A Kernel for Multi-sensor Robotic Systems[1]

Thomas C. Henderson and Charles D. Hansen

Department of Computer Science
The University of Utah

ABSTRACT

Multi-processor and multi-sensor systems are being proposed to solve a wide range of problems. In particular, distributed sensing systems and general robot workstations require real-time processing of information from visual, auditory, tactile and other types of sensors. We propose the *spatial proximity graph* as a low-level representation of sensory data from diverse sources and use this as the basis for high-level organization and control over the acquisition of data. The notion of "logical" (or abstract) sensor allows for flexible hardware/software mix in terms of a multi-sensor system and permits a simple method of reconfiguation whenever logical or physical sensors are added to or removed from the system.

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 [10]. The system can be logically divided into three major parts: the sensor specification, low-level representation, and high-level modeling.

The first major goal is to provide a mechanism for the integration of data available from different sensors into a coherent low-level representation of the 3-D world. Such a representation is crucial to the successful application of multi-sensor systems, and in particular, robot technology. Shneier et al [16] argue:

> the use of easily acquired information from a number of sources can lead more easily to understanding a scene than can exhaustive analysis of an image from a single source.

---

[1]This work was supported in part by the System Development Foundation and NSF Grants ECS-8307483 and MCS-82-21750

Although their work dealt only with visual information, we heartily concur in principle and propose the spatial proximity graph as a structuring mechanism for the integration of data from different sensors.

The second major goal is to provide a simple, yet complete, method for (re)configuring a multi-sensor system. We propose the "logical" sensor as a key notion toward this end. A logical sensor maps either directly onto a physical sensor, or provides a description of how data from one or more physical sensors is combined to produce the desired data. Ultimately, such logical sensors could be implemented in special hardware (a "sensor engine").

The third major goal is to provide a context in which constraints, both physical and logical, can be brought to bear to reduce the amount of computation required to solve problems. Most research in this area is directed toward organizing the information flow between stations so as to achieve an efficient and successful interpretation of the sensed data (see also Smith [17]). Usually the stations transmit reports or evaluations of their own data rather than the raw data itself. Thus, there is a need for a high-level model to provide an interpretation of the various patterns of information provided by the sensors, and a mechanism for controlling the acquisition of data.

Although the system organization and modeling capabilities proposed here are generally applicable to multi-sensor systems, the focus is the design of a multi-sensor robot workstation. In particular, the system is presented in terms of the integration of visual and tactile data toward the goal of forming a model of the 3-D objects within the range of a robot arm.

2. Sensor Specification

Following the work of Foley [4], Pfaff [14], and Rosenthal [15] in the domain of logical graphical input device specifications, we have shown how MKS can provide a similar function in a multi-sensor environment [10]. 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.

3. 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 [11], Barrow and Tennenbaum's intrinsic characteristics [3], and to some extent, the region adjacency graph of Pavlidis [13], 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 [6, 10]. 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 [2]). Other approaches to the organization of point data include minimal spanning trees [19], relative neighborhood graphs [18], 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 [5] which is built directly from the data (see Henderson [8, 9] for the use of the k-d tree for feature organization). 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.

4. 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 1 outlines the relationships between the model, the description and the matcher.

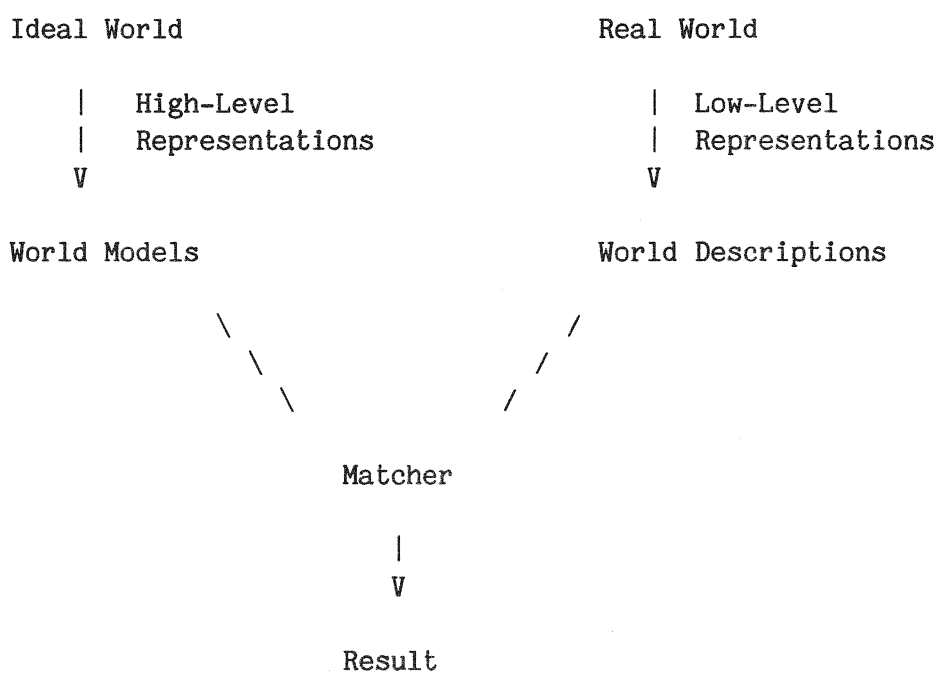

Figure 1. High-Level Modeling and Matching

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 [7]). 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.

5. 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 [10]. 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 [5, 10]. 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. 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 2.

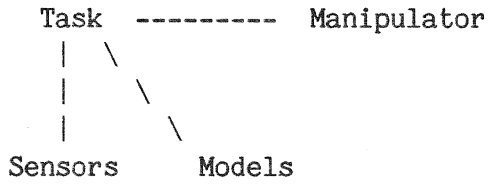

Figure 2. Sensor Integration Into Task Environment

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.

6. 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 [12] 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. and C.M. Brown.
Computer Vision.
Prentice Hall, New York, 1982.

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

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

[5] 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.

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

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

[8] 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.

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

[10] 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.

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

[12] 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.

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

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

[15] 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.

[16] Shneier, M., S. Nagalia, J. Albus, and R. Haar.
Visual Feedback for Robot Control.
In *Workshop on Industrial Applications of Industrial Vision*, pages 232-236.
IEEE, May, 1982.

[17] Smith, R.G.
A Framework for Distributed Problem Solving.
UMI Research Press, 1981.

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

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