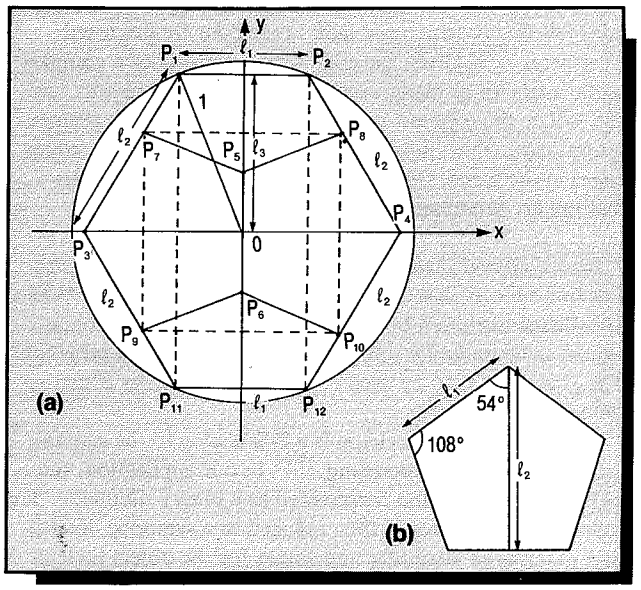


Figure 12. A cross-section of a dodecahedron viewed along the axis of twofold symmetry (a) and a regular pentagon (b).

Table 1. The number of unique values required for the center-coordinates of the tesselated Gaussian sphere computed from two different coordinate systems. Type 1 reflects the axis of twofold symmetry, while Type 2 reflects an axis of fivefold symmetry.

Coordinate System / Cells	Type 1	Type 2
Pentagon	3	8
Larger triangle	9	30
Smaller triangle	32	133
Total	44	171

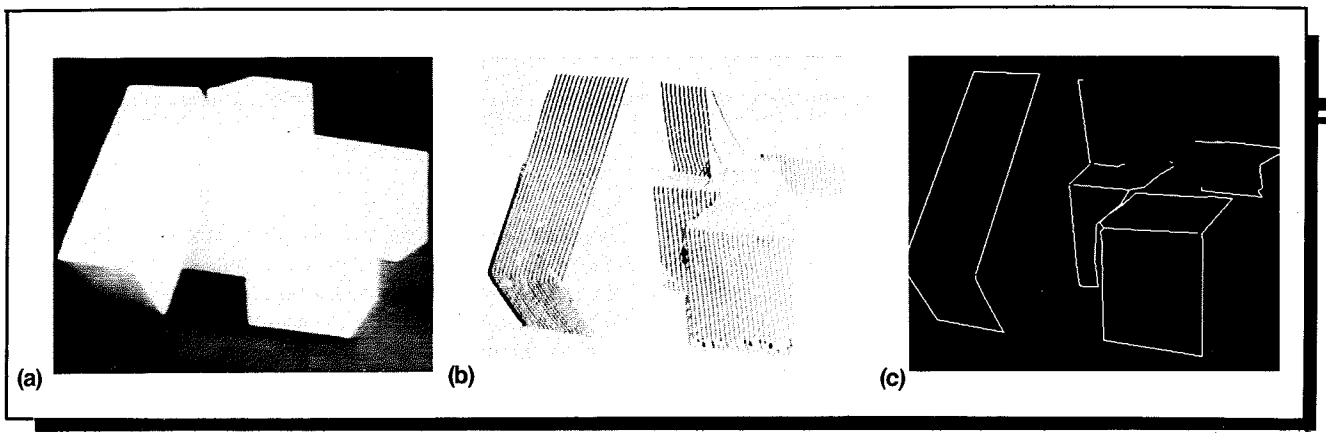


Figure 13. A scene consisting of five overlapping boxes (a), light stripes projected by a structured light sensor (b), and the edges extracted from the range data (c). Although it is not possible to display here, the edge labels are also known; the labels being: convex, concave, interior occluding, and exterior occluding.

vex, concave, interior or exterior obscuring. Figure 13a shows a multiobject scene from which edges and vertices are extracted using a structured light range map (Figure 13b). Figure 13c shows the result. (Methods for edge detection and edge labeling in structured light range data are discussed elsewhere.[25,26]) Given the edge-vertex drawing of a complex scene made up of many overlapping polyhedral objects, we can isolate the topmost object by defining an object segmentation path, which lies along concave and interior obscuring edges in the scene. Object segmentation results using this method are presented elsewhere.[24]

Radial-valued skeletons and attributed structural graphs. For thin objects that are primarily 2-D in nature, the CWM knowledge can often be adequately represented by a radial-valued skeleton and a subsequent attributed structural graph. The graph can then be used to derive the slot-filler constructions described earlier.

The radial-valued skeleton is a 2-D analog to the generalized cone for object modeling.[27,28] In our application, the object or region is first represented as a binary silhouette. The radial-valued skeleton is then reduced to a symbolic description in terms of attributes and values for the object skeletal components. That is, the skeletal arms, and possibly their junction points, serve as the primitives in the structural descriptions we build.

Once we have computed the radial-valued skeleton as an image array, a symbolic list description is constructed in a single raster scan. This description represents the skeletal arms by their endpoints, chain code descriptions of their trajectories in the image plane, and pixel-by-pixel radial and gray-level values. Because the description of the algorithm that constructs the symbolic list is quite lengthy, we omit it here; it is presented in detail elsewhere.[22] Suffice it to say that the list is a deeply nested structure, as shown by its pictorial representation in Figure 14.

The next step in the modeling process is to build an attributed structural graph from the radial-valued skeleton. The attributed structural graph involves assigning attributes to the edges and vertices of a graph formed by associating skeletal arms with edges and arm endpoints with vertices. Some attributes for which we may assign values directly from the list representation of the radial-valued skeleton include edge attributes such as

- M-orient: Mean orientation
- Asp-ratio-rib: Aspect ratio of the bounding ribbon
- Masp-ratio-rib: Aspect ratio of the mean-bounding ribbon
- Edge-length (in pixels), Edge-mass: Area of binary feature
- Dom-radius: Dominant radial value
- Eight-ratio: Ratio of length to 8-distance between endpoints
- Straight-ratio: Ratio of length to Euclidean distance between ends
- Loop predicate: True or False
- Gray-level behavior (roughness, taper, discontinuities)
- Gray-level inflections, peaks, valleys

and vertex attributes such as

- Degree (number of incident edges), location
- Edge orientation sequence
- Edge length sequence
- Local gray level roughness
- Local mean gray level
- Nearest neighboring vertex, distance thereto

We now extend the abstraction of the scene description in terms of the radial-valued skeleton to an attributed graph. We identify branchpoints, or their clusters, with vertices in the graph and skeletal arms with edges in the graph. Arm endpoints that are not branchpoints are identified as vertices of degree one.

We define the attributed graph G′ to be an ordered five-tuple:

$$G' = (V(G), E(G), \Psi_G, \Upsilon_G(V), \Xi_G(E))$$

where, V, E, and Ψ_G are the vertices, edges, and mapping of edges onto vertices, respectively. Υ and Ξ are sets of attribute-value pairs assigned to vertices and edges, respectively. Specifically, let A' and V' represent a set of named attributes and their allowable values, respectively. Then:

$$\Upsilon_G = \{ v_1(v_1), \ldots, v_v(v_v) \}, \Xi_G = \{ \xi_1(e_1), \ldots, \xi_\epsilon(e_\epsilon) \}$$

So each element of the mappings Υ and Ξ is a set of binary relations:

$$v_i \subseteq A' \times V' \text{ and } \xi_i \subseteq A' \times V'$$

The description of an attributed graph comprises three parts. These are the augmented adjacency matrix, which represents the mapping Ψ_G, the attributed vertex descriptions,

which represent the set Υ_G, and the attributed edge descriptions, which represent the set Ξ_G.

The augmented adjacency matrix is $\nu \times \nu$. At each point we store the number of edges appearing between the two vertices identified by the row and column index and a list of the ordinal values assigned to those edges.

To construct structural descriptions from attributed structural graphs, we define and invoke relational predicates.

In the attributed graph, the relations among the primitives (edges and vertices) are built-in, but, with the exception of the explicit information in the augmented adjacency matrix, not precisely defined or named. To make the best use of the information available, we determine which *n*ary relationships are effective, define predicates to compute the primitive sets entering into those relationships, and apply these to construct the structural description. Some examples of relational predicates that may be computed for the edges in a graph as primitives in a structural description include

- One endpoint in common (close)
- Two endpoints in common (close)
- Collinear, parallel, orthogonal
- Left-of, Above

With these structural descriptions extracted from real images, we may invoke stereo matching, object recognition, and so on.

Sensory Subsystem

Many vision processes are being investigated in the Robot Vision Laboratory. No single approach to the vision problem is capable of providing the flexibility needed to support automated assembly. Therefore, several different efforts are underway, including passive stereo, eye-in-hand, linear scan structured light, and color-encoded structured light. Because the information derived by these techniques will be used to guide the task planner, the Sensory Subsystem will be operating at a high level in the control system hierarchy. Sensory information required to perform fine motions, such as those provided by force sensors or RCC devices, operates at a much lower level in the control system. Integration of these two different types of sensory information is currently a topic of research in the lab.

Linear-scan structured light for vision with depth perception. Depth information is important to any sensory-based computerized manufacturing, and forms an important component of the machine intelligence required for eye-to-hand coordination. Depth is important for the speed control of a manipulator approaching a workpiece. Depth also plays an important role in the methods for scene analysis in robot vision. In the past, most algorithms for scene analysis have required as input a near perfect line drawing representation of the scene; which under most practical conditions is impossible

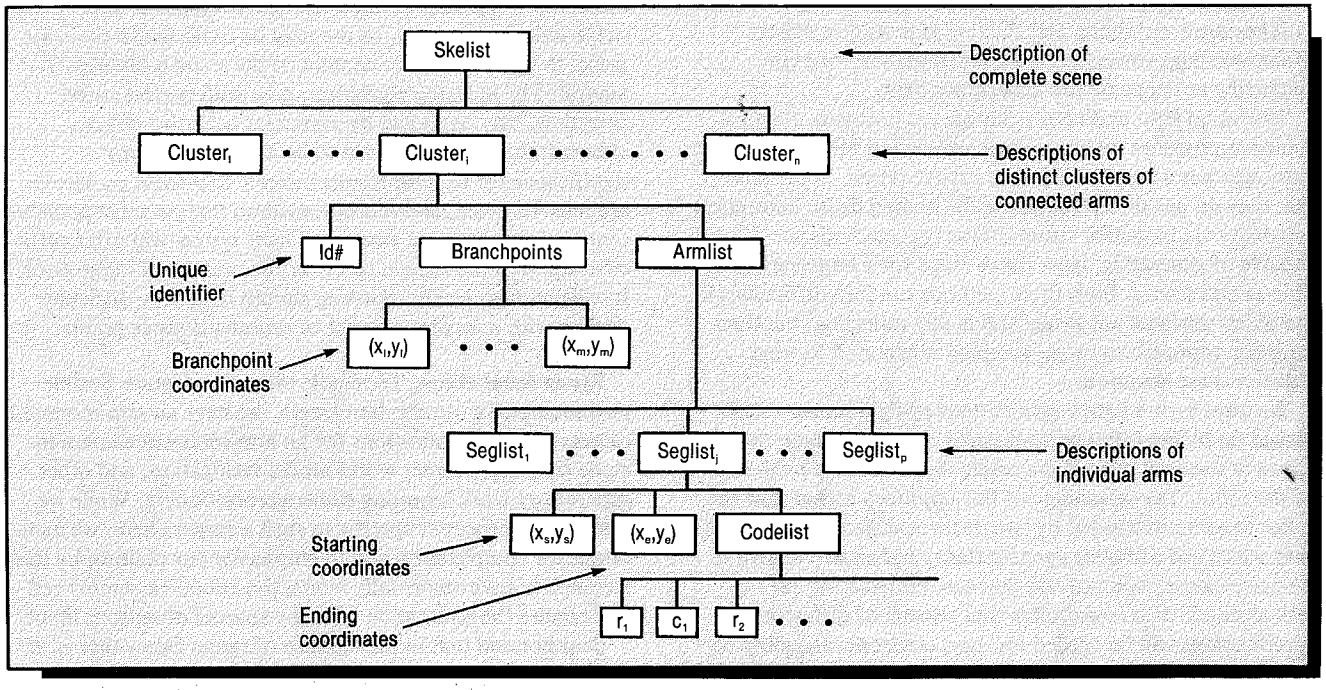


Figure 14. List structure representation of a structural description; r_i = radial value, c_i = chain code entry (N, S, E, W, etc.).

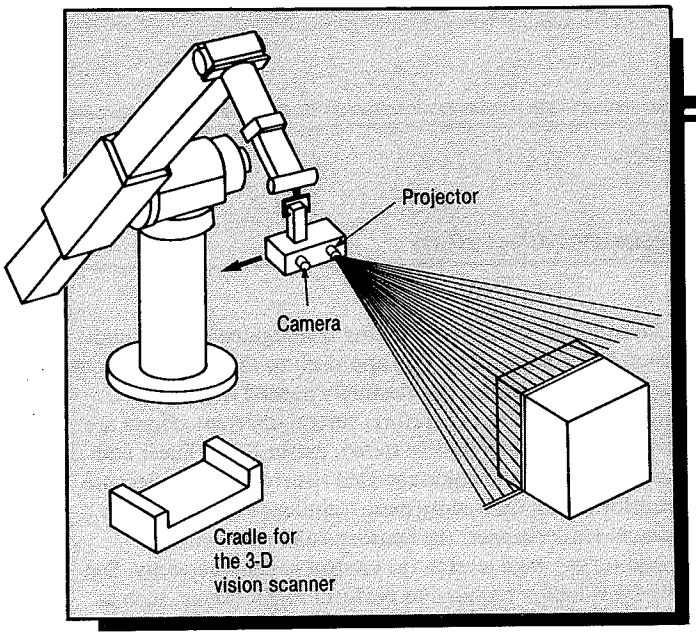


Figure 15. A robot performing a linear structured-light scan.

to obtain from reflectance images. [Under most conditions, a line in an image corresponds to an edge in an object formed by an intersection of two faces.] The difficulty with extracting a line representation is that for many images the photometric differences between the adjoining faces of an object may be too small for reliable edge extraction; and for other images the application of edge extraction can lead to excessive noise enhancement causing false lines to appear in the representation. With edge extraction from reflectance images, false lines are also produced by quantization errors, shading and shadows, and by variability in reflectance properties. With depth information, edges are defined as range discontinuities and range-gradient discontinuities; such edges being immune to reflectance variability and shading and shadow effects. Moreover, edge extraction from range data is not noise enhancing as happens with reflectance data.

Structured light and laser scanning are probably the most robust approaches for range data acquisition for 3-D vision. Although not as sophisticated as passive stereo, in the sense that they do not directly simulate the human depth perception capability via binocular fusion, these two methods are capable of generating dense range maps for a large variety of objects and scenes. Both these methods run into difficulties if the object surfaces are glossy and highly reflecting, but then no other approach to range mapping is expected to work either in these situations.

We have built a lightweight structured-light sensor that is picked up by the robot when it needs 3-D vision data; the sensor is then put away in its cradle that is located by the side of the robot. The advantage of this approach is that now the robot can be surrounded by two or three different types of vision sensors of different specifications, and each one used if its specifications will provide the most information for the task at hand. (It is conceivable that sensors of different specifications will be needed for, say, electronic assembly on the one hand and mechanical assembly on the other.)

As depicted by Figure 15, the manipulator, with the sensor in its gripper, makes a linear scan of the scene. For each position of the projected stripe, triangulation formulas can be used to translate the illuminated pixel locations in the camera image into range information (details are presented elsewhere[25]). A real-time peak detector circuit at the back of the camera in the structured-light sensor locates the illuminated pixel in each raster line of the camera image by using an analog comparator that strobes a synchronous counter.

Color-encoded structured light. To accelerate the process of acquiring a structured light range image of a scene, we could project multiple planes of light simultaneously. However, with this approach, the sequence in which the stripe reflections appear in the image plane may not be the same as that projected, particularly for scenes of practical interest. Therefore, a possible ambiguity arises in the identification of the stripes. Various methods have been proposed for identifying the stripes or the elements of an array of projected rays.[29,30] Our term for the process of matching a detected stripe with its position in the projection grid is *indexing*.

A stripe's index is an integer indicating its ordinal placement (say, from left to right) in the complete set of projected light stripes. Once indexing is accomplished, the range at stripe illuminated image points can be calculated by triangulation.

We have demonstrated a system for acquiring the entire range map of a scene in a single color projection, requiring only one color image.[31] This technique substantially increases speed, while reducing the amount of memory needed. It also results in a system having no moving parts, making it inherently more rugged and thus expanding its range of potential applications. As always, there is a cost associated with these improvements. Because the system employs color in the stripe-indexing strategy, its use may be restricted to environments in which the color content of the scene is inherently neutral. The indexing algorithm, even with perfect stripe detection, may be fooled by particular occlusive effects, although this has not proven to be a problem of any significance for us. Our results indicate that these problems are relatively rare, and it is our position that, in most applications, a color-encoded structured light system will offer sufficient speed improvement to offset the occasional range error. Intelligent high-level processing should have little difficulty dealing with a modest number of erroneous range points.

Eye-in-hand vision. To handle tasks that require flexible positioning of a monocular camera, we have an arm-mounted camera. Useful applications for an arm-mounted camera include tracking, current world model verification, and other supervisory work requiring camera repositioning. While we describe no processes specific to such a camera here, we mention it for completeness and to emphasize our philosophy that a complete environmentally interactive robotic assembly cell will require the integration of many sources of sensory input.

Bear in mind that in the absence of range data—that is, if we were limited to reflectance data—an eye-in-hand camera may possess unique advantages over a static overhead camera for drawing inferences about a scene.[1,32] Compared with the

static overhead camera, an eye-in-hand camera does not suffer from parallax errors in the calculation of location and orientation for an object in the scene. Also, for the same accuracy in calculation, an eye-in-hand camera can get by with lower resolution; and, because of lower resolution, it can yield results in a shorter time.

Symbolic stereo from structural descriptions. The primary difficulty with the use of passive stereo is the so-called correspondence problem, which arises when we try to identify corresponding points in two images of a scene. This problem has been pursued by many researchers, too numerous to be cited individually; the literature is presented in a survey exposition elsewhere.[11] Most of these techniques operate at the pixel or waveform level with little aid or guidance from any sort of structural representation of the scene content.

For scenes composed of objects that are rather "flattish," (meaning that they are basically 2-D in nature), it is often possible to acquire 3-D vision faster with the structural approach to stereo described below. (3-D vision for objects that are basically 2-D is required for dealing with the overlapping parts problem and positioning of the end-effector in bin-picking.) In this method, structural descriptions are used for solving a stereo correspondence problem, and in seeking to contain the combinatorial explosion, we extract primitives rich in information content. Relationships among the primitives serve as another source of constraints.

We invoke the following constraints for solving the structural stereo problem:

(1) The extraction of structural primitives is such that each primitive in the left image is associated with at most one correspondent in the right image, and vice-versa. This uniqueness constraint applies not only to points, but also to the regions we extract as primitives.
(2) A great deal of redundancy exists between the two images. The correct mapping between the primitives of the two images is most likely to be the one that results in the greatest apparent redundancy between the structural descriptions of the two images.
(3) The primitive descriptions and relations are such that the initial stages in the process to find the interprimitive mapping function rely heavily on the information content of the primitives themselves.

Our goal is to determine an effective mapping function between the two sets of primitives such that the likelihood of correct interprimitive correspondence is maximized. This mapping is 1:1 between subsets of equal cardinality from the two primitive sets.

An information-theoretic interprimitive distance measure. The fundamental concept in information theory is that the amount of information derived from some event, or experiment, is related to the number of degrees of freedom available beforehand, or a reduction in uncertainty gleaned from an observation of the outcome. In this manner, the discrete communication problem is analogous to the structural stereo correspondence problem. We expect the properties of corresponding scene features to be similar. As a channel, we expect the stereo correspondence process to convey relatively little information.

Modeling Correspondence as an Information Channel: We may model the set of n primitives characterized by a set A of attributes in a structural description as an information source consisting of $n \times |A|$ independent sources operating in parallel. We may consider the correspondence problem as an information transmission issue.

Let's look at Figure 16. The single primitive p_i on the left has been extracted from the left image of the scene, while q_j has been taken from the right. If p_i and q_j are, in fact, correctly matched, then the image-to-image distortion in the characteristics of the primitives is represented by a set of channels, one per attribute, mapping symbols from the left primitive into symbols from the right primitive.

The apparent information conveyed in the mapping must be characterized by probabilistic models. We characterize the correspondence process in terms of a set of transition probabilities indicating the conditional probability that a primitive will have value $v_r = a(q_j)$ for attribute a in the right image given that it has value $v_l = a(p_i)$ for the same attribute in the left image.

Interprimitive Distance: The amount of information conveyed by the set of channels in Figure 16 is a measure of the dissimilarity between the two primitives. If we imagine a set of $|A|$ channels established between every pair of primitives deemed to correspond by the mapping function h, then the distance between the primitive sets under the mapping h is the amount of information conveyed by all those $\rho \times |A|$ channels, where ρ is the cardinality of h.

We define the conditional information contributed by an attribute a taking value v_r for q_j in the right image given that it has taken value v_l for p_i in the left as

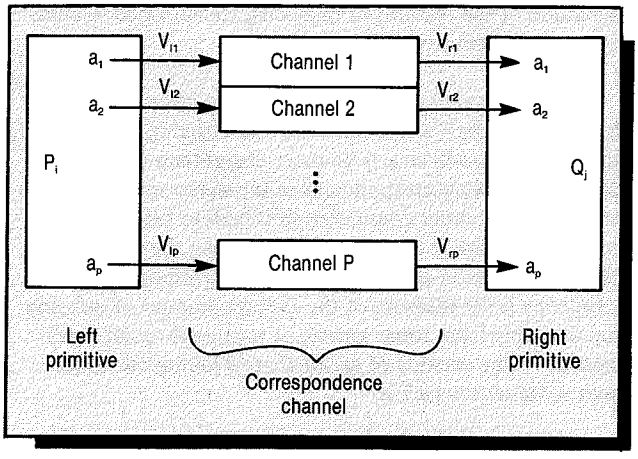


Figure 16. Correspondence as an information channel.

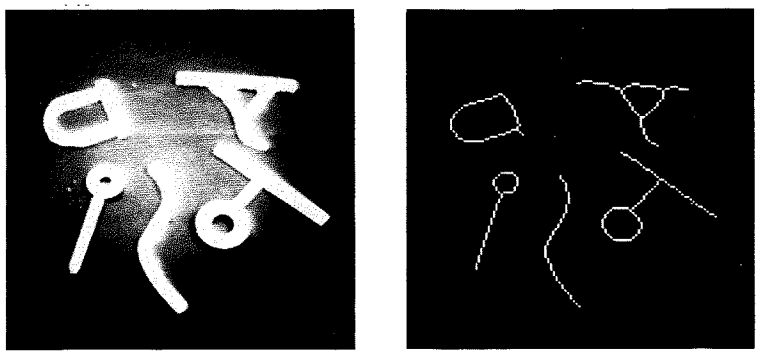


Figure 17. A thinned binary image (a),(b) and part of its structural description (c).

```
(((1 (asp-ratio-rib 15)
     (masp-ratio-rib 15)
     edge-length 71)
     m-orient 82)
     (edge-mass 361)
     (loop-pred t)
     dom-radius 4)
     centroid (37 47)))
(2 (asp-ratio-rib 2)
     (masp-ratio-rib 4)
     (edge-length 18)
     (m-orient 172)
     (edge-mass 126)
     (loop-pred nil)
     (dom-radius 4)
     (centroid (18 76)))
(3 asp-ratio-rib 2)
     (masp-ratio-rib 4)
     (edge-length 18)
     (m-orient 7)
     edge-mass 126)
     (loop-pred nil)
     (dom-radius 4)
     (centroid (20 91)))
. . . )))
((common-end-1 2 ((2 3) (2 5)
 (3 5) (3 4) (3 6) (4 6) (5 6)
 (5 7) (6 7) (8 9) (8 10) (9 10)))
 (common-end-2 2 nil)
 (parallel 2 ((1 11) (3 4) (5 7)
 (6 9) (8 10) (11 12)))
 (orthogonal 2 ((1 14) (2 13) (3 14)
 (4 14) (6 8) (6 10) (8 9) (9 10)))))
```

$$I\left(a(q_j) \mid a(p_i)\right) = \log\left[\frac{1}{p\left(\left\{a(q_j) = v_r\right\} \mid \left\{a(p_i) = v_i\right\}\right)}\right]$$

The total distance between the two primitives p_i and q_j is then defined as the sum of the conditional information contributed by each of its attributes under the mapping h:

$$D_h(p_i,q_j) = \sum_{a \in A} I\left(a(q_j) \mid a(p_i)\right)$$

We then compute the distance between the two primitive sets under the mapping h by summing over all mapped pairs:

$$D_h(P,Q) = \sum_{(i,j) \in h} D_h(p_i,q_j)$$

And our task is one of selecting the interprimitive mapping function h that minimizes $D_h(P,Q)$.

The information-theoretic approach to interprimitive matching should be superior to some more obvious techniques, such as representing each primitive as a vector in attribute space and computing Euclidean distances, because the probabilistics of the situation are considered. Further, for attributes that take on symbolic values, it may be difficult or impossible to order the values in such a manner as to make the Euclidean distance meaningful. In our approach, the distance between two entities is a monotonic function of the likelihood of the features of one being transformed to coincide with the features of the other.

Entropy-based attribute selection. In an information-theoretic approach to the selection and ordering of attributes, some attributes will clearly be more useful than others in identifying likely correspondents in the stereo-matching problem. The issue under consideration is how to best select a subset (ordering) of these attributes yielding the best (quickest) descrimination among the possible correspondents.

Entropy is the measure of the average amount of information available from some source, or passing through some channel.[33] The entropy of an attribute a taking on values from some set $v = \{v_i\}$ is

$$H_a = \sum_i p(v_i) \log\left[\frac{1}{p(v_i)}\right]$$

We are also interested in the conditional entropy of an attribute. The conditional entropy of attribute a taking on values v_i in, say, the right image given corresponding values of v_j in the left is

$$H_a{}^c = \sum_j p(v_j) \sum_i p(v_i \mid v_j) \log\left[\frac{1}{p(v_i \mid v_j)}\right]$$

For us, entropy provides a measure of the average information about a primitive provided by an attribute. Conditional entropy indicates the average amount of uncertainty remaining in the value of that same attribute for corresponding right-image primitives, once a left value is known. Conditional entropy may be considered to be the average a posteriori entropy for the attribute. We define a figure of merit for attributes, q

$$q = \frac{H_a}{H_a{}^c}$$

An attribute exhibiting a large value for q may be expected to outperform one with a small value in discriminating correct from incorrect primitive matches.

There can be problems with q, however. It is possible for an attribute to have a very low a posteriori entropy, yielding a large value for q even if the a priori entropy is also low. As we will see, this is particularly true of the loop-predicate attribute. This attribute has the low a priori entropy of 178 millinats, as determined by our experiments. However, the a posteriori entropy of this attribute, as determined in our experiments, is zero.

To improve on this situation, we define another figure-of-merit, q', which takes values in [0,1]. If we have the following

$$\Delta H = H_a - H_a{}^c$$

Then q' is defined as

$$q' = \frac{\Delta H}{H_a}$$

This figure-of-merit is used for ordering the attributes before graph-theoretic matching. (Further details are presented elsewhere.[22])

Experimental results. There are two sets of experimental results: those used to construct the channel models and characterize the attributes and relations, and those in which the resulting models were used in solving a symbolic stereo correspondence problem.

Building the Models: To develop these ideas and provide a testbed for the procedure, we took 25 pairs of images from an overhead stereo pair of cameras. Figure 17 illustrates one of the images, its thinned version, and part of its structural description.

Channel matrices are stored as distance measures. Some entries in the channel matrices are infinite, establishing some attribute value pairings as impossible for corresponding primitives. Primitive pairings having infinite distance are rejected. Figure 18 is an example of our channel matrices.

The figures of merit, q and q', were computed for each of several attributes. Values of q and q' for these attributes are listed in Table 2. Entropy values are expressed in millinats.

To avoid problems with threshold sensitivities in consistency checks, two versions of each relational predicate involving a threshold were written. The conservative version was applied to the left image, while the liberal version was applied to the right. This provided a bias against the rejection of pairings for relational violations because the composition of the left image relations with the mapping function is required to be a subset of the right image relations. That is, we check for homomorphism.

Matching Results: Matching experiments were run on 13 image pairs using loop-pred, m-orient, asp-ratio-rib, dom-radius, edge-length, edge-mass, and straight-ratio as the inter-primitive mapping criteria. The primitives consisted only of skeletal arms, that is, edges in the graph. Four binary relational predicates were computed and invoked, including

- Common-end-1: One endpoint in common
- Common-end-2: Both endpoints in common
- Parallel: Orientations approximately the same
- Orthogonal: Orientations differ by about 90 degrees

The results are summarized in Table 3. A rank of 1 indicates that the correct mapping function was assigned the lowest distance value of all those found, while a rank of infinity indicates that the correct mapping function was not found. Although a mapping function was nearly always found that was almost completely correct, we made no attempt to tabulate those results. A rank of 1T indicates that more than one mapping was found with the same (low) distance.

Image pairs for which the process failed to isolate the correct mapping function were poorly segmented, leaving only broken pieces available to the matcher. Most of the images contained some segmentation noise. In modest amounts this poses no problem to the matcher, the noise blobs in one image being mapped (correctly) to nil in the other.

Let us now look at the Motion Controller and the Supervisor in more depth.

	Edge-length									
Value	5	7	10	14	18	24	32	40	53	71
5	2.2	0.59	2.2	2.2	2.2	∞	∞	∞	∞	∞
7	2.48	0.47	1.57	2.48	∞	∞	∞	∞	∞	∞
10	∞	1.56	1.56	0.64	2.94	∞	∞	∞	∞	∞
14	∞	∞	2.0	0.72	1.05	3.61	∞	∞	∞	∞
18	∞	∞	∞	1.13	0.84	1.53	3.61	∞	∞	∞
24	∞	∞	∞	∞	3.37	0.23	1.76	∞	∞	∞
32	∞	∞	∞	∞	∞	1.11	0.92	1.32	∞	∞
40	∞	∞	∞	∞	∞	∞	1.54	0.69	1.25	∞
53	∞	∞	∞	∞	∞	∞	∞	2.3	0.16	3.0
71	∞	∞	∞	∞	∞	∞	∞	∞	0.29	1.39

Figure 18. Channel matrix example; distances are in nats.

Table 2. Tabulated values of the figures of merit q and q' for several primitive attributes.

Attribute	H_a	H_a^c	q	ΔH	q'
M-orient	2438	617	3.95	1821	0.75
Loop-pred	178	0	∞	178	1.00
Edge-mass	2032	740	2.75	1292	0.64
Edge-length	2123	956	2.22	1167	0.55
Eight-ratio	1215	706	1.72	509	0.42
Dom-radius	1512	905	1.67	607	0.40
Asp-ratio-rib	1796	1108	1.62	688	0.38
Masp-ratio-rib	1855	1157	1.60	698	0.38
Straight-ratio	1274	814	1.57	460	0.36

Table 3. Tabulated matching results for 13 image pairs; IP = image pair, LP = no. of left-image primitives, MPS-U = no. of mappings without relational constraints, R = ranking accorded the correct mapping, and MPS-C = no. of mappings with relational constraints.

Matching Results					
IP	LP	MPS-U	R	MPS-C	R
1	8	12	1	2	1
2	22	4	1T	0	∞
3	6	4	1	4	1
4	10	2	1	2	1
5	3	1	1	1	1
6	3	1	1	1	1
7	10	6	1	2	1
8	11	32	1T	8	1T
9	9	29	8	13	∞
10	14	28	1	2	1
11	14	136	1T	4	1
12	15	48	1T	2	1
13	14	20	1T	4	1

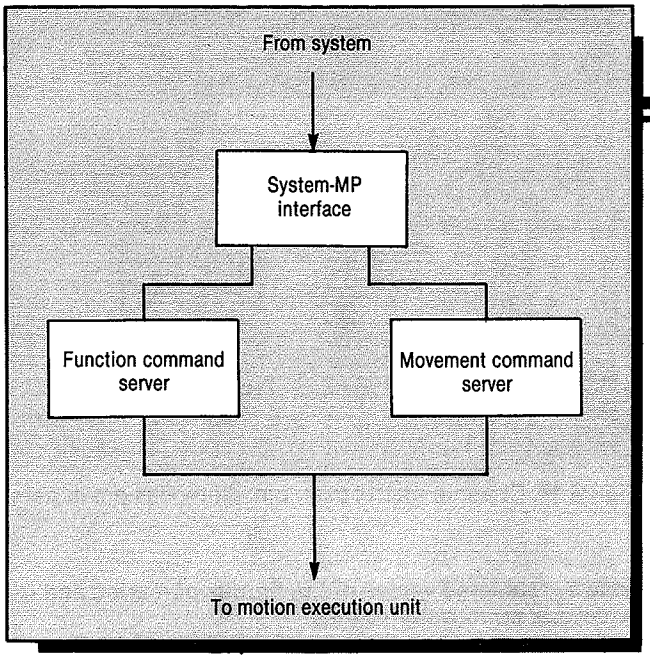


Figure 19. The Motion Planner. The Motion Planner consists of two message servers and a system interface.

The Motion Controller

The Motion Controller controls the programmable actuators in the assembly cell. It consists of a motion planner (MP), which handles move verification and path planning functions, and a motion execution unit (MEU) for each actuator, which executes the commands derived by the MP. Currently, our cell consists of a Cincinnati Milacron T^3-726 electric robot with its associated controller and a pneumatic gripper controlled by a VAX 11/780 minicomputer.

The motion planner. Figure 19 illustrates the structure of the MP. The system-MP interface processes the incoming requests and invokes the appropriate server. Two types of servers currently exist, one for motion requests and another for function requests. The motion request server handles motion planning and move verification, while the function server is in charge of gripper operation, tool transform and segment velocity updating. These servers take an incoming request, perform the appropriate action, and, if necessary, send an action request to the MEU.

The MP maintains a model of the state of the manipulator and its associated hardware. This model contains the current position of the arm, the tool transform in current use, the velocity selected for the current path segment, and the state of the gripper. These parameters are transmitted to the Current World Model whenever the manipulator is in an idle state. When motion begins, a signal is sent to the CWM to lock requests for information about the manipulator state until it has stabilized.

Movement commands. The MOVE function is in charge of servicing movement commands. Figure 20 provides an overview of its function. All manipulator movements are straight line Cartesian motions with respect to the tool center point. The tool center point is defined as

$$\text{tool center point} = T^6\ \text{TOOL}$$

where T^6 is the position of the center of the manipulator's tool-mounting plate with respect to its base, and TOOL represents the transformation from the tool-mounting plate to the tool tip. This arrangement allows motions to be specified by three sets of parameters:

- Goal position
- TOOL transform
- Segment velocity

At the present time, force control has not been implemented. To add this capability to the Motion Controller, the movement command server's capabilities would be augmented. The force constraints will be expressed in a form analogous to those in AL[34] or RCCL.[35]

Path Planning: After a move command is issued, a path planner is invoked to ensure that the manipulator doesn't collide with any fixed obstacles in its workspace. At the present time, no provision is made for automatically avoiding movable objects in the workspace. The path planner takes a starting and ending position, and produces a list of straight line motions that will guide the manipulator between them. The orientation of the manipulator is first set to that of the goal position. The required translation then occurs.

A recursive procedure is used for the translational planning step to ensure that no motion commands are generated or executed that would cause a collision between the robot arm and base (Figure 21). Assume the manipulator is at position A, and is requested to move to position B. Straight-line motion would send the manipulator through the base. To avoid this problem, two transition points are defined. These provide safe points near the corners of the base forbidden space. The path planner examines the line joining the starting and ending points of the motion. If this line intersects the forbidden region surrounding the manipulator base, the transition point nearest the goal is located, and a move from there to the endpoint is appended to the end of the move list. The endpoint is now set to the selected transition point, and the same procedure is applied to this new path. This process continues until no path segment intersects the forbidden region. The move list is now passed to the move verification routine.

Error Detection and Handling: Two types of errors are detected: kinematic errors and world errors. Kinematic errors result whenever the manipulator is requested to move to a position that is kinematically impossible. These errors are detected by using the inverse kinematics of the manipulator to provide the joint angles corresponding to the position in question. Each joint angle is checked against its allowable range. If any are out of limit, the supervisor is notified and presented sufficient information to permit a diagnosis.

World errors occur whenever the manipulator is requested to move to a kinematically valid position that conflicts with some permanent obstacle in the workspace. Currently, the only such obstacle is the work table surrounding the

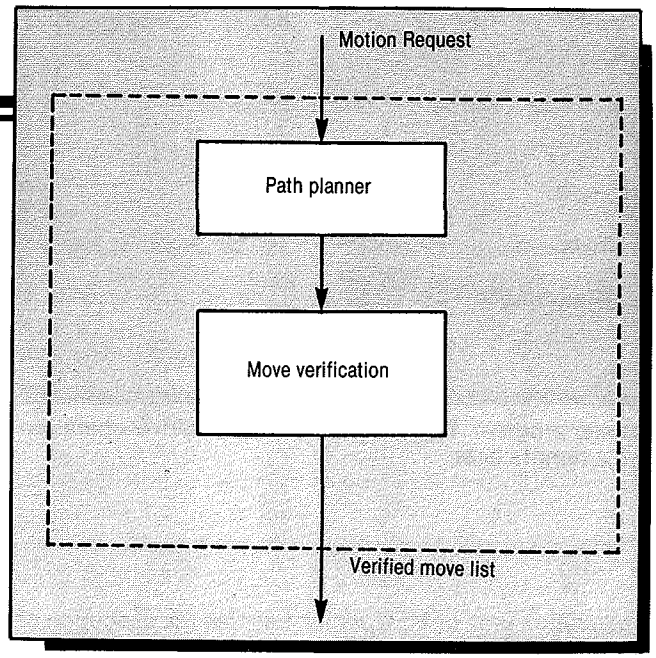


Figure 20. The MOVE command, which is the prime vehicle for serving motion requests.

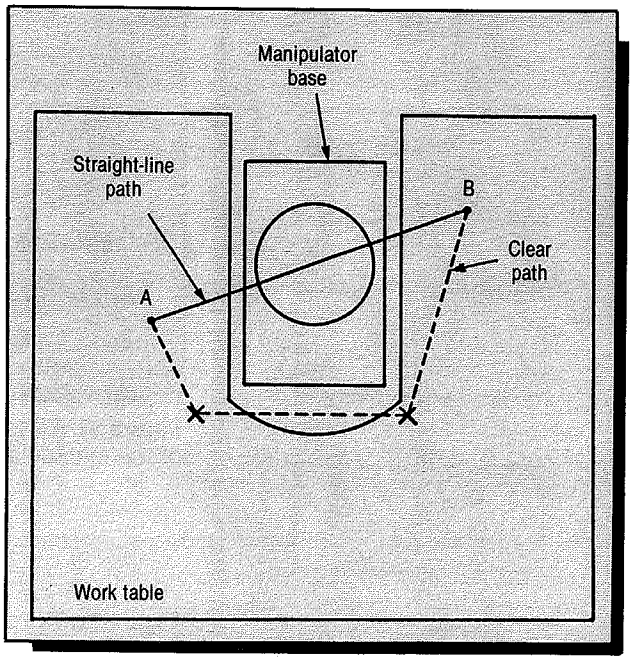


Figure 21. To avoid moving the end effector through the manipulator, a recursive path planner is used.

manipulator. Any move that would put some portion of the manipulator or its gripper through the table is detected, and the supervisor is notified of the problem.

To provide flexibility in error handling, a user modifiable error handler is invoked when either error condition occurs. The default version aborts execution and returns an error indication to the calling procedure.

Function commands. The function commands handle any operation not involving manipulator movement. Currently, the set of function commands supported includes facilities for changing the TOOL transform and translation velocity, activating various switches to control external functions, and controlling the gripper. In addition, several administrative functions are available. These allow the supervisor to establish and reset the MP-to-MEU and MEU-to-Motion Controller communication links.

A model containing the current state of each of the sensors is maintained. When a function request is issued, the MP-level model is updated, and a function request is issued to the MEU. The MEU then carries out execution of the request.

Administrative functions require more effort. The major administrative function is controlling the MEU communication channel. A command, *setup-link*, is responsible for restoring the MP and MEU manipulator models to a known state and verifying that the communication links are operating correctly. First, communication with the MEU is established. Next, the MP waits until the MEU has established a link to the robot controller. The model position and tool transforms are reset to values obtained from the robot controller via the MEU. The gripper state is set to an arbitrary value since there are no facilities for interrogating the state of the controller I/O lines.

The motion execution unit. The MEU provides a uniform low-level interface to the manipulator. Ideally, the MEU would be a part of the manipulator controller, but most current robot controllers do not support user programming at the level required for its implementation.

The MEU consists of three subcomponents (Figure 22): the MP-to-MEU communication interface, the controller model, and the MEU-to-robot controller communication interface.

MP-to-MEU communication interface. The MP-to-MEU communication interface processes the messages sent to, and received from, the Motion Planner. In normal operation, the MEU executes a loop waiting for a message from the MP. Upon receipt, appropriate action is taken according to message content. Three message classes are supported: function, administration, and motion. Messages in the function subclass control the operation of the various peripheral devices on the manipulator, such as grippers and programmable fixturing. Administration messages provide control of the MEU-to-controller communication link and manage the controller's tool and velocity tables. The motion messages specify the robot's movements.

The controller model. This portion of the MEU determines what actions are required to fulfill the incoming request. It maintains a present-state/next-state model of all pertinent manipulator parameters. To avoid unnecessary effort, actions are initiated only if the next state of a parameter differs from its present state.

The robot controller incorporates two tables that are of interest at this level: the velocity table and the tool table. The velocity table contains 15 preset velocities that may be specified on each move request, evenly spaced from 3.0 to 45.0 ips. Velocity is selected by choosing the table entry closest to the requested velocity. The TOOL transform used in computing the tool center point is stored in the tool table.

MEU-to-robot controller communication interface. The language available on the T^3-726 is a teach-by-showing-

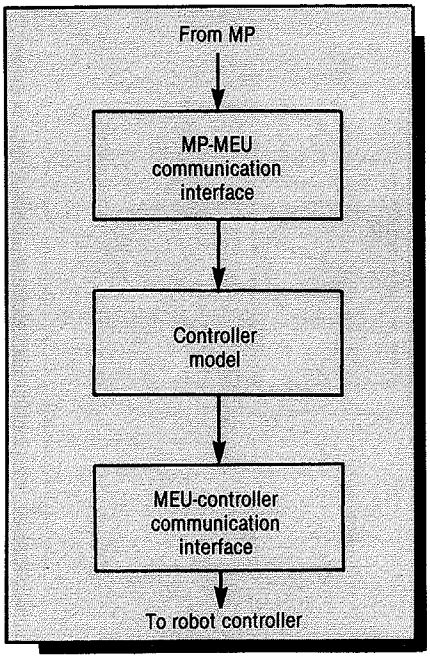


Figure 22. The motion execution unit, or MEU, which handles the execution of movement-related requests; MP = Motion Planner.

```
(1) Controller in a/c mode under host control
(2) Replace point command issued by host. Point in I/O sequence is set to
    perform desired operation.
(3) Host releases from a/c mode.
(4) Controller performs I/O sequence.
(5) Desired I/O operation is performed.
(6) Main cycle is reentered. Control is returned to host by entering a/c
    mode.
```

Figure 23. A short controller program. The controller program allows operation of the manipulator from a remote host.

oriented system.[36] This language allows the user to enter a program by moving the manipulator to a desired location and storing the corresponding joint coordinates. In addition, the programmer may specify a function to be performed at each point. Primitive subroutines and user-accessible variables are also available.

Recently, a REMOTE function was added to this language.[37] This function allows the controller to communicate with a remote host and execute commands received from it. The command language supporting this function is called the Remote Link Level Interface (RLLI). RLLI is a packet-oriented system that communicates with the controller over a serial channel. This interface was originally conceived as a means for storing and modifying programs offline. Nonetheless, a set of facilities is available to control the major functions of the manipulator over the remote link. Commands for moving the arm to a specific position at an explicit velocity and for interrogating the controller for position are available. By programming the controller to execute a tight loop consisting of a REMOTE command and a PERFORM (subroutine call) statement, the manipulator may be controlled from a remote host. This program is provided in Figure 23.

The PERFORM statement is required because no direct facilities for controlling I/O ports in the controller exist. To work around this limitation, the program modification facilities provided by the RLLI were brought into play. Whenever the state of an I/O line is to be modified, the host sends a request to the controller to modify a point in the subroutine. By changing this point to a function that controls the correct I/O line, and leaving the REMOTE mode, it is possible to control any of the I/O lines.

The Supervisor

The Supervisor orchestrates the operation of the other modules. In its simplest (and current) form, it is a human operator to oversee the remaining system functions. At the other extreme, the supervisor can plan, execute, and monitor all aspects of cell operation. Development of a such a system is a long-range goal of our program.

An important component of the supervisor is the task planner. Several planning systems have been proposed in the literature. These include the Strips family of planners,[38,39] Build,[40] and Pulp1.[41] As input, they take a description of the current state of the world, and a desired goal state. As output, they produce a sequence of operations that will transform the current state into the goal state. None of these systems allows sensory feedback to guide the planner—an ability essential to a flexible assembly cell.

For the incorporation of sensory feedback in automatic planning, we have developed a special-purpose language, FProlog, in the Robot Vision Laboratory.[42] FProlog, an extension of the Prolog language[43] and implemented in Lisp, integrates logic and functional programming in a package that allows the automatic generation of task plans from knowledge of the current state, the desired goal state, and a rule set. FProlog has been used to generate task plans and automatically invoke those plans with the robot manipulator to solve the blocks world problem. Extension to other problem domains will follow. FProlog clauses are defined to access the motion control functions, retaining the advantages of Prolog as a resolution theorem prover. Work is underway to explore the use of this language in planning sensor-guided assembly.

Experimental results in sensory-guided parts mating

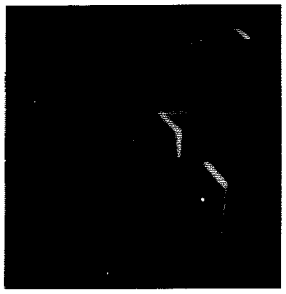
The aim of this section is not to give an experimental corroboration of the overall system described here. Rather, it is to show some basic, yet successful, experimental studies we have conducted in mating parts from random locations and orientations. Our aim was to learn about the low-level sensory and manipulation issues involved so that they could then

be incorporated in the overall system. Note that if we remove the requirement that parts be allowed in any position and orientation (within reasonable limits, of course), the resulting simpler problem has a well-known solution.[44] When the position and orientation of the parts are random, the task becomes difficult without proper environmental feedback.

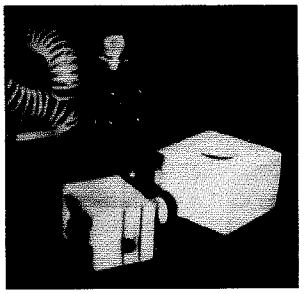
In this section, we will show how 3-D vision feedback is used to meet our goal. In the following experiment, the sensing consists of a portable structured light unit that is picked up by the robot for scanning the scene when it needs the range data. The processing consists of transforming the 3-D vision data into what we call an altitude map, segmenting the objects from the background, discriminating between the male and the female components, computing the correct approach path for the gripper so as to allow mating, and finally mating the two pieces.

The experiment described was run on a Cincinnati-Milacron T^3-726 electric robot coupled to a VAX 11/780 computer. Figure 24a shows a scene consisting of two pieces, a male and a female, located at random positions and oriented randomly in the work area for the robot. Note that the diameter of the peg on the male part and the diameter of the hole on the female part differ by approximately ¼ inch—in other words, the fit is not tight between the peg and the hole. The allowable tolerances on the dimensions depend on the accuracy of the vision data, which, in our case, is a function of the number of light stripes used to illuminate the scene. The dimensional difference we have used gives us a sufficient margin for error for the 80 stripes we typically use in our experiments. (For close fitting parts, it is necessary to use some form of force feedback in conjunction with vision.) A range map is acquired by directing the robot to pick up a structured-light sensor and then to scan the scene with it (Figure 24b). This range map is then transformed to an altitude map, which is constructed by retaining for each x,y location on the work table the highest vertical coordinate as obtained by 3-D vision. In the altitude map, the male and female objects are then segmented by a region-growing algorithm and identified. The orientation and position of each object are then computed. A stored object model helps the computer determine the holdsites for the gripper on the male part. (Both the object model and the mating strategy are a part of the Global Knowledge Base.) As shown in Figures 24c through 24i, we are able to successfully mate the two parts, and to do so, the robot automatically reorients the male part to make mating kinematically possible. First, the male piece is picked up and reoriented so that the cylindrical portion is pointed away from the manipulator's base (Figures 24c and 24d). It is then reacquired with the gripper forming a 135-degree angle with the vertical (Figure 24e). The male part is then picked up and rotated 180 degrees about the gripper approach vector (Figures 24f and 24g). The result is that the peg points straight down. The peg is then centered over the hole in the female part (Figure 24h) and mated (Figure 24i).

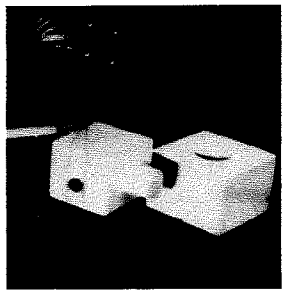
(a) Male and female parts are in random positions with random orientations.



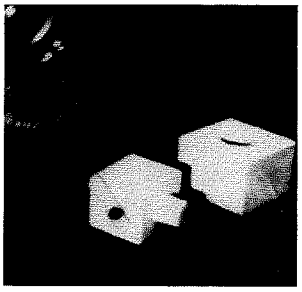
(b) Scene is scanned with structured-light unit for the acquisition of 3-D vision data.



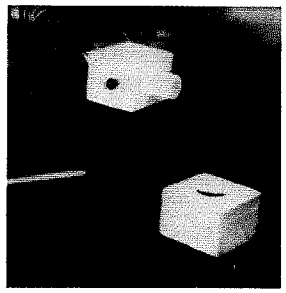
(c) Robot has correctly calculated grip points on male piece and picked it up.



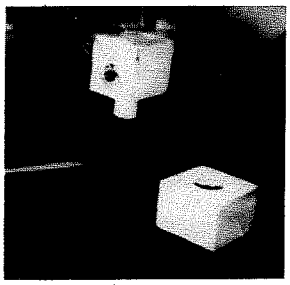
(d) Reorientation of male piece for mating.



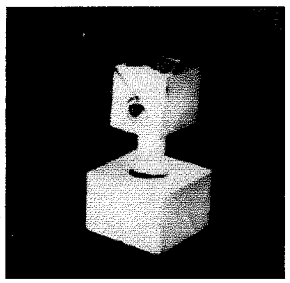
(e) Reoriented male piece is put back on work table and freed gripper rotated by 135° about vertical axis.



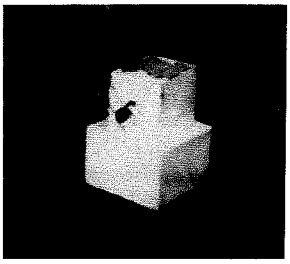
(f) Male piece is picked up by reoriented gripper.



(g) Male piece is rotated by 180° about gripper-approach vector.



(h) Male piece is centered over hole in female piece.



(i) Mating is completed.

Figure 24. Male and female objects used in the parts-mating experiment (a), an example of a typical light stripe projection from the data collection phase, and views after the objects have been identified and the peg inserted in the hole (c-i).

We have discussed the ongoing development of an integrated, knowledge-based robotic assembly cell that uses sensory feedback for true flexibility. Although much work remains to be done on some components of the system, we have reported major progress in the vision and motion planning modules. Items of high priority at this stage include the development of a master template for the top levels of both the Global Knowledge Base and the Current World Model. E

Acknowledgments

This research was jointly supported by the National Science Foundation and the Computer Integrated Design, Manufacturing, and Automation Center (CIDMAC) at Purdue University. We are grateful to Larry Hollingshead, director of the Artificial Intelligence Laboratory at Cincinnati-Milacron, for the gift of the T^3-726 robot used in the experiments described here. This work would not have been possible without the robot-VAX 11/780 communication software developed by Kirk Smith; we owe him much appreciation for the effort. We also thank Howard Moraff, National Science Foundation, for many useful discussions on sensory and systems integration for robots.

References

1. A. C. Kak and J. S. Albus, "Sensors for Intelligent Robots," in *Handbook of Industrial Robotics*, S. Nof, ed., John Wiley & Sons, New York, 1985, pp. 214-230.
2. G. Gleason and G. Agin, "A Modular Vision System for Sensor-Controlled Manipulation and Inspection," *Proc. Ninth Symp. Industrial Robots*, Mar. 1979.
3. J. McMillen and O. R. Mitchell, "An Intelligent Robot Vision System for Research Application," *Proc. IEEE Int'l Conf. Robotics and Automation*, 1985, pp. 40-45.
4. R. Alami, "NNS: A LISP-Based Environment for the Integration and Operating of Complex Robotics Systems," *Proc. IEEE Int'l Conf. Robotics and Automation*, 1984, pp. 1984, pp. 349-353.
5. I. Lee and S. M. Goldwasser, "A Distributed Testbed for Active Sensory Processing," *Proc. IEEE Int'l Conf. Robotics and Automation*, 1984, pp. 925-930.
6. K. G. Shin and M. E. Epstein, "Communication Primitives for a Distributed Multi-Robot System," *Proc IEEE Int'l Conf. Robotics and Automation*, 1985, pp. 910-917.
7. K. L. Boyer, R. J. Safranek and A. C. Kak, "A Knowledge Based Robotic Vision System," *Proc. First IEEE Conf. Artificial Intelligence Applications*, 1984, pp. 45-50.
8. H. S. Stone et al., *Introduction to Computer Architecture*, H. Stone, ed., Science Research Assoc., Chicago, 1975.
9. G. Agin, "Vision Systems," in *Handbook of Industrial Robots*, S. Nof, ed., John Wiley & Sons, New York, 1985, pp. 231-261.
10. R. C. Bolles, R. Horaud and M. J. Hannah, "3DPO: A Three-Dimensional Part Orientation System," *Proc. Eighth Int'l Joint Conf. Artificial Intelligence*, Aug. 1983, pp. 1116- 1120.
11. A. C. Kak, "Depth Perception for Robots," in *Handbook of Industrial Robotics*, S. Nof, ed., John Wiley & Sons, New York, 1985, pp. 272-319.
12. H. S. Yang and A. C. Kak, "Determination of Identity, Position, and Orientation of the Topmost Object in a Pile," *Proc. Third IEEE Computer Society Workshop on Computer Vision, Representation and Control*, 1985, pp. 38-48.
13. W. E. L. Grimson, "Computational Experiments with a Feature Based Stereo Algorithm," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 1, Jan. 1985, pp. 17-34.
14. W. E. L. Grimson and T. Lozano-Perez, "Model-Based Recognition and Localization from Sparse Range or Tactile Data," *Int'l J. Robotics Research*, Vol. 3, No. 3, Fall 1984, pp. 3-35.
15. W. D. Hillis, "A High-Resolution Imaging Touch Sensor," *Int'l J. Robotics Research*, Vol. 1, No. 2, 1982, pp. 33-44.
16. M. Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P. Winston, ed., McGraw-Hill, New York, 1975.
17. R. B. Roberts and I. P. Goldstein, "The FRL Manual,," MIT AI Memo, Mass. Inst. of Technology, Cambridge, Sept. 1977.
18. D. G. Bobrow and T. Winograd, "An Overview of KRL, A Knowledge Representation Language," *Cognitive Science*, Vol. 1, No. 1, Jan. 1977.
19. M. Deering, J. Faletti, and R. Wilensky, *Using the PEARL AI Package*, ECL Tech. Report, Univ. of California, Berkeley. 1982.
20. L. G. Shapiro and R. M. Haralick, "Structural Descriptions and Inexact Matching," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-3, 1981, pp. 504-519.
21. T. Sakai, M. Nagao and H. Matsushima, "Extraction of Invariant Picture Sub-Structures by Computer," *Computer Graphics and Image Processing*, 1972, pp. 81-96.
22. K. L. Boyer and A. C. Kak, "Symbolic Stereo from Structural Descriptions," *Proc. Second IEEE Conf. Artificial Intelligence Applications*, 1985, pp. 82-87.
23. B. K. P. Horn, "Extended Gaussian Images," *Proc. IEEE*, Dec. 1984, pp. 1671-1686.
24. K. L. Boyer and A. C. Kak, "Symbolic Processing for Pile Analysis in Robot Vision" submitted for publication.
25. H. S. Yang and A. C. Kak, *Detection of Edges in Range Data*, Tech. Report TR-EE 84-18, Purdue Univ., West Lafayette, Ind., 1984.
26. H. S. Yang, K. L. Boyer and A. C. Kak, "Range Data Extraction and Interpretation by Structured Light," *Proc. First IEEE Conf. Artificial Intelligence Applications*, 1984, pp. 199-205.
27. R. A. Brooks, R. Greiner and T. O. Binford, "The ACRONYM Model Based Vision System," *Proc. Sixth Int'l Joint Conf. Artificial Intelligence*, 1979, pp. 105-113.
28. R. A. Brooks, "Symbolic Reasoning Among 3-Dimensional Models and 2-Dimensional Images," *Artificial Intelligence*, Vol. 17, 1981, pp. 285-349.
29. M. Minou, *Theoretical and Experimental Studies on Basic Relations Between Real World Pictorial Patterns and Their Generating Constraints*, PhD dissertation, Dept. of Information Science, Kyoto Univ., Japan, 1982.
30. J. Posdamer and M. Altschuler, "Surface Measurement by Space Encoded Projected Beam Systems," *Computer Graphics and Image Processing*, Vol. 18, 1982, pp. 1-17.
31. K. L. Boyer and A. C. Kak, "Color-Encoded Structured Light for Rapid, Range Map Computation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, 1986 (to be published).

32. C. Loughlin, "Eye-in-Hand Robot Vision Scores Over Fixed Camera," *Sensor Review*, Vol. 3, 1983, pp. 23-26.
33. C. Shannon, "A Mathematical Theory of Communication," *Bell Systems Tech. J.*, Vol. 27, July-Oct. 1948, pp. 379-423 and 623-656.
34. R. A. Finkel et al., *AL: A Programming System for Automation*, Tech. Report AIM-177, Artificial Intelligence Lab., Stanford Univ., Calif., Nov. 1974.
35. V. Haywood, *RCCL User's Manual: Version 1.0*, Tech. Report, TR-EE 83-46, Purdue Univ., West Lafayette, Ind., Oct. 1983.
36. *Operating Manual for Cincinnati Milacron T^3-726, T^3-746, and T^3-776 Industrial Robot with Acramatic Version 4.0 Robot Control*, Pub. 16-IR-82121-1, Cincinnati Milacron Marketing Co., Ohio, 1982.
37. *Supplemental Communication Manual for Cincinnati Milacron Acramatic Version 4.0 Robot Control*, Pub. 10-IR- 8332-S, Cincinnati Milacron Marketing Co., Ohio, 1984.
38. R. E. Fikes, P. E. Hart and N. J. Nillson, "Some New Directions in Robot Planning," *Machine Intelligence*, Vol. 3, 1972, pp. 405-430.
39. R. E. Fikes, P. E. Hart and N. J. Nillson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, Vol. 3, 1972, pp. 251-288.
40. S. E. Fahlman, "A Planning System for Robot Construction Tasks," *Artificial Intelligence*, Vol. 5, 1974, pp. 1-49.
41. S. Tangwongsan and K. S. Fu, "An Application of Learning to Robotic Planning," *Int'l J. Computer and Information Sciences*, Vol. 8, No. 4, 1979, pp. 303-333.
42. S. A. Hutchinson and A. C. Kak, "FProlog: A Language To Integrate Logic and Functional Programming," *Proc. IEEE Int'l Conf. Robotics and Automation*, 1986 (to be published).
43. W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, Springer-Verlag, Berlin, 1981.
44. H. Inoue, *Force Feedback in Precise Assembly Tasks*, Tech. Report AIM-308, Artificial Intelligence Lab., Mass. Inst. of Technology, Cambridge, Aug. 1974.

Avinash C. Kak is a professor of electrical engineering at Purdue University, where his current research includes programming paradigms for reasoning, especially in the domain of computer vision, and sensors and sensory integration for robots. He has coauthored *Digital Picture Processing* (Academic Press, 1982) and *Principles of Computerized Tomographic Imaging* (IEEE Press, 1986). He is an Associate Editor of *Computer Vision, Graphics and Image Processing*.

Kim L. Boyer is currently working on a PhD in electrical engineering at Purdue University, where he has been since 1983 working in the area of robot vision. Prior to 1983, he was with Bell Laboratories, Holmdel, N.J., where he helped develop the architecture for a 6:1 speech interpolation system. There his work on extended delay echo cancelers led to a US patent.

Chien-Huei Chen is a research assistant with the Robot Vision Laboratory at Purdue University, where he works on 3-D vision and machine learning projects. He received an MS in electrical engineering from the State University of New York at Stony Brook in 1984.

Robert J. Safranek is currently working toward a PhD in electrical engineering at Purdue University. He has been a research assistant with the Robot Vision Laboratory since 1982, where he studies robotics, image processing, speech processing, artificial intelligence, and parallel and distributed architectures for AI and signal processing.

Hyun S. Yang is with the Robot Vision Laboratory at Purdue University, where he is working towards a PhD in electrical engineering. His research interests include image processing, pattern recognition, robot vision, and artificial intelligence. Yang received an MSEE from Purdue in 1983.

The authors can be reached at Robot Vision Laboratory, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.