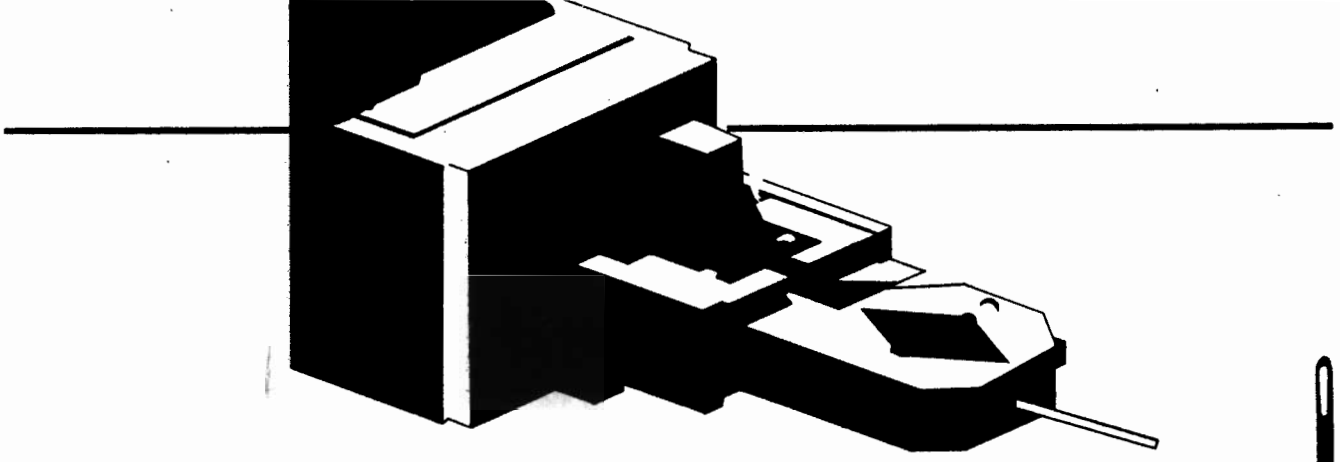


Prediction-Based Vision for Robot Control

reprinted from
COMPUTER
magazine

Michael O. Shneier, Ronald Lumia, and Martin Herman
National Bureau of Standards

This sensing and control system employs a feedback loop to bring a robot's internal representation of its environment into registration with the real world.

The domain of robot sensing has much more structure than that of general sensing. A robot sensing system must operate within time and accuracy limits usually mandated by the application. It commonly does this by precomputing as much information as possible about the robot's environment and the objects in it, and storing this information as a model of the world. In most cases, this knowledge corresponds to statistical or structural methods of identifying objects in images, but is encoded in such a way as to be useful only for recognition or object location.[1]

As robot tasks become more complicated, this approach becomes less viable. A more general approach to modeling is required when task and path planning must be effected at runtime rather than fixed beforehand. And when unknown objects must be handled or when the environment becomes too complicated, the simple methods break down. As more flexibility is required of the system, more generality is required of the models. Fortunately, in industrial robotics environments a good source of models is usually available—the computer-aided design files used to define the objects. A designer can take advantage of this geometrically complete information to substantially enhance the capabilities of a robot sensing system.

In addition to modeling the geometry of objects, a sensing system must usually model the environment surrounding the objects and account for the many instances of each object that might appear in the world and the changes that might be made to each of them. This makes representation and modeling substantially more than a simple description of the objects. Path planning needs an explicit representation of the space surrounding objects, to make it possible to compute optimal motion trajectories. When objects can change during a task either by being machined or by being joined to one another in assemblies, it becomes necessary to treat descriptions as functions of time as well as of space.

At the National Bureau of Standards, we have developed and implemented a sensing and path planning system for a robot meant to operate in a metal machining shop (Figure 1). This system has many of the properties deemed desirable for a robot working in a complex environment. The sensing part of the system consists of a sensing module and a representation module. The sensing module operates the sensors, performs feature extraction and matching, and presents information about

0018-9162/87/0800-0046$01.00©1987IEEE

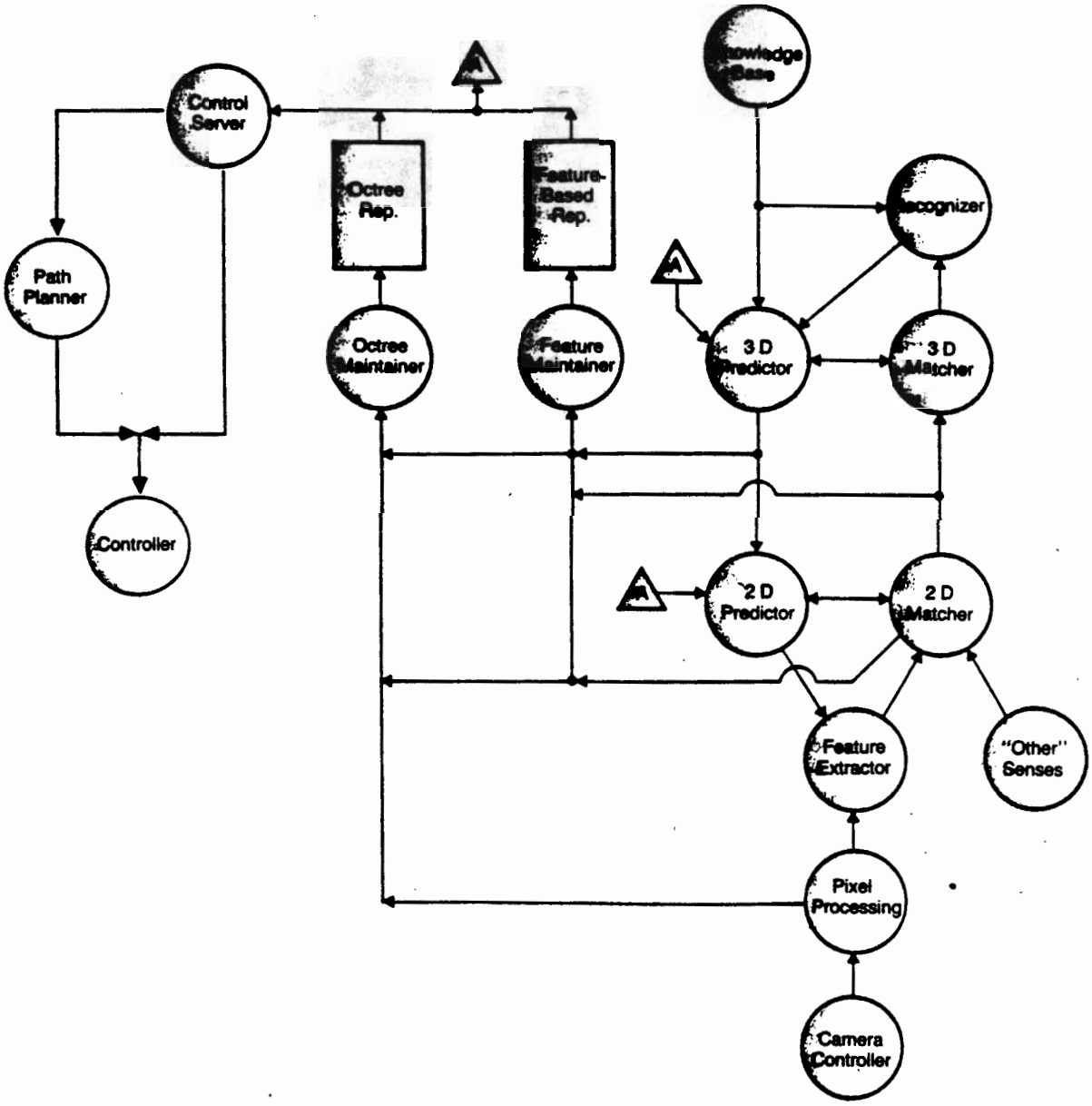


Figure 1. Architecture of the robot sensing and path planning system.

object identities and positions to the representation module. The representation module is responsible for incorporating the sensed data into an internal representation of the world, for maintaining consistency in the internal representation, and for making predictions about the world that are used to guide the sensing. The path planning part of the system plans collision-free paths, and the control system executes these as well as other (predefined) paths.

A special feature of our approach is that the planning and control systems are decoupled from the sensors, with the internal representation acting as the interface. This means that the sensors must be able to keep the internal representation in registration with the world and must accomplish this task fast enough to maintain stability in the control algorithms. This approach has the advantages that the control system does not have to wait for the sensors to respond, does not have to know about the internal structures in the sensory system, and can get responses about the world independent of the sensory modality used to determine the information. (The information may be obtained from models, hypotheses, or sensors.)

Currently, two kinds of representation are used in the system. The first is an object-based representation, which describes the geometry and attributes of parts and their instances. The second is a spatial representation, which describes objects by the space they occupy and explicitly represents space that is full, empty, or unseen. The representations are active, in the sense that processes operate

constantly to add new information, modify obsolete information, and maintain consistency as new sensed data are processed. The path planner uses the spatial representation to generate collision-free paths for the robot.

Sensing is heavily prediction-based. Given the expected position of a sensor at some time in the future, the sensing system predicts which important features will be visible and where they will appear in the data. The predicted features are labeled with their names and the names of the objects to which they belong. They are passed to the modules that handle sensory processing, where they are used to guide feature extraction and matching. Matching has the effect of computing new positions for the objects. These are used to modify the internal representation and so allow more accurate predictions the next time around. This feedback loop brings the internal representation into registration with the world.

Our system has been implemented on a set of microprocessors that operate asynchronously and communicate by means of common memory. It makes use of two sensors—an ordinary camera and a structured light range sensor. The sensors are mounted on the wrist of the robot so accurate position and orientation information can be obtained and so they can be moved along specified trajectories.

Representations

As mentioned previously, there are two basic representations, one for spatial information and one for geometric information about objects. We distinguish between generic models, which describe the pure geometry of objects, and instances of models, which describe the unique aspects of individual instantiations of models, such as their positions and orientations. Generic models have an internal coordinate system to which everything is fixed. Instances have extra transformations that locate and orient them in the global coordinate system. There are also generic models of the space occupied by each model, again referenced to an internal coordinate system. There is a single spatial representation for object instances, in which each instance is represented explicitly by the volume it occupies and in which free space is also explicitly represented.

Object representation. The purpose of the object representation is to organize the data describing the objects so that the processes of prediction and updating can be performed as rapidly as possible. There are many ways to represent objects. The best representation depends on the intended application. The representation should be complete, concise, and unambiguous.[2] However, use of the representation should be fast and easy.

In our system, objects are represented by boundary representations. Surfaces are built from edges that are in turn constructed from vertices. These representations are augmented by redundant information to decrease the processing delay associated with the execution of algorithms. For example, given a straight edge and the location of the starting and ending vertices, an algorithm can straightforwardly calculate the length of the edge. However, if the length is explicitly stored, an algorithm using the edge length can execute slightly faster. Several representations to enhance the geometric description are also included. These include octree information, parametric equations, and aspect graphs. A more complete description of the geometric object model can be found in Lumia.[3] While the precise information stored in these representations can change to reflect a better match with the algorithms, the fundamental concept remains the same: calculate as much as possible off-line and store it explicitly in the model.

Generic models have an internal coordinate system to which everything is fixed.

A *generic object* is defined as the description of an object in an object-centered space. An *instance*, on the other hand, provides information unique to a specific part in the real world. While the generic object description is relatively long, the instance description needs only a small amount of information concerning the part's precise location and orientation in the world. Consequently, each instance has a homogeneous matrix that transforms a part location from its standard orientation into a position in the real world.

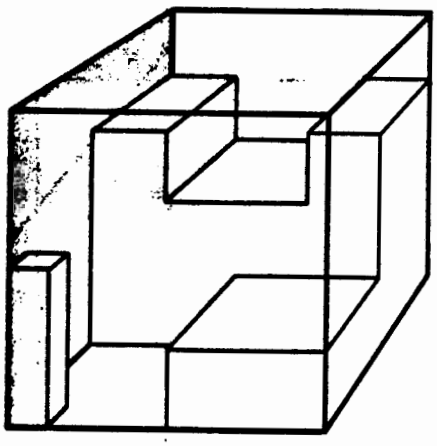
We require that every object be represented in the system at all times. There may be several intermediate stages in the machining of a part. Each stage is considered to be a different part and must be given a description from the external CAD database. This should not be too inconvenient, since the system must know precisely how it intends to process each part.

The generic objects and the instances of each generic object are linked to describe the contents of the robot's workspace. The generic object representations, which store the equivalent of CAD data, are stored in a linked list. The instance information, which includes the homogeneous transform and provides the confidence that the instance is at the location specified by the transformation, is also stored in a linked list. Generic objects and instances appear (disappear) by allocating (freeing) memory and modifying the links.

The current method of inputting object models to the system is through a menu-driven front end and takes the form of sequences of surface, vertex, and edge descriptions for each object. This information is often available from CAD systems, but a parser is needed to convert from the particular internal format of each CAD system to that used by the sensory system. It does not appear that current CAD systems use internal representations that are optimal for most of the operations described above, although they usually support some forms of prediction (e.g., rendering). The problem of converting from one format to another is difficult, but the conversion can usually be done off-line, either at the time the part is defined or before the models are downloaded at the beginning of a task.

Spatial representation. Complementing the object representation, the spatial representation provides a way of indexing into the world by position. This facilitates finding out information about free paths through space and answering questions about what is in front of the robot. It is also very useful for predicting what will be visible from a particular viewpoint (doing such predicting through occlusion analysis and the computation of relative positions of objects and features). The requirements of a spatial representation are that it should encode spatial information explicitly and should allow fast computation of spatial relationships.

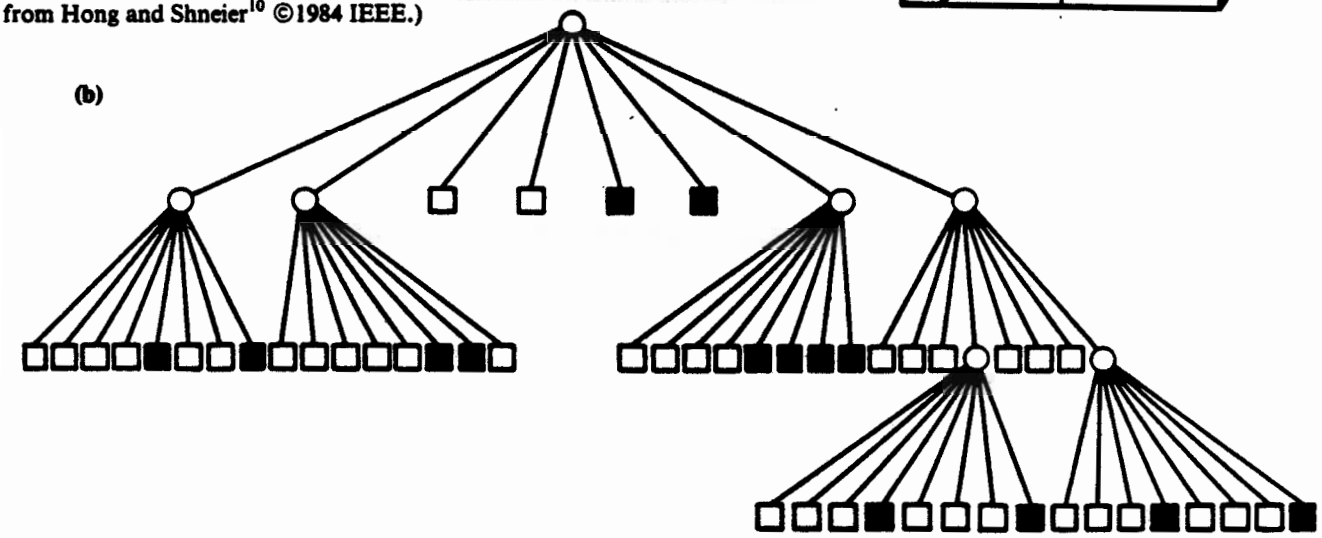
The spatial representation in our implementation is organized as an octree. An octree is a recursive decomposition of a



(a)

Figure 2. Objects enclosed within a cube (a), and the octree representing the volume contained by the cube (b). Round nodes are nonterminal nodes and square nodes are terminal nodes. Black nodes represent regions occupied by objects. Reprinted from Hong and Shneier[10] ©1984 IEEE.)

(b)



cubic space into subcubes[4] (Figure 2). Initially, the whole space is represented by a single node in the tree called the root node. If the cubic volume is homogeneous (is full or empty), then the root is not decomposed at all and comprises the complete description of the space. Otherwise, it is split into eight equal subcubes (octants), which become the children of the root. This process continues until all the nodes are homogeneous or until some resolution limit is reached.

Many of the operations on the octree involve casting rays or projecting volumes through the space. To discover what is in front of the robot (or a sensor), a cone is projected into the tree and the objects that intersect its volume are extracted. To decide if a particular feature is occluded from some viewpoint, a ray is cast from the viewpoint to the node and checked for intersection with objects along the way.[5] To decide if a particular motion of the robot is collision-free, the volume swept out by this motion is checked for intersection with objects.[6,7] In the octree, these operations can be done rapidly, especially if special hardware is available to perform transformations. (To speed up our implementation, we designed a homogeneous matrix multiplier.)

The spatial representation is linked to the object-based representation so that queries that need information from both representations can be answered easily. This can be especially useful where fine motion close to objects must be planned, since the octree alone does not represent object shapes to sufficient accuracy, and the object representation alone does not adequately represent the relationships between objects and the space around them.

The octree is constructed initially from information given to the representation system at the start of the task. Later, it is modified by incoming sensed data, in a way to be described below. The a priori information consists of individual object-centered octrees, one for each generic object, and expected positions for each of the instances of these objects. We should mention that the original design of the system assumed that the models would be developed using the PADL-2 CAD system. This system, in addition to being able to generate boundary representations, uses octrees for volume computations. While the current system does not take output from PADL-2, it does assume that the octrees are available. We wrote a special program to construct the octrees from the object models described above.

The initial data are used to build a world-centered octree describing the entire workspace of the robot. The objects are assumed to appear exactly as expected, and they are projected into the tree. The regions that are not filled are labeled as unseen. (This projection algorithm is described by Hong and Shneier.[8]) At this stage, it is possible to start predicting what the sensors should see and to begin planning paths through the space.

Prediction

Prediction requires a knowledge of the parameters of the sensors and of the sensors' positions at some future time. It makes use of both the object and spatial representations to construct a set of features to be extracted from the sensed data.

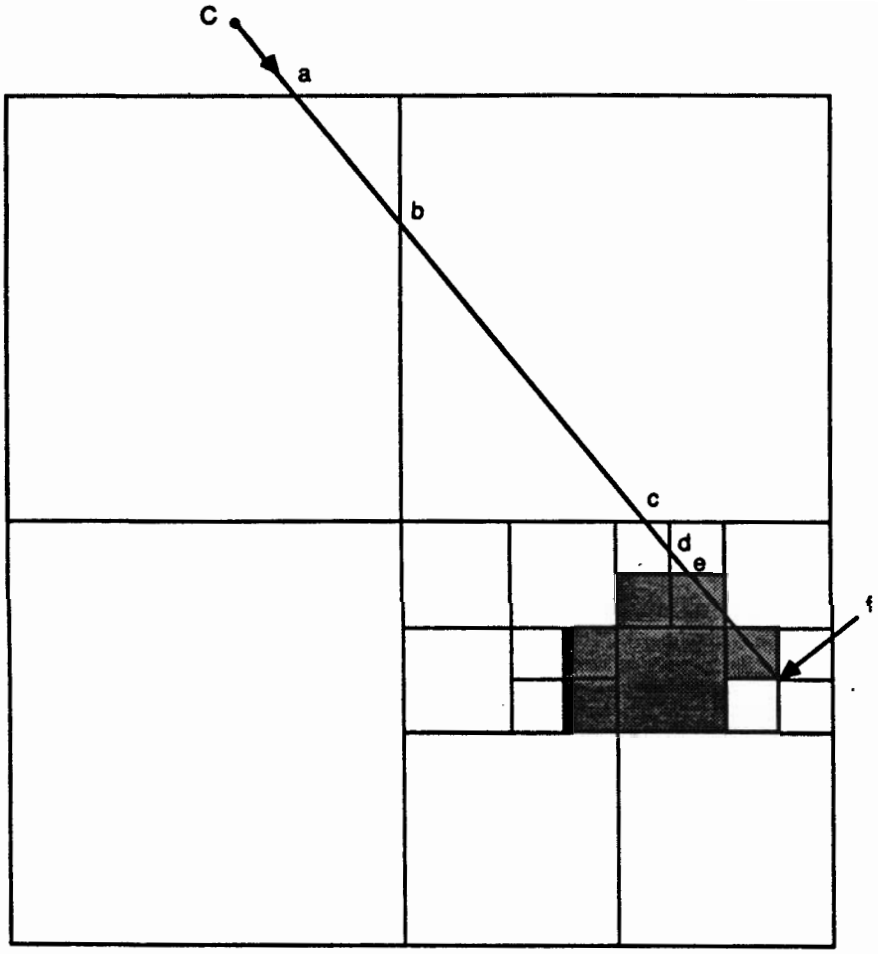


Figure 3. Ray tracing in a quadtree.

These features are a subset of those which the sensory system can extract and are described in terms of the feature type and parameters. (For example, a feature may be described as a corner with a given included angle in the coordinate system of the sensor.) Each feature has an associated certainty window, which corresponds to a region in the image in which it may appear. It also has a label giving its name and that of the object to which it belongs. This information is passed to the sensory system and is used in both feature extraction and matching. The process of constructing the predictions is as follows.

Predictions must be made for both known and unknown objects. For unknown objects, the methods are quite primitive. Features associated with an unknown object are predicted on the basis of the time sequence (velocity measurement) of the location of the feature. This can present severe problems if the features appear and disappear as a result of occlusion. However, once the object is identified, many of these problems go away.

The prediction of features for known objects uses the geometric and spatial representations. Currently, only point features—a corner with an included angle, for example—are predicted. Eventually, prediction of nonpoint features—the observed edge length, for example—will also be desired, but such prediction will be more difficult to do because of partial occlusion. The discussion here is limited to prediction of point features.

The prediction of point features uses both the geometric and spatial representations. First, the geometric representation is used to predict the location of each of the features of each instance. Then, the spatial representation is used to remove the features that are hidden from the camera. Given the camera position in the world, a cone of visibility is constructed. Each instance of each generic object is checked for intersection with the cone, using the spatial representation. For each instance that intersects the cone, the features (which are stored in the geometric representation) are first transformed into the world coordinate system and then projected into the image plane of the camera. This results in a prediction that includes a specific feature type, its parameters, its expected location in the image plane, the generic object name, and the instance name. Some of the features projected into the image plane may not be visible due to occlusion. Occlusion may result from either interobject occlusion, in which a feature cannot be seen because another object is in the way, or self-occlusion, in which the position of the object itself hides the feature from the camera. Removal of hidden features is performed by casting rays into the spatial representation and noting whether the ray first intersects an object or the feature location. Figure 3 shows this process working on quadtrees. A number of *full* nodes are in the lower right-hand corner. The purpose of the algorithm is to determine if the feature at point f is visible when the camera is at point c. The idea is to trace the ray from point c to point f until a full node is reached. If point f is contained in the full node, the feature is declared to be visible. Otherwise, the feature is declared hidden. Lumia[5] provides more details.

The above process results in a list of visible feature locations and the generic object and instance associated with each feature. Although the list of features could be passed directly to other algorithms, it is better to define a neighborhood, or window, around each predicted feature location to minimize the number of calculations. The instance information stores a confidence value that indicates how sure the system is about the actual location of the object. A window with a size inversely proportional to this confidence value can be created. The feature is expected to fall within the bounds of this window. The feature, window, generic object, and instance information are passed to the feature extractor as well as to the 2D matcher. The use of this information in matching and updating the representation is described below.

Sensory processing and matching

Both sensory processing and matching make use of the predictions computed from the current internal representations. Sensory processing is simplified by the focusing effect of the predicted features and the windows provided for each fea-

ture, while matching is made easier by the labels attached to the predictions. A major problem that has to be dealt with, however, is that of unexpected objects. These have to be detected and described so that the representations can be updated. This implies that the areas outside of predicted windows must also be processed, but in a more generic way.

Sensory processing thus involves two passes. We will describe only the technique used for camera data; the range data techniques are less developed. What is currently implemented involves a certain amount of generic processing. This includes dynamic image thresholding and connected-components analysis. With the components and the set of predicted features provided as givens, the sequence described below is executed.

For each predicted feature, each connected component is checked to see if it intersects the window surrounding the feature. If it does, the appropriate feature detector is applied to that component, within the window, and any detected features that have the correct parameters are labeled with the name of the predicted feature. The component is also labeled as expected. It is possible for a predicted feature to find more than one match or for an image feature to match with more than one prediction. This ambiguity is treated by the matching routines. After all the predicted features have been processed, the connected components are checked in sequence to find any that have no matches with predictions. When such a component is found, a set of generic feature detectors is applied to the component. The resulting features are all labeled as belonging to the same object (for which a new name is constructed) but are each given a unique feature label. Thus, each connected component is required to belong to an expected object or is treated as a separate unexpected object. Later processing can coalesce objects that are found to be connected—when seen from another viewpoint, for example—or can separate objects that were seen as part of a single object, perhaps due to occlusion. These operations are accomplished using the spatial representation.

What is passed up from the sensory system is thus a set of labeled features, where some of the labels are those that were predicted and some are newly invented names for unexpected objects. Information about errors in finding expected features is also available. The matcher takes the labeled features as input and attempts to find consistent sets that correspond to instances of the objects. When successful, the matching process produces as output an updated position and orientation for each object, which are used to modify its representation (see below).

Matching has two phases. The first, which is quick and simple, applies to features that were both predicted and found. Here a least squares technique is used to transform the model, whose name is known from the prediction, onto the data. The transformation that has the least error provides the new position of the object and is passed back to the representation. (This process is explained by Rutkowski, Benton, and Kent.[9]) The matching process also supplies an error term that is used in the predictions to decide on the window size for predicting features. For unexpected objects, the set of features is passed to the representation to be used in describing the object. The features are also passed to a special recognition module. This module attempts to match the data to all the known models but is not required to work in real time, since the objects can be described and manipulated even without being recognized.

A problem that currently is not well handled is that of noise in the sensing process.

A tracking process was developed specially for unexpected features. By computing the velocities of the features in the image and knowing the motions of the sensor, it attempts to attach the same label to features across successive images. When a feature is seen from several viewpoints, triangulation can be used to compute its three-dimensional position, which then provides a better prediction of its appearance in later views (and can also help the recognition process).

Updating the representations

The representations are updated from information supplied by the sensory system. This information can take the form of new positions and orientations for known objects or new features grouped to form new unknown objects. There are two ways of updating the spatial representation. One makes use of the processed data used for matching, and one uses the original image data.

Object-based representation. Each instance of an object is stored as a row in a table. Information about the instance includes its current position, which is stored as a homogeneous matrix, a set of features that provide evidence for the object, and pointers to its spatial representation and to its generic object description if it has been identified. Updating the information for expected objects is straightforward. The row in the table corresponding to the object is found, and the new position matrix replaces the previous matrix. Each feature is also updated, either by modifying its position or by adding new features. (For objects that have been recognized, it is not really necessary to store the features, but they are available and are stored for consistency with the representation for unexpected objects.)

The first time an object is seen, a new entry is created in the table. It contains the unique name of the object instance and the set of features associated with the instance. Initially, there is no pointer to a generic model, but a pointer to the spatial representation is created. Strictly speaking, an object can be unexpected only the first time it is seen, since thereafter there will be some capability for predicting its appearance. We nevertheless maintain a distinction until the object is recognized and the pointer to its generic model is filled in.

A problem that currently is not well handled is that of noise in the sensing process. Unexpected objects may simply be artifacts of the imaging process or may be distorted by that process. Certain features may be the result of special viewing angles or occlusions. Currently, we simply keep a count of the number of times an item has appeared. This is clearly inadequate, since observing an object from substantially the same position many times yields much less information than observing it twice from widely separated viewpoints.

Spatial representation. The octree describing the spatial layout of the robot's world has three kinds of nodes. There are nodes labeled as empty, which are known to contain free space; nodes labeled as full, which are also labeled with the objects they contain; and nodes labeled as unknown, whose volume has not been seen. The processes that update the representation

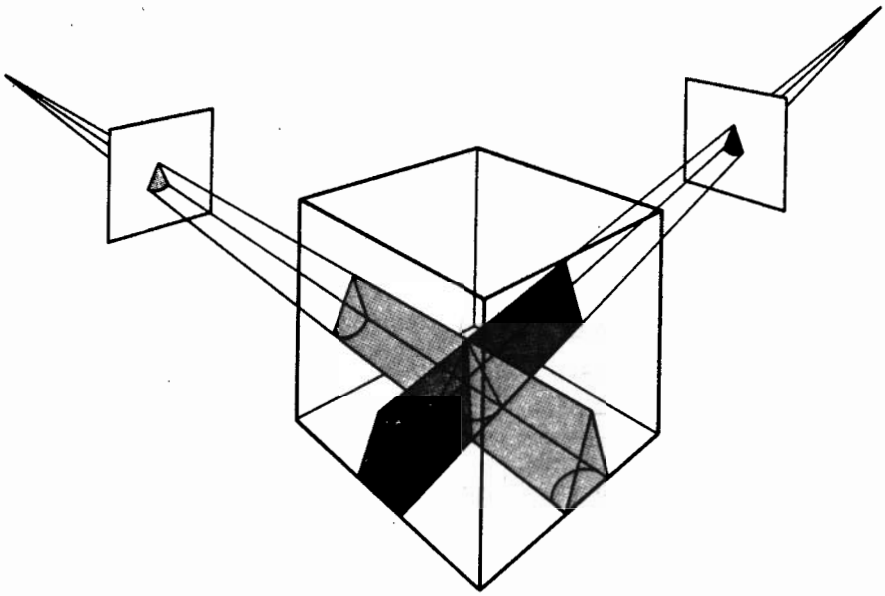


Figure 4. The effect of intersecting the cones produced by two views of an object. (Reprinted from M.O. Shneier, E.W. Kent, and P. Mansback, "Representing Workspace and Model Knowledge for a Robot with Mobile Sensors," *Proc. 7th Int'l Conf. Pattern Recognition*, ©1984 IEEE.)

modify the tree according to current sensory information. Their goal is to reduce the number of unseen nodes and to ensure that the full nodes are in the right places.

The first of the two processes used to update the tree takes as input the position matrices of those objects whose positions have been changed as a result of sensory processing. It erases the old representations of the objects and projects the object-based generic octrees for each object into the global octree, using the updated position matrices. This process is described by Hong and Shneier[8] and is essentially the same procedure as is used to construct the initial spatial octree.

The second process is more primitive but more comprehensive. It handles unexpected objects as well as expected objects and explicitly accounts for the uncertainties in their spatial extents. Given an image of the world from a known viewpoint, a set of "cones" is projected into the octree (Figure 4). The cones are defined by the sets of rays that pass through the focal center of the camera and are tangent to the silhouettes of the objects. Each component in the image gives rise to a cone, and another cone is created by the boundary of the image. This last cone reflects the free space "visible" from the given viewpoint and has holes in it for each of the objects. This projection process is described in detail by Hong and Shneier.[10]

It is clear that from a single view little can be said about where in the cone an object actually lies. From successive views, however, the object can be constrained to lie in the intersections of the resulting cones (Figure 4). These intersections not only provide position information but also constrain the possible shapes of objects. They are also useful in segmenting unexpected objects. A set of features extracted from an image may appear to belong to a single unexpected object from some viewpoint, but may be found to belong to separate objects when information from a different viewpoint shows the separation. This extra information can be useful in the recognition process. The cones associated with each object also provide information that may be useful in the future for deciding where to point a sensor to reduce ambiguity in the scene as much as possible, by reducing the volume of the cones.

Planning and control

The planning and control components of the robot system interface with the sensory system through the internal representation, as shown in Figure 1. The sensory system constructs and updates this representation, while the planning and control system queries the representation about object locations and orientations, object velocities, object sizes, and so on. Here, we discuss the path planning and collision avoidance component. The goal of this component is to generate collision-free paths in real time for three-dimensional movement by the robot.

In our system, path planning and collision detection use the octree spatial representation. The nodes of the octree actually form the search space during path planning. For collision detection to be performed, the robot's volume, and the volume it sweeps out when it moves, must be represented. The path planner takes as input the configurations of the robot in the start and goal positions. The output is a sequence of intermixed translations and rotations in 3D space. This path is then passed to the robot controller for execution.

In performing a search through the octree space, the planner combines several techniques in an attempt to achieve the greatest speed in finding free paths. The first technique, called hypothesize-and-test, involves hypothesizing a simple path for the robot by generating the volume it will sweep out during a motion. We consider two kinds of simple paths, one for translation and one for rotation. Any complex path can be approximated by an intermixed sequence of these two paths.

The second and third search techniques are hill climbing and the A* search. A* is a best-first, tree-structured search method.[11] Hill climbing and A* are applied to a graph representation of the octree search space, initially obtained by connecting all adjacent leaf nodes of the octree. These two methods are used to obtain translation components of the robot's path. The methods complement one another—hill climbing is fast, but because it searches only locally, it can get stuck at a local minimum in the cost function; A*, though slower, can get the robot out of local minima since it searches globally.

The fourth search technique is called the multiresolution grid search. Using the following method, this technique offers a way to search at a finer resolution than that of the octree search space. A high-resolution grid is placed within the octants of the octree, and this grid is searched in a multiresolution fashion. To determine whether a path from one point to another is valid, the hypothesize-and-test technique is applied by forming a swept volume between the two positions and checking for collisions with obstacles.

For potential collisions to be detected, the swept volume representing the robot's path must be compared with obstacles represented in the octree. The robot is repre-

sented in terms of its links, their attachment relations, and their axes of motion. Each link as well as the swept-volume paths formed by translation or rotation of a link are approximated by a set of primitive shapes.

There are three requirements for defining a primitive shape. The first is that the computation that determines whether the shape intersects an object in the octree must be fast. The primitive shapes are therefore defined in terms of a *spine*—either a point, a simple curve segment (e.g., a straight line segment), or a simple surface segment (e.g., a parallelogram)—and a *radius*—a single extension outward from the spine that defines the shape's surface. By representing octants in the octree as combinations of spheres, the intersection test only has to determine the shortest distance from the center of a sphere to the spine of the primitive shape and check whether this distance exceeds the sum of the radii of the sphere and shape.

The second requirement for a primitive shape is that it should be a reasonable approximation of a part of the robot or its swept volume. The third requirement is that the generation of primitive shapes should be very fast because a particular shape, if it is to represent a swept-volume path, must be dynamically generated during searching. Many of the shapes are therefore defined as translational or rotational sweeps of some other primitive shape. Examples of primitive shapes used in the system are

- a sphere, defined as a point spine and a radius (Figure 5a),
- a cylsphere, a volume swept out by linear translation of a sphere and defined as a line segment spine and the radius of the sphere (Figure 5b),
- a volume swept out by linear translation of a cylsphere and defined as a parallelogram spine and the radius of the cylsphere (Figure 5c),
- a volume swept out by rotation of a cylsphere about an axis intersecting and perpendicular to its spine, and defined as a planar spine shape and the radius of the cylsphere (Figures 5d and 5e), and
- a volume swept out by rotation of a sphere about an axis outside the sphere and defined as an arc segment spine and the radius of the sphere (Figure 5f).

The latter volume is obtained if a rotation of a robot link is to occur about its axis of motion. More details about the path planning system can be found in the papers by Herman.[6,7]

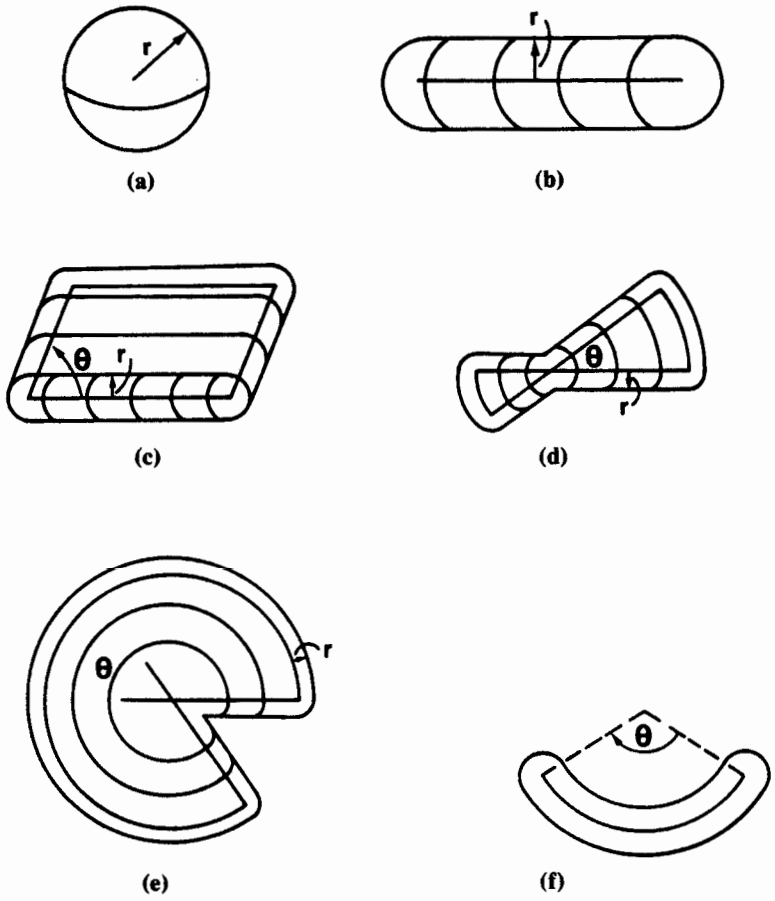
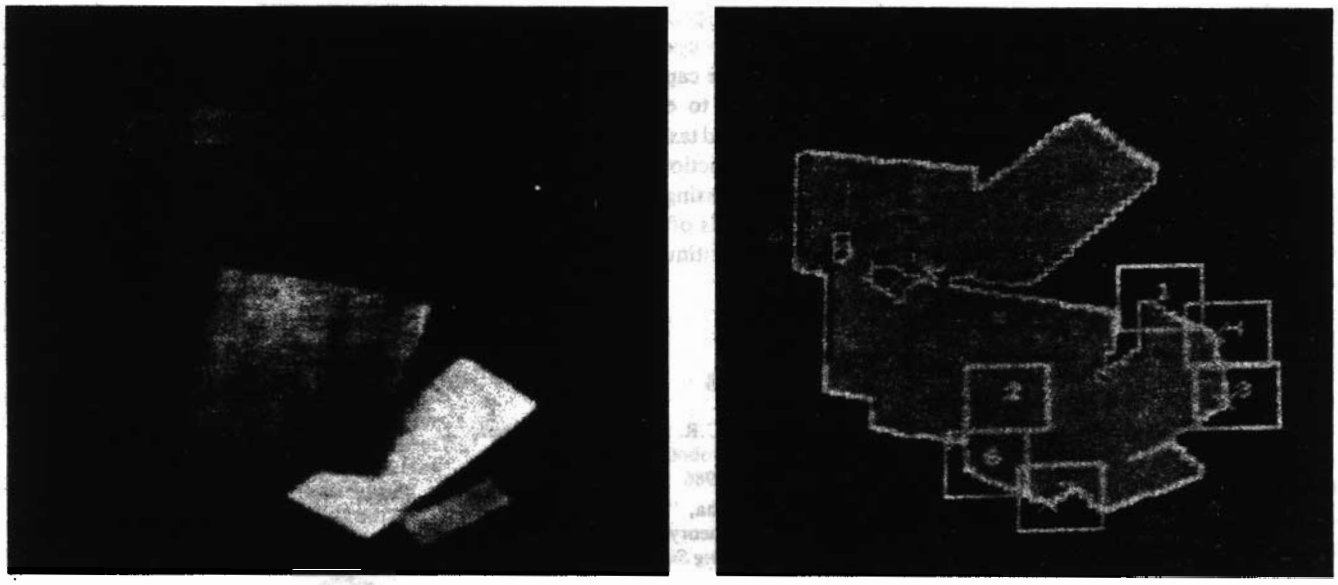


Figure 5. Primitive shapes used by the path planning module—sphere (a), cylsphere (b), translation-swept cylsphere (c), rotation-swept cylsphere (d), another rotation-swept cylsphere (e), and torus section (f). (Reprinted from Herman[6] ©1986 IEEE, SPIE.)

Implementation

The sensory, representation, and path planning system has been implemented as a set of modules, each of which runs in a separate microprocessor. A distributed operating system called GRAMPS has been developed to control the communication between processes, which takes place through common memory.[12] GRAMPS is a truly distributed operating system in that parts of its code reside on each of the processors and an unlimited number of processors can operate on the same bus, either independently or cooperatively. Any of the processes can be restarted at any time without affecting the operation of the others, except in so far as they communicate with each other.

Processes communicate via common memory. Two basic interfaces are available. In the first, regions of memory are defined as "files" and are read and written by those processors that know their file names. A semaphore system is used to ensure that only one user at a time can write a file and, if desired, that only one reader can read it. There are no restrictions on the number of users for each file. The second interface relies on dynamic memory allocation and the passing of pointers to structures allocated. Any processor can allocate structures in common memory and pass pointers to any set of other processors (through a file). The memory can be modified by all the processors that have pointers to it and will not be freed until the last of the processors finishes using it. This is the preferred way of com-



(a) (b)

Figure 6. Image of a stack of objects (a); processed image of stack, with windows showing predicted locations of corners (b).

munication, since it requires much less bus traffic.

Currently, eleven processors are defined for the system (see Figure 1 again). One processor handles sensor interactions and controls the illumination. Another performs generic image processing and builds low-level descriptions of the images. These two processors do not make use of any top-down information, except commands to take pictures using either the range sensor or the camera. All the rest of the processors are much more closely linked to the model data.

A third processor is dedicated to feature extraction. It takes as input a set of predictions and the preprocessed image data. Its outputs are labeled features, descriptions of unexpected features, and errors. These are sent to the matcher, which is implemented on yet another processor. The matcher attempts to register models with the labeled features and computes new positions as well as errors in the fitting process. These are sent to the modeling modules along with any unexpected features, which are grouped into tentative objects.

There is also a prediction module, which constructs the expected features for future views, two modules to update the representations (one for the object-based representation and one for the spatial representation), and a feature-tracking module. A supervisory module has been designed but has not yet been integrated into the system. Its role will be to decide which sensors to use to best update the representations and where to concentrate processing efforts. It will also make sure that all the other modules are running correctly. Finally, there is the path planning module, which interfaces with the robot controller.

All the modules have been implemented, although some are not yet running on the multi-microprocessor under GRAMPS. The whole system has nonetheless been run on real images, and data have been passed from module to module, in some cases using common memory and in others using serial or parallel links. As the hardware difficulties inherent in running so many processors on the same backplane are overcome, more and more components of the system are being integrated into a uniform whole.

Figure 6a shows a gray-scale image of a stack of objects. The system was given as input a description of the blocks and the position of the camera when it took the picture. Prediction was restricted to the corners of the parallelepiped in front of the pile and was able to reject corners that were occluded. The windows in Figure 6b show the predicted locations of corners, with numbers indicating their identities. Where corners were indeed found, crosses indicate their actual positions. Some of the corners were not found, since the prediction system assumes that gray-scale images will be processed and the feature extraction was actually done on thresholded images. Thus, some features that would normally be extracted do not appear. Only those features in the silhouette of the object pile are visible. We have not attempted to remove the internal corners because we expect to use gray-scale images in the future and because the system is able to deal with the missing information. From the set of corners that were matched, a new position was computed for the object and fed back to the representations. A sequence of experiments was carried out to measure the convergence and accuracy of the three-dimensional pose-update algorithm. The pose is represented by rotation and tilt. In the experiments, rotation was always calculated within five degrees of the true value and tilt within three degrees. Convergence took two to three iterations in all cases and was well behaved. These results are presented in detail in the paper by Rutkowski, Benton, and Kent.[9]

The path planning system has been tested separately with a real robot. The volume of the robot gripper was approximated as the union of fifteen spheres. Up to five block-shaped obstacles were placed arbitrarily on the robot's workbench. Arbitrary start and goal points were then chosen. The system successfully found reasonable collision-free paths between all sets of start and goal points. However, only translation of the gripper was tested. Rotation of the gripper links has not yet been tested.

The industrial robotics environment is particularly well suited to the use of models of objects. More and more often, these models are available from CAD systems. However, there is an important question about the adequacy of most CAD systems as the sole source of model information for a sensory system. Many of the properties of objects that are needed for a sensory system are not typically represented in CAD systems. Often, the concept of an object is not very well defined. An object may be created merely as a coincidence of a set of surfaces or volumes rather than as an explicit entity with its own properties and attributes. Often, surface finish, surface markings, and even surface shape (for example, the pitch of a screw) are stored only as annotations. Some of these properties are essential for recognizing and locating objects using a sensory system.

The diversity of CAD systems, each based on very different primitive concepts, makes approaching the automatic model transfer problem difficult, especially in an environment in which the choice of the design system is not in the hands of the designers of the sensory system. The system described here does not provide the interfaces and additional information needed to use a CAD system directly but, at the same time, does not use any internal representations that cannot currently be extracted from a CAD system, albeit with the application of a certain amount of intelligence.

The system described here is unique in its scope. It combines sensory processing, object and spatial representation, and path planning in a unified and integrated manner. It uses prediction-driven vision and sequences of intensity and range images to maintain an internal representation of the world. It uses this internal representation to answer questions about the world posed by the control system. These include questions about path planning and questions about locating objects by name or position.

Geometric models play a central role in performing the sensing, planning, and control. They are used for predicting features in the sensory data and for recognizing sensory data, and they provide a vocabulary by means of which the control and sensory components can communicate.

The system has been successfully run with a number of sample inputs. Our current efforts are aimed at bringing up the entire system on a distributed processing system, building special hardware where necessary to allow operation in real time, and extending the capabilities of each of the components to encompass a wider class of objects and tasks. We will continue to focus on prediction-based techniques for sensory processing, making heavy use of a priori models of objects and of the coherency and continuity exhibited by the real world. □

References

1. R.T. Chin and C.R. Dyer, "Model-based Recognition in Robot Vision," *Computing Surveys*, Mar. 1986, pp. 67-108.
2. A.A.G. Requicha, "Representations for Rigid Solids: Theory, Methods, and Systems," *Computing Surveys*, Dec. 1980, pp. 437-464.
3. R. Lumia, "Modeling Solids for a Real-Time Vision System," *Proc. PROLAMAT*, Paris, France, June 1985.
4. C.L. Jackins and S.L. Tanimoto, "Octrees and Their Use in Representing Three-Dimensional Objects," *Computer Graphics and Image Processing*, Vol. 14, 1980, pp. 249-270.
5. R. Lumia, "Rapid Hidden Feature Elimination Using an Octree," *Proc. 1986 IEEE Int'l Conf. Robotics and Automation*, Apr. 1986, pp. 460-464.
6. M. Herman, "Fast, Three-Dimensional Collision-Free Motion Planning," *Proc. 1986 IEEE Int'l Conf. Robotics and Automation*, Apr. 1986, pp. 1056-1063.
7. M. Herman, "Fast Path Planning in Unstructured, Dynamic, 3-D Worlds," *Proc. SPIE—Applications of Artificial Intelligence III*, Vol. 635, Apr. 1986, pp. 505-512.
8. T.H. Hong and M.O. Shneier, "Rotation and Translation of Objects Represented by Octrees," *Proc. 1987 IEEE Int'l Conf. Robotics and Automation*, Apr. 1987, pp. 947-952.
9. W.S. Rutkowski, R. Benton, and E.W. Kent, "Model-Driven Determination of Object Pose for a Visually Servoed Robot," *Proc. 1987 IEEE Int'l Conf. Robotics and Automation*, Apr. 1987, pp. 1419-1428.
10. T.H. Hong and M.O. Shneier, "Describing a Robot's Workspace Using a Sequence of Views from a Moving Camera," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 7, 1985, pp. 721-726.
11. N.J. Nilsson, *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.
12. P. Mansbach and M.O. Shneier, *The GRAMPS Operating System: A Guide for Users and System Administrators*, Robot Systems Division, National Bureau of Standards, Gaithersburg, Md., 1987.

Readers may write to Herman at the Sensory-Interactive Robotics Group, National Bureau of Standards, Bldg. 220, Rm. B124, Gaithersburg, MD 20899.

Michael Shneier is project leader for machine vision at Philips Laboratories, a North American Philips corporation. His interests lie mainly in robotics and include object representation and recognition, space representaiton, prediction-guided feature extraction, and multisensor fusion. He is also interested in medical image processing, including image enhancement, image coding and compression, and picture archiving and communication.

Shneier holds a PhD in artificial intelligence from the University of Edinburgh, Scotland.

Ronald Lumia is group leader for intelligent control at the National Bureau of Standards. His research interests include robotics, teleoperation, computer vision, and factory automation.

Lumia received the BS degree from Cornell University in 1972 and the MS and PhD degrees from the University of Virginia in 1977 and 1979, respectively. He has held both industrial and academic positions and has authored over 30 technical publications. Most recently, he was a senior teaching fellow at the National University of Singapore.

Martin Herman is group leader of the Sensory-Interactive Robotics Group at the National Bureau of Standards. His interests include robotics, real-time path planning, robot vision, image understanding, and autonomous vehicles. He is currently working on the MAUV (Multiple Autonomous Underwater Vehicle) Project, where he directs research in real-time planning and world modeling. He also directs research in real-time vision for robotics. Previously, at Carnegie Mellon University, he was the principal scientist involved in developing a vision system, 3D Mosaic, for 3D interpretation of complex aerial images.

Herman received the BS degree in physics from The Cooper Union, New York, and the MS and PhD degrees in computer science from the University of Maryland.