Pattern Recognition in a Multi-sensor Environment[1]

Thomas C. Henderson and Wu So Fai

UUCS 83-001

18 July 1983

Department of Computer Science
University of Utah
Salt Lake City, Utah 84112

Abstract

Current pattern recognition systems tend to operate on a single sensor, e.g., a camera, however, the need is now evident for pattern recognition systems which can operate in multi-sensor environments. For example, a robotics workstation may use range finders, cameras, tactile pads, etc. The Multi-sensor Kernel System (MKS) provides an efficient and coherent approach to the specification, recovery, and analysis of patterns in the data sensed by such a diverse set of sensors. We demonstrate how such a system can be used to support both feature-based object models as well as structural models. The problem solved is the localization of a three-dimensional object in 3-space. Moreover, MKS allows rapid reconfiguration of the available sensors and the high-level models.

---

[1]This work was supported in part by the System Development Foundation and by NSF grants ECS-83-07483 and MCS-82-21750

Table of Contents

List of Figures

List of Tables

1. Introduction

The Multi-sensor Kernel System has been proposed as an efficient and uniform mechanism for dealing with data taken from several diverse sensors [15]. The system can be logically divided into three major parts: the sensor specification, low-level representation, and high-level modeling. In this paper, we discuss in detail the high-level modeling aspects of the system.

1.1. Sensor Specification

Following the work of Foley [7], Pfaff [21], and Rosenthal [22] in the domain of logical graphical input device specifications, we have shown how MKS can provide a similar function in a multi-sensor environment [15]. Basically, a logical (or abstract) sensor can be viewed as a program whose inputs are either system defined devices (such as cameras) or the output of other logical sensors. Thus, logical sensors are formed from existing device drivers or from programs which perform some analysis on the data from other logical sensors. This allows for hiding of sensor type in that such a device can be implemented in hardware with only a driver interface, or can exist through several levels of software. Moreover, such a system can run interpretively in the sense that failure of a physical device does not preclude the definition of an alternative method for obtaining the same information. Such a sensor specification can be viewed as a sensor definition language, and thus, a semantics of sensor definitions can be given, and this provides the user an extremely useful tool. Finally, a logical sensor is also defined in terms of its output. For uniformity of processing, the output of each logical sensor is defined as an n-tuple whose elements all have a meaningful label that the user can understand and a range of possible values for that element. For example, a camera could be defined in terms of a triple: (x-location, y-location, intensity) with allowed ranges: (0:1023, 0:1023, 0:255). A camera then returns a stream of such triples. In this way, the output of one logical sensor can be quite conveniently used as the input to another logical sensor.

1.2. Low-Level Processing

The results of several workers in computer vision have shown the usefulness of performing a low-level processing step before attempting high-level analysis of the sensed data. In particular, Marr's primal sketch [17], Barrow and Tennenbaum's intrinsic characteristics [4], and to some extent, the region adjacency graph of Pavlidis [20], have all been proposed as a low-level organizational tool for image data analysis. We have

shown how the recovery of 3-D information can be usefully organized in the spatial proximity graph [9, 15]. Most features (e.g., surface curvature, surface normal, range, texture, etc.) can be localized in 3-space using current computer vision techniques (see Ballard and Brown for an introduction [3]). Other approaches to the organization of point data include minimal spanning trees [25], relative neighborhood graphs [23], and Voronoi triangulations [1].

The spatial proximity graph simply takes a set of points and creates a graph whose nodes are the points, and whose edges connect each node to the m nearest neighbors of that node. MKS allows the spatial proximity graph to be generalized by allowing points from any k-dimensional space, and as will be discussed later, this permits one to analyze structure in any feature space chosen by the user. The spatial proximity graph is built quite efficiently in terms of the k-d tree [8] which is built directly from the data (see Henderson [13, 14] for the use of the k-d tree for feature organization). A k-d tree is a binary sort tree for k-dimensional keys where a nonterminal represents a subset of the points, and a terminal contains a bucket of points. The root node represents the complete set of data, and the set of data points at each nonterminal is recursively divided into two sets by splitting at the median key value along the axis with greatest spread in value. Such an organization minimizes the average search time in terms of record accesses [8]. The user defines the k-tuple (to be used in low-level processing) as a subset of the n-tuple returned by the class of sensors providing the data to the low-level processing module. These k-tuples serve as the basic organizational element of low-level processing, and all high-level models must be defined either directly in terms of them (as when feature models are defined in terms of particular values for each element) or indirectly in terms of the spatial relations existing between the vectors (as in the Hough shape model). As a simple example, consider the following. A logical sensor "Circle_Detector" is defined which takes direct camera data as input and returns a triple giving the (x-location, y-location, radius) of each circle detected in the image, where the x- and y-locations give the center of the circle with respect to the image coordinates. The low-level processing is defined in terms of a 1-tuple (radius) which is a subset of the original triple. The radius value models the invariant part of the data from the "Circle_Detector" and will be sorted by the low-level processing. To discover circles of any given radius, it is only necessary to query the sorted data. In case of a match, the index of the matching data vector can be used to recover the original x-location and y-

location. Finally, the user may also define a distance function to be used by the system if the standard ones such as Euclidean distance and Manhattan distance are not desired.

2. High-Level Modeling

To discover patterns in data requires three things:

1. an abstract model of the pattern,
2. a description of a pattern from the data, and
3. a method for matching the model and the description.

Figure 2-1 outlines the relationships between the model, the description and the matcher.

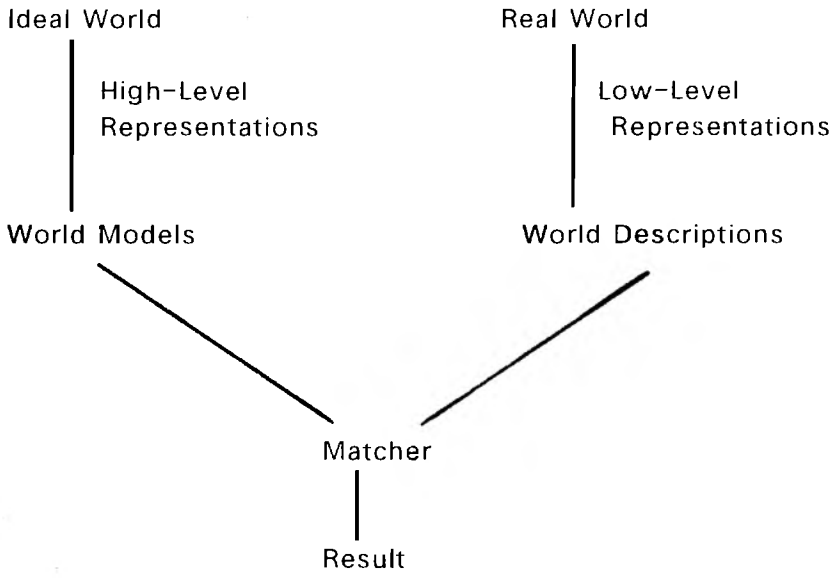


Figure 2-1: High-Level Modeling and Matching

Various types of models exist, and the choice of the model determines in large part how the descriptions will be derived from the data and what form those descriptions take; moreover, once the model and description have been specified, the nature of the matcher should be clear.

The model is first determined in terms of an ideal world, e.g., as a mathematical or prototypical abstraction. For example, it may be decided to model polyhedra. Given this notion, it is then possible to specify some particular mechanism for representing this

class; we call this the high-level representation. For the polyhedra example, this might be one of several possibilities, e.g., each polyhedron might be represented as a graph whose vertexes have a 3-space location. Finally, depending on whether or not the high-level representation is parameterized, it may be necessary to provide a method for deriving a world model from the high-level representation. For example, if it were necessary to detect polyhedra in digital images, then a rendering technique would have to be applied to the high-level representation in terms, perhaps, of lighting sources and surface reflectance properties. Of course, it is possible that the high-level representation and the object model are the same.

The data analyzed is sensed from the real world. This data is then organized into a low-level representation which imposes a desired structure and uniformity on the data. A world description of the sensed data is then derived from the low-level representation. This description is in terms of the object model, i.e., the type and attributes of the object model determine the type and attributes of the description.

The kind of matching performed depends on the object model and description to be matched. Two standard methods of matching can be used: functional and structural. A functional similarity (or dissimilarity) measure is defined as a combination of parameter values defined in terms of measures on the data. These parameters can be conveniently viewed as forming a single vector; this makes it possible to take advantage of standard distance functions on a vector, e.g., Euclidean distance, Manhattan distance, etc. Such a distance can also be defined as a logical function. Structural matching, on the other hand, usually requires solving some form of the subgraph isomorphism problem. This problem is computationally very expensive to solve, and we will therefore make some simplifying assumptions and solve a restricted structural matching problem (see the Hough shape transform below).

Our discussion of high-level object models will center around the following definitions (see Henderson [10]). An object model consists of:

1. a spatial decomposition of the object,

2. a description of the parts of the object, and

3. a description of the relations between the parts of the object.

We classify object models according to the presence or absence of these three components. An object model which provides only (1) and (2) will be called a feature model. An object model which provides all three components will be called a structural model.

Feature models describe objects by specifying a set of features to be computed and a set of reference values (in terms of the mean and standard deviation, perhaps) for the object being modeled. Since values for such features for objects of the same class are rarely exact, we often use feature models together with statistical methods of classification. We define a measure which gives a distance between the feature vector for the model and the computed feature vector of a test shape. The test shape can be described as similar to the model if the distance is less than some specified threshold.

A structural object model describes the spatial decomposition of an object, and consequently must describe the primitive parts composing the object. The primitives should provide a compact description of the object with little or no loss of information. Once the primitives have been obtained, we compute the relations, such as adjacency, collinearity, and symmetry, between the primitives.

In terms of the object recognition problem, we have investigated two approaches to object modeling: feature modeling and the Hough shape transform. Feature modeling provides a simple method for analyzing low-level features that can be recovered reliably and efficiently from the sensors. In the implementation described here, a 3-D object is represented in terms of some simple features of the object, e.g., number of vertexes, number of holes, etc. A sensor is then specified which recovers just these features from newly detected objects in the environment, and then the feature values of the unknown objects are compared to the reference values of the known objects. The Hough shape transform provides for the recovery of the spatial location and orientation of the detected object. This is accomplished by taking advantage of the spatial relations between the vertexes of the object and matching each detected vertex of an object to a specific model vertex.

2.1. Feature Models

Distinct features of the ideal world such as weight, area, major-axis, minor-axis, etc., of objects can be used effectively for object recognition. For example, almost all of the commercial computer vision systems available for use perform object recognition and analysis in terms of some small number of features (10 to 20) which can be easily extracted from a 2-D binary image. Most often there exists a many to one map from real world objects to such a set of features. This usually poses no problem since the set of features is chosen with the entire set of objects to be modeled in mind, and ambiguities can be avoided. On the other hand, a certain amount of location and scale invariance can be easily accounted for by not using any features based on absolute size or location. For example, the class of rectangles might be modeled as four 90 degree corners.

In the context of the multi-sensor framework, feature models are defined directly in terms of a logical sensor. Every logical sensor has a characteristic output vector. The characteristic output vector gives the number of features returned by the sensor, a meaningful name for each feature, and the range of possible values for the feature. For example, the logical sensor "Circle_Detector" returns three features: x-location, y-location, and radius (all with respect to a known viewpoint or camera), and the allowed ranges might be 0:1024, 0:1024, and 0:512, respectively. Thus, to specify a model for circles of radius 2, call the model C2, there must be a mechanism to specify that x-location and y-location are irrelevant for the model, whereas the radius value must be 2. This is easily implemented by allowing "don't care" as the assigned reference value for a feature in the model. Thus, a model is completely specified by naming a logical sensor and then giving a value for every feature returned by the sensor. The vector obtained by omitting the "don't care" values is called the reference vector for the model.

Once a model has been defined, this has certain ramifications for the construction of the k-d tree. Namely, the k-vector used in the construction of the k-d tree is formed precisely from those elements of the vector produced by the logical sensor for which "don't care" was not specified in the model. For example, the previously mentioned "Circle_Detector" based model called "C2" causes the data from "Circle_Detector" to be sorted on the last element of each triple. In general, the spatial proximity graph does not have much meaning in this context. However, we will discuss a convenient mechanism for matching which does exploit the spatial proximity graph.

Matching can be accomplished in several ways; we give three here. First of all, if the models are all based on the same subset of the same logical sensor, then the model vectors can be organized in a k-d tree (the model tree), and as new description vectors are produced by the sensors, a query can be made on the model tree for the nearest neighbor of the new description vector. A threshold can be specified for the query, and the matching function can be built into the tree search. A simple matching function for feature vectors can be defined as follows. Let the model reference vector be $m = (m_1,m_2,...,m_k)$, and let the new description vector be $v = (v_1,v_2,...,v_k)$. Then we define the distance between m and v as:

$$\text{distance}(m,v) = \text{SUM } \{f_i(m_i,v_i)\}.$$

The f_i's provide a mechanism by which feature value ranges can be normalized with respect to each other. The user can specify these functions appropriately with respect to the feature being measured, and also a threshold for the distance function below which a match will be recorded between the model and the description vector. An alternative method for feature matching is to build the description vectors into a k-d tree and then query that tree with the appropriate models' reference vectors. Matching takes place much as before.

Finally, it is also possible to incorporate the model reference vectors into the data. Then when the spatial proximity graph is built, the descriptions which match the reference vectors will be connected to them by edges. When building the spatial proximity graph, it is merely necessary to specify the threshold associated with the model and an infinite number of neighbors (since it can't be known in advance how many objects there are which match the model).

2.2. Hough Shape Models

An extension of the 2-D Hough shape transform to handle 3-D surfaces offers an alternative approach to object identification, and furthermore permits the localization of objects in space. That is, the exact transformation can be found which maps the reference object onto the detected object. The classic Hough transform [16] is a method for detecting curves by using the duality between points on a curve and parameters of that curve [6]. For instance, in the case of straight lines, we could parameterize them by their corresponding slopes and intercepts. The parameter space is then quantized, and an accumulator is associated with each point in the parameter space. The accumulator is

incremented for each detected point whose associated curve in parameter space crosses that accumulator. Local maxima in the accumulators correspond to collinear points in the image space. The values in the accumulators measure the number of points on the line.

The Hough shape transform is a generalization of the classic Hough transform for handling objects which have no simple analytic forms, but have particular shapes [2, 5, 18]. A reference point is picked. For each boundary point, compute the displacement vector from the boundary point to the reference point. Store the reference point and the displacement vectors. The basic strategy of the Hough shape transform is to compute the possible loci of the reference point given edge point data in an image, and is achieved by associating an accumulator with each point in space, and applying the following algorithm: for all e, a detected edge location, and for all d, a displacement vector, increment the accumulator at (e+d). Possible locations for the reference point are given by the maxima in the accumulator array. Figure 2-2 gives an example of the Hough shape definition. Figure 2-3 shows the corresponding Hough shape detection, and blank represents a zero in the accumulator array.

The extended 3-D Hough algorithm works as follows [11, 12]. A 3-D object is represented as a collection of n vertexes. These vertexes in turn are denoted by their (x,y,z) locations. Call this set of 3-D points P, where P = $\{(x_i,y_i,z_i\},)$ i=1,...,n. Choose some reference point, P_0 = $(x_0,y_0,z_0,)$, e.g., the centroid of the set of triples. The object representation is given as a list of displacement vectors from each point in P to the reference point P_0. The model is then a characterization of P as a displacement from each vertex to the reference P_0.

Given the spatial proximity graph representation of a set of points sampled from the surface of a polyhedral object, the points in the graph can be grouped to find the planar regions. The planar faces of the detected object are then intersected to find the vertexes of the object. The detection procedure is then to match the set of model vertexes with the detected vertexes as points in the transform space. From matching each point on the surface of an object to matching a small set of points representing the vertexes of the object, we arrive at a much reduced set to be matched.

For each detected vertex, we associate the n displacement vectors to compute the possible loci of the reference point, P_0, in parameter space. Accumulate counts for the

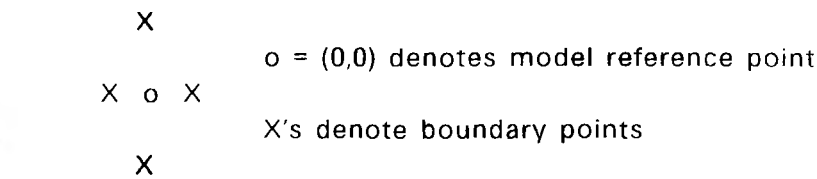
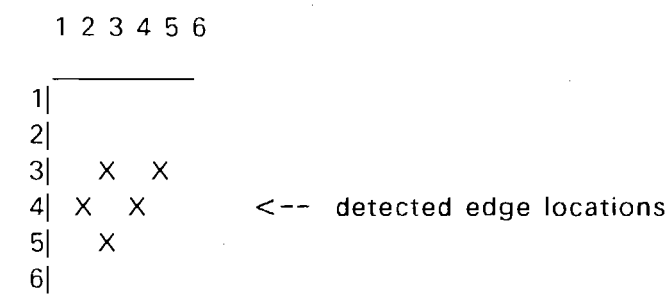


Figure 2-2: Example of Hough shape definition



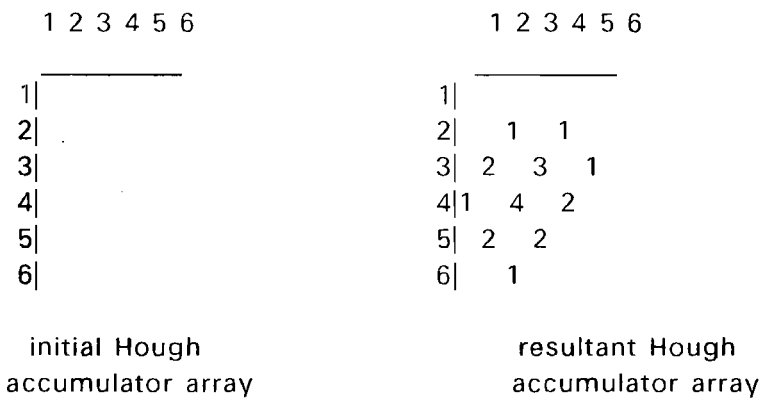


Figure 2-3: Example of Hough shape detection

possible locations of P_0. The location that has the maximum count in the 3-D space corresponds to the translated position of the reference point P_0 of the object model.

The above algorithm produces a unique maximum for any translated P, and the maximum value is equal to the number of detected object vertexes. But if all the points in P are not in the detected object, then the maximum will be less than the number of points in the model object. Moreover, if there are several copies of the object, then there may not be a unique maximum. However, the reference point is always guaranteed to be one of the maxima.

Since in general, objects are both translated and rotated, a more realistic Hough shape model will be the one which deals with rotation as well as translation. The following section outlines the algorithm to compute rotational invariant 3-D Hough transform. (For details, see Wu [24].)

2.2.1. Rotational Invariant 3-D Hough Shape Model

Given a set of points $P=\{(x_i,y_i,x_i)\}$, i=1,n, representing a 3-D object model, choose some reference point P_0, (x_0,y_0,z_0), such that the lengths of all the vectors of $R=\{(dx_i,dy_i,dz_i)\}$, where $dx_i=x_0-x_i$, $dy_i=y_0-y_i$, and $dz_i=z_0-z_i$, are distinct. The model representation is then in terms of R and the reference point, P_0.

Given a set of detected points $D=\{(x_i,y_i,x_i)\}$, i=1,m, use a 3-D array H, to accumulate counts for possible location of P_0 in space. Then the rotation invariant 3-D Hough transform is computed by:

```
for all p = (x,y,z) in D,
    for all r in R,
        increment H(sphere with center at (x,y,z) and radius r) by 1.
```

In actual implementation, the accumulators are defined in terms of intersections of spheres which represent possible loci of rotated and translated P_0's. Intuitively, the rotation invariant 3-D Hough transform is computed by keeping accumulators for points of intersection of the various spheres centered at all the detected points. Then the location in H having the maximum value corresponds to the translated and rotated

position of the reference point P_0, of the object model. However, if there exists possible rotational symmetry of the object model, there is no guarantee of a unique maximum in the accumulator array H. In this case, any of the maxima may be chosen.

Recall that P_0, the model reference point, is of distinct distance from all vectors in R. Call the possible location of the transformed point P_0', (x_0',y_0',z_0'). With the finding of the possible location of the transformed P_0, construct $R'=\{(dx_i,dy_i,dz_i)\}$, $i=1,m$, where $dx_i=x_0'-x_i$, $dy_i=y_0'-y_i$, and $dz_i=z_0'-z_i$, and (x_i,y_i,z_i) in D. Matching is done by finding for each r' in R', its counterpart r in R, such that r'=r. Then the detected point in D which gives rise to r' corresponds to the possible transformed location of the model point in P which gives rise to r. The match is reported in terms of the transformation that maps the model points to the detected points.

3. MKS Modes of Operation

In terms of these three major parts, MKS operates in two distinct modes:

1. Configuration Time. Sensors are specified, the low-level representation is determined, and high-level object models are described and stored in a model database.

2. Execution Time. The data from the sensors is organized in terms of low-level data structures, model-based descriptions of the data are constructed, and finally, the object models are matched to the descriptions derived from the sensor data.

An overview of the configuration time view of MKS has been given by Henderson and Wu [15]. Figure 3-1 shows the relation between the configuration time view and the execution time view of MKS.

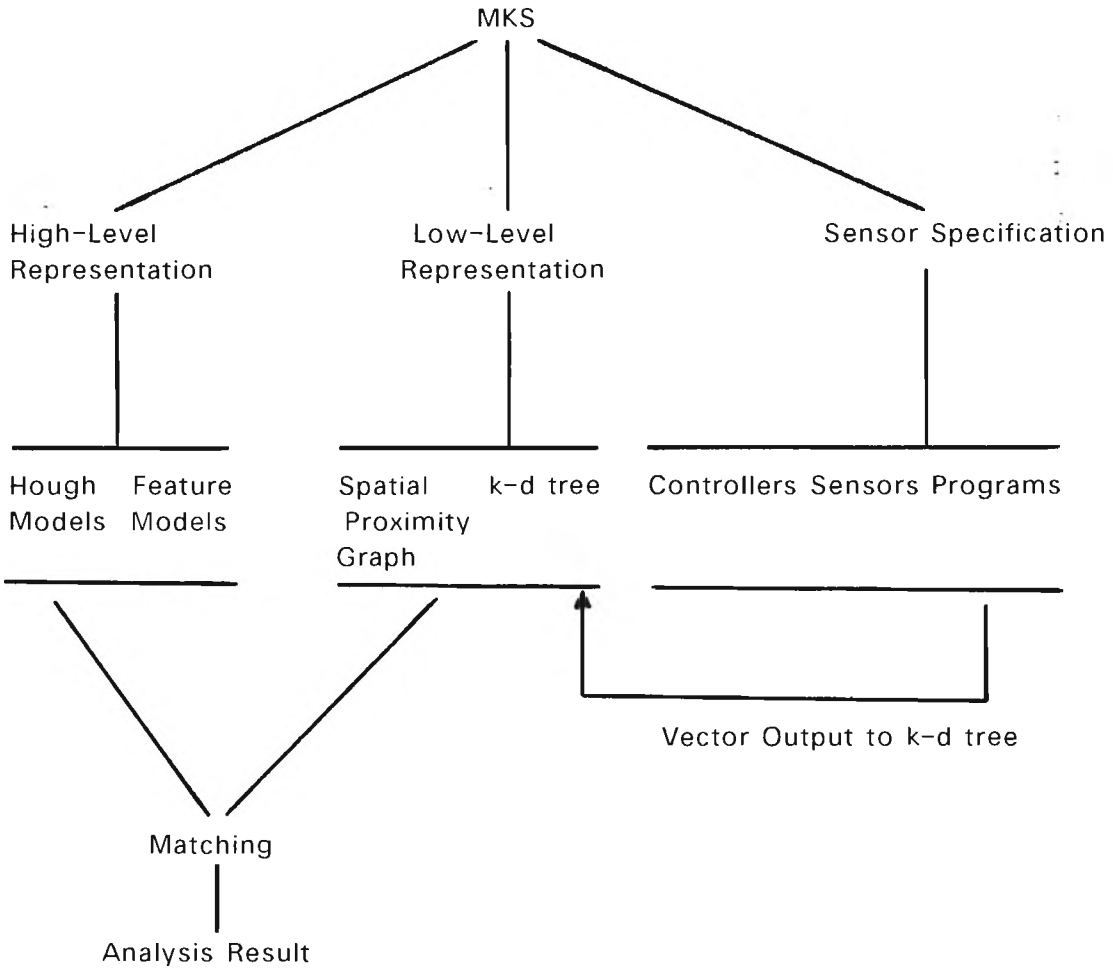


Figure 3-1: Organization of MKS

Obviously, the system is driven by the arrival of data from the actual sensors. Currently, the data from one or more sensors can be analyzed as a sequence of snapshots of data. If more than one sensor is providing data to the system, then several instances of the system run concurrently; each handles a specific model. Moreover, rather than analyze data continually as it is received, the data is collected in a buffer until the buffer is filled, then the data in the buffer is analyzed. If more data arrives, it can either be discarded, or integrated into existing data structures; however, it is computationally expensive to add data to the existing data structures, since the k-d tree must be completely rebuilt when new data is added.

The sensor provides output vectors which are then formulated into the appropriate k-tuple for low-level processing. This may involve a normalization step to weight the data more evenly than the raw feature values allow. For example, given the two features: number of holes and perimeter length (in pixels), it is clear that a difference of one hole

is much more significant than a difference of one in the perimeter length in pixels. At the present time, this problem is handled by normalizing the ranges of the features and assigning weights to the various features, however, it would be more convenient to allow a decision tree or some other form of logical analysis of the relations between features.

The k-tuples provided by the formatting step are then built into a k-d tree [8, 15]. This provides an efficient method for recovering the spatial structure of the data. This step is accomplished by querying the k-d tree with each data point in turn and creating a graph linking the m nearest neighbors to each query point. Note that the graph is not necessarily symmetric. Moreover, the k-d tree itself can be used as the basis for feature modeling, and therefore in some cases the spatial proximity graph need not be built.

In general, the matching program is allowed to query both the high-level model database and the low-level representations constructed by MKS (see Figure 2-1). The models are compared, one after the other, with the descriptions derived from the low-level data structures. The ideal world can be anything from a set of particular feature values which characterize an object to a computer aided design 3-D model. The world model generated would be the normalized vector and an image depicting a rendering of the object (given lighting, surface characteristics, etc.), respectively. The matcher can then compute any distance function desired to compare the world model and the world description, for example, a simple distance function on vectors regarded as points in Euclidean k-space, or an algorithm to solve the subgraph isomorphism.

Clearly, the results of the analysis can be used in a feedback loop to control the acquisition of new data or to manipulate the environment in some way. For example, once a 3-D object is localized in space, i.e., its exact position and orientation are known, then a manipulator can be moved to that location to grasp it. In our view, the localization of 3-D objects will most often be done with non-contact sensors; however, once an attempt at grasping is begun, then contact sensors will play a crucial role in grasping and manipulating the object. Our goal is the application of MKS to a robotics workstation having several cameras, range finders, robot arms, and dextrous hands with multiple contact sensor pads in the fingers.

The analysis of the data performed by MKS is essentially a cycle of data collection, data organization, data analysis and environment manipulation. MKS can be integrated into

such a workstation as shown in Figure 3-2.

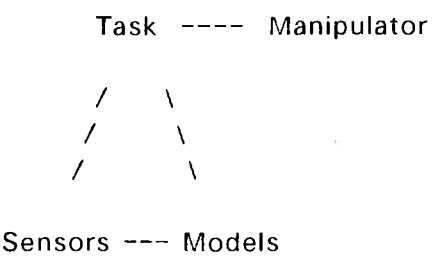


Figure 3-2: Sensor Integration Into Task Environment

The definition of the matcher, as well as the actions to take based on its result, are given within the task. The basic interface to MKS occurs through public functions allowed on the abstract data types: the k-d tree and spatial proximity graph.

4. Case Studies

Next we present the execution flow of the essential processes involved in object recognition and localization within the framework of our multi-sensor system. It also shows some sample runs of object identification based on a feature-based high-level object modeling technique and the Hough shape modeling technique.

As explained in the previous section on high-level modeling, the type and attributes of the object model, from a specific 3-D object modeling technique, determine the type and attributes of the world description of the sensed data. Therefore, a particular 3-D object model initiates the process of object recognition through the following mechanism. Once the object model is chosen, it controls the calling parameters of the k-d tree building and the spatial proximity graph builder. This is because the world description is derived from the spatial proximity graph, and it also is directly in terms of the object model. For example, if we are working with a feature model to detect circles of radius two units, then a feature vector on the k-d tree is a singleton formed by the magnitude of the radius of a circle. Moreover, the specific distance function associated with this particular feature model also determines the dissimilarity function used to construct the k-d tree and its related spatial proximity graph.

Upon closer examination, the influence of the object model does not stop at the level of the execution of the k-d tree builder. It propagates down to the choice of which sensors must be activated to acquire the corresponding features from the sensed world.

As a result, we choose to run in our multi-sensor environment by invoking a certain object model. The flow of the execution of the multi-sensor framework is as follows:

1. Invocation of a high-level object model to be detected which automatically fixes the dimension of the k-d tree and the dissimilarity function used in constructing both the k-d tree and the spatial proximity graph.

2. Activation of the types of logical sensors which are required to acquire the features corresponding to the high-level object model.

3. Organization of the sensed data into a k-d tree structure.

4. Construction of, if necessary, the spatial proximity graph from the k-d tree.

5. Grouping, if necessary, on the spatial proximity graph to obtain features required by the high-level object model.

6. Matching of the detected world description to the world model.

7. Report of the possible transformations that map the model object to the detected object in the case of Hough shape model.

To provide data for the following case studies, we have digitized three scenes with an RCA camera. We refer to these scenes as Scene 1, Scene 2, and Scene 3, as shown in Figure 4-1, Figure 4-2, and Figure 4-3, respectively. Scene 1 gives a typical 2-D type of scene, and object descriptions are given in terms of the 2-D silhouette. Scene 2 and Scene 3 give a stereo pair view of a set of simple 3-D objects. These scenes and the discussion given are intended to illustrate the ways in which the system can be used.

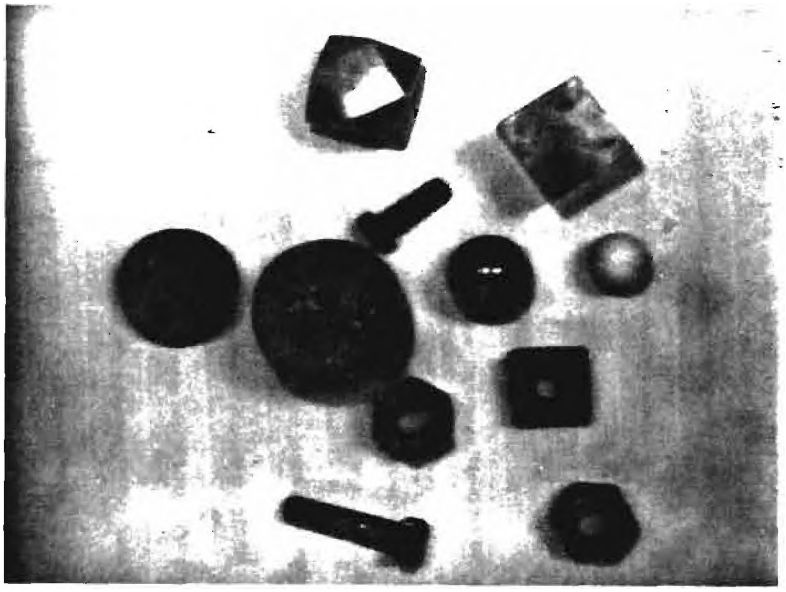


Figure 4-1: Scene 1, typical 2-D scene

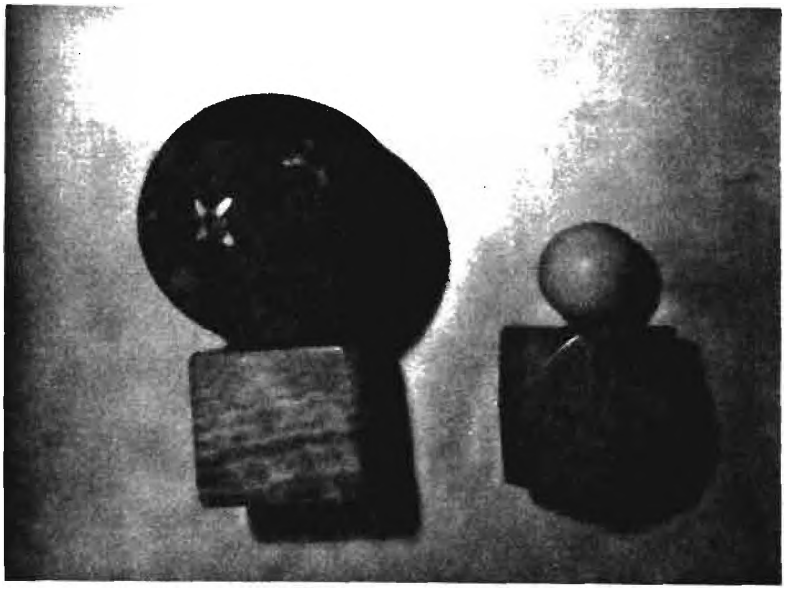


Figure 4-2: Scene 2, a set of simple 3-D objects, stereo view 1

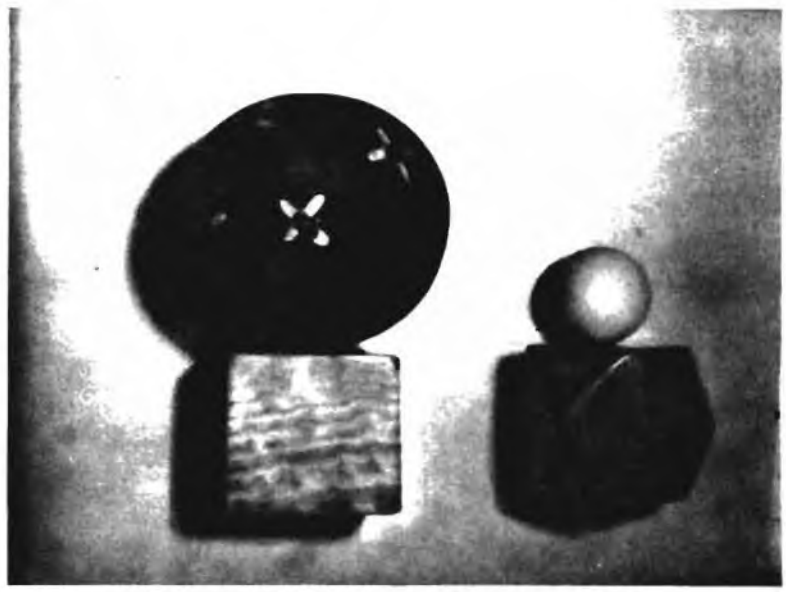


Figure 4-3: Scene 3, a set of simple 3-D objects, stereo view 2

4.1. Feature Models

Within the multi-sensor framework, we define each feature model directly in terms of a logical sensor. Every logical sensor has a characteristic output vector which contains all the features detected by the sensor. However, sometimes only a subset of the possible features is used for matching. The k-d tree builder only organizes sets of these k features on the k-d tree. For matching, we incorporate the model reference vectors into the detected data to build the spatial proximity graph. In the spatial proximity graph, if a detected vector matches a certain reference vector, they will be neighbors. Therefore, in building the spatial proximity graph, it is merely necessary to specify the threshold dissimilarity associated with the model, and an infinite number of neighbors (this is due to the lack of advance knowledge of the number of detected objects which match the model).

The following three examples illustrate the feature-based object modeling technique.

Circle Model

We use the logical sensor called "Circle_Detector" to analyze Scene 1. The output of "Circle_Detector" is vectors of the form (x_i, y_i, r_i) where x_i, y_i are the coordinates of the

centroid of the i^{th} detected object, and r_i is the radius of the detected object. The model reference vector is:

("don't care", "don't care", 2).

"Don't care" indicates that the field it appears in is not to be used as part of the model definition. We include the model reference vector at the beginning of the file containing the detected vectors. Table 4-1 gives a set of vectors input to the k-d tree builder.

Object #	(x-location,	y-location,	radius)
0	"don't care",	"don't care",	2
1	256,	86,	0
2	371,	137,	0
3	269,	171,	0
4	397,	223,	1
5	320,	224,	2
6	128,	240,	3
7	230,	257,	5
8	358,	309,	0.5
9	333,	343,	0.5
10	256,	429,	0
11	384,	429,	0.5

Table 4-1: Vectors produced by "Circle_Detector"

Table 4-2 gives the resultant spatial proximity graph. The integers on the i^{th} row are the indexes of the neighbors of the i^{th} object. The number of nearest neighbors chosen is 12, and the distance threshold is 0.1.

Object #	Neighbors
0	5
1	2 3
2	1 3
3	1 2
4	-
5	0
6	-
7	-
8	9 10
9	8 10
10	8 9
11	-

Table 4-2: Spatial Proximity Graph for the Circle Radii

Since the 5^{th} detected vector is the only one with radius 2, the reference vector and this vector are neighbors.

Nut Model

We use the logical sensor called "Corner_Holes_Detector" to analyze Scene 1. The output of "Corner_Holes_Detector" is vectors of the form (x_i,y_i,c_i,h_i) where x_i,y_i are the coordinates of the centroid of the i^{th} detected object, c_i is the number of corners detected in the object, and h_i is the number of holes detected. The model reference vector is:

("don't care", "don't care", 4, 1).

Table 4-3 gives a set of vectors input to the k-d tree builder.

Object #	(x-location,	y-location,	corners,	holes)
0	"don't care",	"don't care",	4,	1
1	256,	86,	6,	0
2	371,	137,	4,	0
3	269,	171,	8,	0
4	397,	223,	0,	0
5	320,	224,	0,	0
6	128,	240,	0,	0
7	230,	257,	0,	0
8	358,	309,	4,	1
9	333,	343,	6,	1
10	256,	429,	8,	0
11	384,	429,	6,	1

Table 4-3: Vectors Produced by "Corners_Holes_Detector"

Table 4-4 gives the resultant spatial proximity graph. The integers on the i^{th} row are the indexes of the neighbors of the i^{th} object. The number of nearest neighbors chosen is 12.

Object #	Neighbors
0	8
1	-
2	-
3	10
4	5 6 7
5	4 6 7
6	4 5 7
7	4 5 6
8	0
9	11
10	3
11	9

Table 4-4: Spatial Proximity Graph of Nut Data

Since the 8^{th} detected vector is the only one with four corners and one hole, the reference vector and this vector are neighbors.

Planar Surface Model and Curved Surface Model

We use the logical sensor called "Surface_Curvature" to analyze the sphere-cube pair in Scene 2 and Scene 3. (We assume that the cube and the sphere atop it have been separated out from the other objects in the scene, e.g., by the connectivity of the spatial proximity graph based on the 3-D surface points.) The output of "Surface_Curvature" is vectors of the form: (x_i,y_i,z_i,n_i) where (x_i,y_i,z_i) are the coordinates of the i^{th} detected object, and n_i is the encoded curvature at the i^{th} surface point. For a curved surface, n_i is encoded as "1". For a planar surface, n_i is encoded as "-1". The planar surface model reference vector is:

("don't care", "don't care", "don't care", -1).

The curved surface reference vector is:

("don't care", "don't care", "don't care", 1).

Table 4-5 gives a small sample of the set of vectors input to the k-d tree. The first two entries in the table are the reference vectors, then the next few lines give some sample points from the sphere, and finally, the last few lines give some sample points from the cube.

Point #	(x-location,	y-location,	z-location,	curvature)	
0	"don't care",	"don't care",	"don't care",	-1	Planar Model
1	"don't care",	"don't care",	"don't care",	+1	Curved Model
2	0.0,	0.0,	3.00,	+1	
3	-1.76,	0.0,	2.43,	+1	
4	2.31,	1.68,	0.93,	+1	Sphere Points
5	-2.31,	-1.68,	0.93,	+1	
6	2.31,	1.68,	-0.93,	+1	
7	-3.0,	-6.0,	-3.0,	-1	
8	-3.0,	-6.0,	3.0,	-1	
9	3.0,	-6.0,	0.0,	-1	Cube Points
10	0.0,	-6.0,	0.0,	-1	
11	3.0,	-6.0,	3.0,	-1	

Table 4-5: Vectors Produced by "Surface_Curvature"

Table 4-6 shows the corresponding spatial proximity graph for the above input vectors.

Point #	Neighbors	
0	7,8,9,10,11	Planar Model
1	2,3,4,5,6	Curved Model
2	1,3,4,5,6	
3	1,2,4,5,6	
4	1,2,3,5,6	Sphere Points
5	1,2,3,4,6	
6	1,2,3,4,5	
7	0,8,9,10,11	
8	0,7,9,10,11	
9	0,7,8,10,11	Cube Points
10	0,7,8,9,11	
11	0,7,8,9,10	

Table 4-6: Spatial Proximity Graph for Planar and Curved Surface Points

The neighbors lists for objects 0 and 1 give the planar surface points and the curved surface points, respectively. The spatial proximity graph (SPG) actually divides the set of detected vectors into one set with planar surface and the other set with curved surface. If we use this information to build separate spatial proximity graphs from the two distinct sets of points (see Figure 4-4 and Figure 4-5) for the two types of objects, we get the results shown in Figure 4-6 and Figure 4-7. We can use the Hough shape modeling technique to find out what type of polyhedra the object with the planar surface is, if necessary.

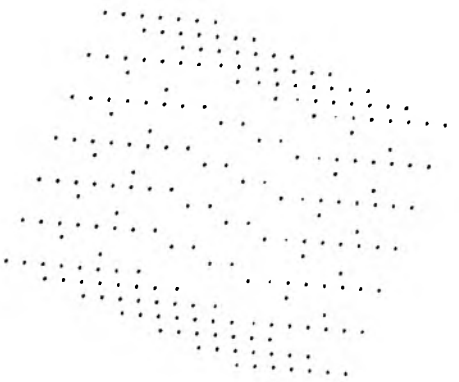


Figure 4-4: Surface points on the cube

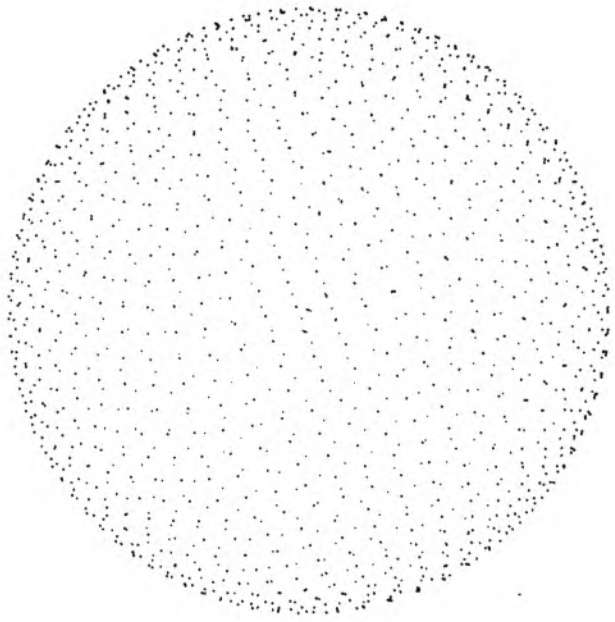


Figure 4-5: Surface points on the sphere

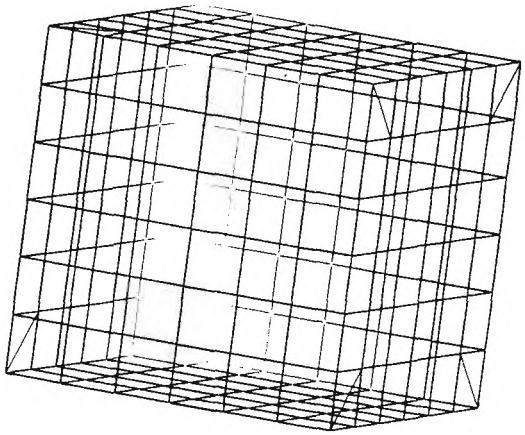


Figure 4-6: SPG for a cube with 4 nearest neighbors

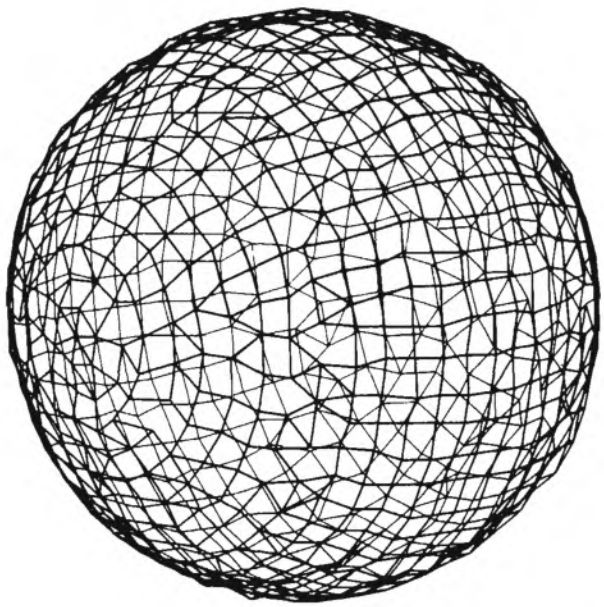


Figure 4-7: SPG for a sphere with 4 nearest neighbors

4.2. Hough Shape Model

To directly apply the Hough shape transform to a model of a 3-D object in terms of its surface points would require a 3-D accumulator array, as discussed in section 2.2, which could easily exhaust the memory of a machine. Therefore, it is necessary to compress the size of the model representation. We recommend two approaches, illustrated by two examples, which drastically reduce the set of accumulators.

The first approach works well with polyhedral models. We reduce the size of the model representation by using the vertexes of the polyhedron to represent it.

The second approach is suitable for objects of irregular shape. This approach consists of choosing four control points which are recoverable from the type of data available, and determining the geometric transformation from the model to the detected control points. If the control points are not directly distinguishable, e.g., they are all vertexes of the same order, then the Hough shape transform can be used to label them; otherwise, the transformation can be determined directly.

Cube

We would like to use a cube as the model object to illustrate the first approach. The logical sensor used is "Range_Finder". The model is given in terms of :

reference point: (1.5, 1.3, 1.25),

and radii of magnitude :

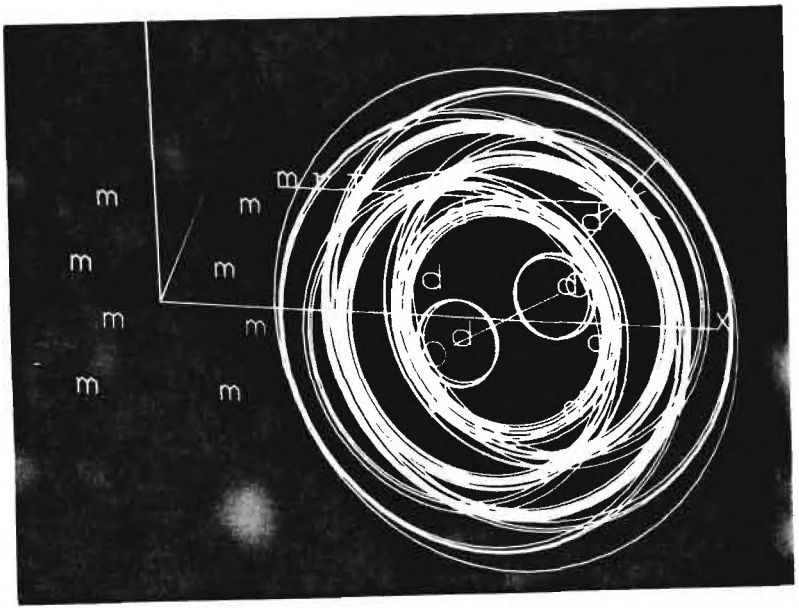
4.09,
3.43,
3.38,
3.28,
2.54,
2.40,
2.33,
0.65.

Figures 4-4 and 4-6 show the detected surface points of the cube and the corresponding spatial proximity graph obtained, respectively, as displayed on the PS300. After performing grouping on the spatial proximity graph to find the faces of the cube and intersecting the different faces, we found the vertexes given in Table 4-7.

x	y	z
4.0	-1.0	1.0
4.0	1.0	1.0
6.0	1.0	1.0
6.0	-1.0	1.0
4.0	1.0	-1.0
4.0	1.0	-1.0
6.0	1.0	-1.0
6.0	-1.0	-1.0

Table 4-7: Vertexes of the Cube

Figure 4-8 shows the initial set of accumulators when detecting the cube.



m's denote model vertexes

mrf denotes model reference point

d's denote detected vertexes

* denotes detected reference point

Figure 4-8: Initial circular accumulators when detecting cube

Since there are rotational symmetries in the cube with respect to the chosen model reference point, we actually detected eighteen possible locations for the model reference point. Here is the one we picked, and along with it is the corresponding transformation that maps the model cube to the detected cube:

detected reference point: (6.5, 1.3, 1.25)

transformation matrix:

$$\begin{bmatrix} 1. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 5. & 0. & 0. & 1. \end{bmatrix}$$

In this case the detected cube is an instantiation of the model cube which has simply been translated by 5 units in the positive direction along the x-axis.

Renault Piece

We would like to use the industrial object shown below as the model object to illustrate the second approach. The logical sensor used is "Range_Finder". There are about 2000 point samples. We call this piece the Renault piece. Figure 4-9 shows the detected surface points of the Renault piece. (This data was digitized on a laser range finder developed at INRIA, Rocquencourt, France by F. Germane.) Figure 4-10 is the spatial proximity graph obtained as displayed on the Evans and Sutherland PS300.

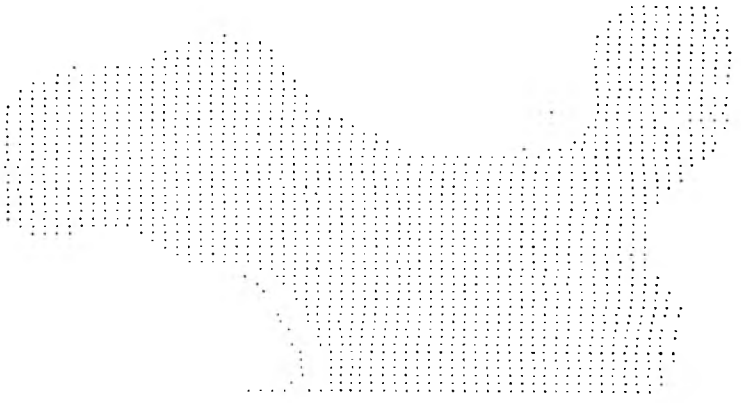


Figure 4-9: Detected surface points of the Renault piece

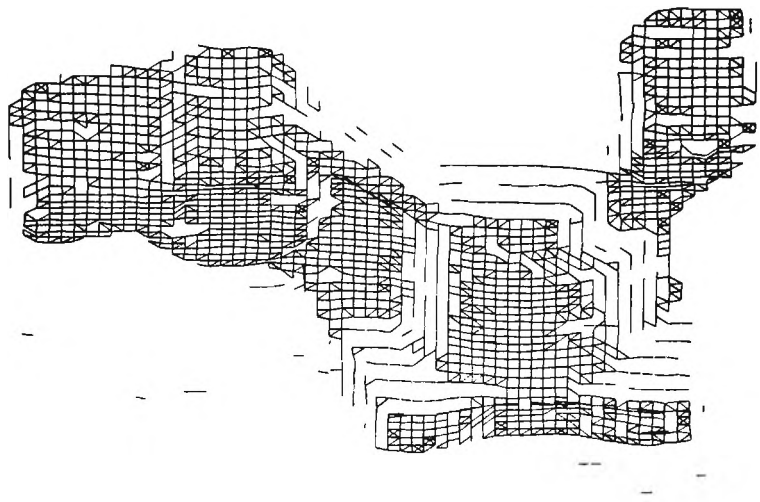


Figure 4-10: SPG of the Renault piece with 4 nearest neighbors

The four control points chosen as model points could be derived from the spatial proximity graph of the model. The model is given in terms of:

model reference point (10.344000, 19.333334, 25.122999)

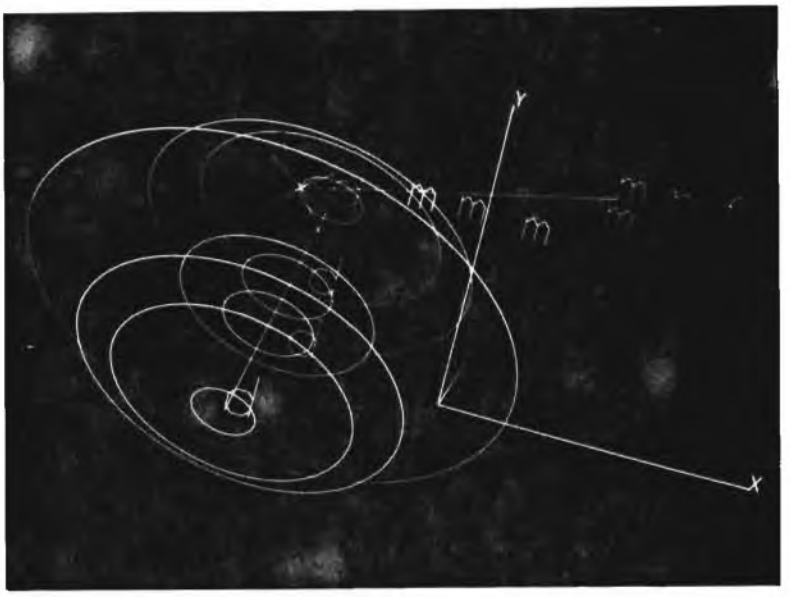
and the radii of magnitude:

29.900755,
18.041958,
13.079391,
3.379884.

The detected control points are:

(-21.000000,-7.246000,2.706000),
(-17.600000, -3.183000, 16.096001),
(-17.200001, 3.755000, 16.305000),
(-18.000000, 10.844000, 25.872999).

Figure 4-11 shows the initial set of accumulators using the four control points for detection.



m's denote model vertexes

mrf denotes model reference point

d's denote detected vertexes

* denotes detected reference point

Figure 4-11: Initial circular accumulators when detecting Renault piece

Here is the transformation that the maps the model object to the detected object:

```
|  0.000000   1.000000  -0.000000  0.000000 |
| -1.000000  -0.000002   0.000002  0.000000 |
| -0.000000  -0.000000   1.000001  0.000000 |
|_-2.000000   1.000031   0.999962  1.000000 _|
```

In this case, the detected object is an instantiation of the model object after it has been translated by (1.0, 2.0, 1.0) and rotated by 90 degrees with respect to the z-axis.

5. Conclusions and Further Research

The multi-sensor integration and data acquisition system which is of interest to us is configured by defining the sensors, the low-level representation, and the high-level modeling techniques. The specified task description dictates when new sensor data is required and how it should be obtained. Matching models to descriptions can take place

in the task. Based upon the results of the analysis on the data, the objects or the environment can be manipulated.

The primary goal of this research was to develop a low-level representation of the real world phenomena and to integrate that representation into a meaningful interpretation of the real world. We accomplished this goal by developing the spatial proximity graph as the low-level representation and incorporating it with two high-level modeling methods, a feature-based modeling method, and the Hough shape modeling method.

We extended certain previous researchers' 2-D low-level representations, such as the region adjacency graph, to a realistic and flexible 3-D low-level representation, the spatial proximity graph. We can handle not only 3-D visual information, but we can also manage 3-D nonvisual information, such as tactile information from a robot hand. In using the Hough shape model, we introduced a new extension, the rotational invariant 3-D Hough shape model, to the classic Hough shape model which was originally designed to handle 2-D shapes without any simple analytical form.

Concerning the issue of object localization, we must take into consideration treatment of errors in detected data, strategies which are best for acquiring new information for object determination, and measures to disambiguate situations such as multiple objects which are similar in one view but different in actuality. In matching we define a dissimilarity measure, as mentioned in section 2.1, between the model and the detected object. The distance function for each vector dimension, and the overall tolerance in the matching function should be tailored to take into account errors in the detected data. In gathering new information to complete object determination for partially recognized objects, the strategy is to activate the appropriate sensors for missing features. In distinguishing objects which are similar in one view but different in actuality, information should be gathered from more than one view so as to capture fully the three dimensionality of the target environment.

Based upon case studies with our framework for a multi-sensor system in a simulated environment, we observe the following in relation to computation speeds for real-time hand manipulation. Before manipulation can occur, objects must be localized. Object localization involves organization of sensor systems through controllers and actuators to achieve a smooth flow of data in a multi-sensor environment, organization of sensor data

into their corresponding spatial proximity graphs, and matching between the world descriptions and the high-level models. Since the spatial proximity graph can be efficiently constructed by an algorithm of order(nlogn) time complexity, the possibility of real-time hand manipulation is constrained by the efficiency both of the sensors in supplying features of the detected environment, and of the high-level modeling technique used in matching. Another way of considering this issue of real-time manipulation in connection with the tactile sensors is as follows. The robot hand and other sensors, such as a zooming device, which are mechanical devices, consume time when being positioned from one physical location to another. Consequently, the amount of sensor data output in a small time interval will most likely be of a manageable quantity which can be organized into a coherent low-level representation by our system. Some of the most important areas for further research are presented in the following paragraphs.

Of crucial importance to building up-to-date spatial proximity graphs to organize a continuous flow of a massive amount of sensor data is the ability to dynamically insert and delete data on a k-d tree or any equivalent database storage structure that allows efficient query and searching to be performed on the data. Overmars and van Leeuwen [19] have presented some initial work on dynamic multi-dimensional data structures, and the usefulness of their results to our application must be investigated.

Another important area for research concerns the logical sensor system. Physical sensors are defined by parameters associated with the individual sensor of some known class, e.g., TV cameras, tactile pads, etc. Logical sensors are defined in terms of physical devices, and algorithms on their data, e.g., an "Edge_finder" can be defined in terms of a tactile pad on a robot hand, and a pressure analysis program. As described in section 1.1, associated with each sensor is a characteristic output vector which defines exactly the name and the allowed range of data. With these characteristic output vectors, we can combine vector elements of same name but different allowed range by some appropriate coercion, such as forcing a higher precision range to become a lower one. With this mechanism of treating the "same" kind of data from different precision sensors, in addition to the uniform sensor output, namely a feature and its 3-D location, we can integrate sensor data from different sensors with the logical sensor system. An easily reconfigurable sensor system certainly facilitates the acquisition and derivation of appropriate features needed for the construction of a low-level world description which is

defined in terms of a high-level model description.

A third area of further research pertains to the investigation of structural modeling techniques which allow the automatic derivation and exploitation of constraints which can then be used to control the acquisition of data, in terms of limiting the amount of data to acquire and specifying the type of data to acquire. It would be interesting to know what kinds of constraints can be derived from such representation methods as generalized cylinders, shape grammars, and relational networks.

We picked the Hough shape model for its simplicity as our first attempt. We would like to suggest a closer examination of a possible improvement to our current vertex-based implementation. When dealing with polyhedral models, an alternative to using vertexes to represent the polyhedral faces of a 3-D object is to map each face of the 3-D object into a 4-D transform space, and model these points considered as an object. The 4-D points are the coefficients of the planes that contain the faces of the polyhedron. Since the number of faces is usually small, we can keep the storage needed for accumulators under control. However, the difficulty of such an approach lies in interpreting the geometric meaning and geometric correspondence of the detected reference point found to the model reference point. An undesirable consequence of lacking a clear understanding of the geometric meaning and geometric correspondence is that the orientation of the recognized object will remain unknown.

In order to have a versatile and robust pattern recognition system, the class of objects to model should include, besides polyhedra, objects with curved surfaces. The inclusion of such objects with curved surfaces naturally opens up avenues for research in determining relevant models which in turn controls the kind of features appropriate to be acquired through the low-level representation for object recognition. This may offer some exciting research in conjunction with current research on representing curved surfaces analytically using splines for instance.

Since we hope to operate our multi-sensor framework in the context of a robot workstation, the list of possible research areas will be incomplete without including research on manipulation of objects. The idea in abstract is that once object recognition and localization are achieved, how should manipulation be performed? The current system provides the necessary first step, namely object recognition and localization, for

performing such manipulation tasks.

References

[1] Ahuja, N.
Dot Pattern Processing Using Voronoi Neighborhoods.
IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-4(3):336-343, May, 1982.

[2] Ballard, D.H.
Generalizing the Hough Transform to Detect Arbitrary Shapes.
Pattern Recognition 13(2):111-122, 1981.

[3] Ballard, D.H. and C.M. Brown.
Computer Vision.
Prentice Hall, New York, 1982.

[4] Barrow, Harry and Jay Tennenbaum.
Recovering Intrinsic Scene Characteristics from Images.
Technical Report 157, SRI International, April, 1978.

[5] Davis, L.S. and S. Yam.
A Generalized Hough-like Transformation for Shape Recognition.
Technical Report TR-134, U. of Texas, FEB, 1980.

[6] Duda, R.O. and P.E. Hart.
Use of the Hough Transform to Detect Lines and curves in Pictures.
Communications of the ACM 15:11-15, January, 1972.

[7] Foley, J.D. and A. Van Dam.
Fundamentals of Interactive Computer Graphics.
Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.

[8] Friedman, J.H., J.L. Bentley and R.A. Finkel.
An Algorithm for Finding Best Matches in Logarithmic Expected Time.
ACM Trans. on Math. Soft. 3(3):209-226, September, 1977.

[9] Henderson, T.C.
An Efficient Segmentation Method for Range Data.
In SPIE Conference on Robot Vision, pages 46-47. Arlington, VA, May, 1982.

[10] Henderson, T.C.
Feature Based 2-D Shape Models.
In O.D. Faugeras (editor), Vision par Ordinateur, pages 215-224. INRIA-Roquencourt, June, 1982.

[11] Henderson, T.C. and N. Keskes.
L'analyse des Objects Tridimensionnels.
In Proc. 3rd Conf. on Patt. Recog. and Art. Intell., pages 277-287. Nancy, France, September, 1981.

[12] Henderson, T.C. and A. Mitiche.
Modeling 3-D Structure.
In B. Radig (editor), Modelle und Strukturen, pages 112-116. Springer-Verlag, Berlin, October, 1981.

[13] Henderson, T. and E. Triendl.
Storing Feature Tree Descriptions as 2-D Trees.
In Proceedings of Pattern Recognition and Image Processing Conference, pages 555-556. June, 1982.

[14] Henderson, T. and E. Triendl.
The k-d Tree Representation of Edge Descriptions.
In Proceedings of International Conference on Pattern Recognition. October, 1982.

[15] Henderson, Thomas C. and Wu So Fai.
A Multi-sensor Integration and Data Acquisition System.
In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 274-280. IEEE, June, 1983.

[16] Hough, P.V.C.
Method and Means for Recognizing Complex Patterns.
U.S. Patent 3069654.
1962.

[17] Marr, D.
Early Processing of Visual Information.
AI Memo 450, MIT, Cambridge Mass, December, 1975.

[18] Merlin, P.M. and D.J. Farber.
A Parallel Mechanism for Detecting Curves in Pictures.
IEEE Transactions on Computers C-24:96-98, 1975.

[19] Overmars, M.H. and Jan van Leeuwen.
Dynamic Multi-Dimensional Data Structures Based on Quad- and K-D Trees.
Acta Informatica 17:267-285, 1982.

[20] Pavilidis, T.
Structural Pattern Recognition.
Springer-Verlag, 1977.

[21] Pfaff, G., H. Kuhlmann and H. Hanusa.
Constructing User Interfaces Based on Logical Input Devices.
Computer 15(11):62-69, November, 1982.

[22] Rosenthal, D.S., J.C. Michener, G. Pfaff, R. Kessener and M. Sabin.
The Detailed Semantics of Graphics Input Devices.
Computer Graphics 16(3):33-38, July, 1982.

[23] Toussaint, G.T.
The Relative Neighborhood Graph of a Finite Planar Set.
Pattern Recognition 12:261-268, August, 1980.

[24] Wu So Fai.
A Multi-sensor Integration and Data Acquisition System.
Master's thesis, University of Utah, June, 1983.

[25] Zahn, C.T.
Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters.
IEEE Transactions on Computers C-20(1):68-86, 1971.