Sensory Interactive Robots*

James S. Albus**, Anthony J. Barbera**, M. L. Fitzgerald**, and Marilyn Nashman**
Submitted by R. D. Young (1), NBS, Washington, D.C. 20234

For robots to operate effectively in the partially unconstrained environment of manufacturing, they must be equipped with control systems that have sensory capabilities. This paper describes a control system that consists of three parallel cross coupled hierarchies. First is a control hierarchy which decomposes high level tasks into primitive actions. Second is a sensory processing hierarchy that analyses data from the environment. Third is a world model hierarchy which generates expectations. These are compared against the sensory data at each level of the sensory processing hierarchy. Deviations between expected and observed data is used by the control hierarchy to modify its task decomposition strategies so as to generate sensory-interaction goal-directed behavior.

This system has been implemented on a research robot, using a network of microcomputers and real-time vision system mounted on the robot wrist.

I. Introduction

In order for robots to operate effectively in the partially unconstrained environment of manufacturing, they must be equipped with control systems that have measurement and sensory capabilities. But more than just sensory data is required. The data must be processed and analyzed and the results introduced into the robot control system in real-time so that the response is goal-directed, reliable, and efficient. This is a problem in which complexity grows exponentially with the number of sensors and with the number of branch points in the control program. Once there are more than a few sensors, each producing data which can modify the robot's behavior or select among a number of optional behavioral patterns (or trajectories), control programs can become enormously complex to write and virtually impossible to debug.

It is at this point that the problem of controlling a sensory interactive robot becomes similiar to that of controlling any complex system such as an army, a government, a business, or a biological organism. It then becomes necessary to introduce the type of hierarchical command and control structure which has historically proven itself successful in controlling such systems [1]. The secret of hierarchical control is that it allows the problem to be partitioned so as to limit the complexity of any module in the hierarchy to manageable limits regardless of the complexity of the entire structure.

II. Hierarchical Control

a) The Organizational Hierarchy

Figure 1 illustrates the basic logical and temporal relationships in a hierarchical computing stucture. On the left is an organizational hierarchy wherein computing modules are arranged in layers like command posts in a military organization. Each computing module receives commands from a superior as well as feedback from subordinates and the environment. It then computes an appropriate string of subcommands which are issued to a small and exclusive set of immediate subordinates. In this way a high level goal is successively decomposed into a number of coordinated strings of action primitives which produce observable behavior in accomplishing the goal [2]. In a single highest level module, the goals are selected and the discussions are made which commit the entire organization to coordinated action.

b) The Computational Hierarchy

Each chain-of-command in the organizational hierarchy consists of a computational hierarchy of the form shown in the center of Figure 1 [3]. This computational hierarchy contains three parallel hierarchies: one, a task decomposition hierarchy which decomposes high level tasks into low level actions; two, a sensory processing hierarchy which processes sensory data and extracts the information needed by the task decomposition modules at each level; and three, a world model hierarchy which generates expectations of what sensory data should be expected at each level based on what subtask is currently being executed at that level.

c) The Behavioral Hierarchy

The input commands to each of the levels in the computational hierarchy can be represented as vectors which trace out trajectories through state-space as time progresses. This creates a behavioral hierarchy as shown on the right of Figure 1. The lowest level trajectories in the behavioral hierarchy correspond

*Contribution of the U.S. National Bureau of Standards, not subject to copyright.

**Industrial Systems Division, Center for Manufacturing Engineering, U. S. National Bureau of Standards, Washington, D.C. 20234.

to obserable output behavior, e.g. the time history of the joint position of a robot manipulator. All the other trajectories constitute the deep struction of behavior. For example second level trajectories correspond to x,y,z position, velocities, and forces of the robot hand. The third level trajectories correspond to sequences of symbolic names of elemental moves such as "reach", "grasp", etc. The fourth level and fifth trajectories corrspond to sequences of simple and complex task names, respectively.

d) Sensory Processing

The sophisticated real-time use of sensory data for coping with uncertainty and recovering from errors requires that sensory information be able to interact with the control system at many different levels with many different constraints on speed and timing. For example, joint position, velocity, and sometimes force measurements are required at the lowest level in the hierarchy for servo feedback. This data requires very little processing, but must be supplied with time delays of only a few milliseconds. At the action primative level, x,y,z position, velocity, and force data are needed. This requires a coordinate transformation. Visual depth (proximity) and information related to edges and surfaces are also needed at this level to compute offsets for gripping points. This data requires a modest amount of processing and must be supplied within a few tenths of a second. Recognition of part position and orientation requires more processing and is needed at the elemental move level where time constraints are on the order of seconds. Part identity and relationships between objects is required for making behavioral decisions at the simple task level. In general, sensory information at the higher levels is more abstract and may require the integration of data over longer time intervals. However, behavioral decisions at the higher levels need to be made less frequently, and less quickly, and therefore can tolerate the greater amount of sensory processing required.

Attempting to deal with this full range of sensory feedback in all of its possible combinations at a single level leads to extremely complex and inefficient programs. The processing of sensory data, particularly vision data, is inherently a hierarchical process [4]. Only if the control system is also partitioned into a hierarchy can the various levels of feedback information be introduced into the appropriate control levels in a simple and straightforward manner.

e) The World Model

Typically, the type of information required by the control system depends upon what task is being performed. As conditions change, different sensors, different resolutions, and different processing algorithms may be needed. Furthermore, sensory data can often be predicted from the actions being executed by the control system. The world model hierarchy may contain information as to the shape, dimensions, and surface features of parts and tools and may even indicate their expected position and orientation in the work environment. This information enables the sensory processing modules to select processing algorithms appropriate to the expected incoming sensory data, and to correlate observations against expectations [5,6]. The sensory processing system can thereby detect the absence of expected events and measure deviations between what is observed and what is expected.

Feedback can be used by the task decomposition hierarchy either to modify action so as to bring observations into correspondence with expectations, or to change the input to the world model so as to pull its expectations into correspondence with observations [7]. In either case, once a match is achieved between expectation and observation, the task decomposition hierarchy can act on information contained in the model. This means the robot can now act on the basis of knowledge which is more complete than can be derived from sensory observations. For example, a robot control system may use model data to reach behind an object and grasp a point which is hidden from view.

III. Programming a Hierarchical Control System

At each level in the task decomposition hierarchy, the string of commands flowing between the H submodules defines a program. This implies that there is a programming language unique to each level of the computational hierarchy, and that the procedures executed by the computing modules at each level are written in a language unique to that level. This partitioning of the control problem into hierarchical levels limits the complexity of the programs programs at each level.

a) State-Machine Representation

If we now further partition the robot control problem along the time axis, we can represent each computational submodule as a state-machine [7]. At every time interval each computational submodule samples its inputs (command and feedback), and computes an output. The programs resident in each of the computational submodules then become simple functions which can be represented by formulas of the form $P=H(s)$, by a set of productions, or rules, of the form IF (S)/THEN (P), or by a state-transition table.

One method of implementing the H, M, and G submodules of Figure 1 is by a state-table such as shown in Figure 2. Here the simple task <FETCH(X)> is defined. The lefthand side of the table consists of a command vector C and a feedback vector F. The C vector consists of a command FETCH and an argument X. The F vector consists of a state defined by the previous output plus a set of feedback variables consisting of processed sensory data from the external environment as well as progress reports from lower level modules. The righthand side of the state table defines (or points to a procedure which defines) an output vector P which contains commands to lower level H modules, next state information to be used internally, and action context information to be sent to M submodules. The action context information addresses the M submodules which retrieves expectations to be sent to the G submodules. In this example, when the action context output is g2, the M submodule tells the G submodules to expect data related to the orientation of X on a planar surface. The G submodule then applies an algorithm which computes whether Orientation (X)<0, or Orientation(X)>0. When the action context output changes to g1, the G function changes to an algorithm that computes the distance to X.

Each H submodules contains an entire library of procedures in an extended state-table. At each clock tick k, the lefthand side of this state-table is searched for an entry corresponding to the current input S=C+F. If an entry is found, the first column in the right side of the state-table is used as a pointer to a procedure which computes an output P=H(S). If no entry can be found, the pointer is set to an error condition and a procedure is evoked to output the appropriate failure activities. In most cases, a failure condition will output a STOP command to the H submodule below and a failure flag to the H submodule above.

b) Programming the State-Machine Robot

Each entry in the state-table represents an IF/THEN rule, sometimes called a production. This construction makes it possible to define behavior of arbitrary complexity. An ideal task performance can be defined in terms of the sequence of states and state transition conditions that take place during performance. Deviations from the ideal can be handled by simply adding the deviant conditions to the left hand side of the state-table and the appropriate action to be taken to the right hand side. Any conditions not explicitly covered by the table results in an "I don't know what to do now" failure routine being executed. Whenever that occurs, the robot simply stops and asks instructions. If the condition can be corrected, the operator can simply enter a few more rules into the state-table and the robot will continue. By this means, the robot gradually learns how to handle a larger and larger range of problems.

IV. A Microcomputer Network Implementation

In our laboratory at the National Bureau of Standards Anthony Barbera and M.L. Fitzgerald have constructed a state-machine hierarchical control system for a robot in a network of microcomputers [8,9]. This system maps the computational hierarchy of Figure 1 into the physical structure of Figure 3. The coordinate transformations of Figure 1 are implemented in one of the microcomputers of Figure 3. The elemental move trajectories are calculated in a second microcomputer of Figure 3. The processing of the vision data is accomplished in a third microcomputer, and the processing of force and touch data in a fourth microcomputer (arm interface.) A fifth microcomputer provides communication with a minicomputer wherein reside additional modules of the control hierarchy. It is anticipated that these will eventually be embedded in a sixth microcomputer.

Communication from one module to another is accomplished through a common memory "mail drop" system. No two microcomputers communicate directly with each other. This means that common memory contains a location assigned to every element in the input and output vectors of every module in the hierarchy. No location in common memory is written into by more than one computing module, but any number of modules may read from any location.

a) Time Slicing

Time is sliced into 28 millisecond increments. At the beginning of each increment, each computational module reads its set of input values from the appropriate locations in common memory. It then computes its set of output values which it writes back into the common memory before the 28 millisecond interval ends. Any of the logical modules which take longer than the 28 millisecond interval merely wait for the next occurrence of the synchronization pulse and output during the write portion of that interval. The process then repeats.

Each logical module is thus a state-machine whose outputs depend only on its present inputs and its present internal state. None of the logical modules use any interrupts.
Each starts its read cycle on a clock signal, computes and writes its output, and waits for the next clock signal. Thus, each logical module is a state-machine with the IF/THEN, or P=H(S) properties of an arithmetic function.

The common memory "mail drop" communication system has a number of advantages and disadvantages. One disadvantage is that it takes two data transfers to set information from one module to another. However, this is offset by the simplicity of the communication protocol. No modules talk to each other so there is no handshaking required. In each 28 millisecond time slice, all modules read from common memory before any are allowed to write their outputs back in.

b) System Extensibility

The use of common memory data transfer means that the addition of each new state variable requires only a definition of where the newcomer is to be located in common memory. This information is needed only by the module which generates it so that it knows where to write it, and by the modules which read it so that they know where to look. None of the other modules need know, or care, when such a change is implemented. Thus, new microcomputers can easily be added, logical modules can be shifted from one microcomputer to another, new functions can be added, and even new sensor systems can be introduced with little or no effect on the rest of the system. As long as the bus has surplus capacity, the physical structure of the system can be reconfigured with no changes required in the software resident in the logical modules not directly involved in the change.

c) Program Debugging

Furthermore, the common memory always contains a readily accessible map of the current state of the system. This makes it easy for a system monitor to trace the history of any or all of the state variables, to set break points, and to reason backwards to the source of program errors or faulty logic.

The read-compute-write-wait cycle wherein each module is a state-machine makes it possible to stop the process at any point, to single step through a task, and to observe in detail the performance of the control system. This is extremely important for program development and verification in a sophisticated, real-time, sensory-interactive system in which many processes are going on in parallel at many different hierarchical levels.

d) Single Computer Implementation

It should also be noted that a hierarchical control system can be implemented in a single computer [10]. The architecture of the microcomputer network can be simulated in a single program which cycles once per time tick, and can be stopped, or single stepped. The single program would consist of a set of processes, each of which in their turn are allowed to read input variables from a block of common memory, perform some functional operation on those input variables, and hold their outputs in temporary storage until all the processes have completed their read cycle. Then each of the processes is allowed to write its output variables into common memory. The program then cycles back to the beginning and restarts. This is a programming technique which is often used in process-control, systems-simulation, and multitask modeling.

d) Simplicity

The modular, state-machine approach separates the H, M, or G functions into simple understandable blocks of code which can be written, debugged, and optimized independently. The modules have a simple canonical form which makes them understandable and the code readable. It forces a partitioning of the problem into manageable chunks, which can be independently analyzed, reduced to algorithms, and then reassembled into a complex intelligent system. This provides a systematic approach to the synthesis of intelligent behavior.

V. Future Developments

It seems likely that it will soon be possible to design a cross-coupled processing-generating hierarchy, similar to that suggested in Figure 1, consisting of tens or even hundreds of microcomputers. Such large systems are presently being contemplated for the control of entire factories [11]. The hierarchical sensory-control structure makes it possible for many different computing modules, each doing its limited assigned task to be integrated into a coordinated system so that parts, tools, and materials all arrive at the right place at the right time. The correct operations can then be performed, the results inspected, the finished work dispatched to the next work area, and a report made to the next higher level in the hierarchy. Thus, the same type of hierarchical control system suggested here for robots can be extended to integrate robots, machine tools, materials transport systems, inventory control, safety, and inspection systems into a sensory-interactive goal-seeking hierarchical computing structure for a totally automatic factory.

References

1. Albus, J. S., "Mechanisms of Planning and Problem Solving in the Brain," Mathematical Biosciences 45:247-293 (1979).

2. Nilsson, N. J., Problem-solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.

3. Albus, J. S., A. J. Barbera, J. M. Evans, and G. J. VanderBrug, "Control Concepts for Industrial Robots in an Automatic Factory," SME Technical Paper MS77-745, 1977.

4. Barrow, H. G., and Tenenbaum, J. M., "Recovering intrinsic scene characteristics from images." In Computer Vision Systems, A. Henson and E. Riseman (Eds.) Academic Press, New York, 1978.

5. Perkins, W. A., "Model-Based Vision System for Scenes Containing Multiple Parts," 5th Int. Joint Conf. on A. I. (2) 678-684, 1977.

6. Ballard, D. H., C. M. Brown, J. A. Feldman, "An Approach to Knowledge-Directed Image Analysis," 5th Int. Joint Conf. on A.I. (2) 664-670, 1977.

7. Rosenfield, A. R., R. A. Hummel, and S.W. Zucker, "Scene Labelling by Relaxation Operations," IEEE Trans. SMC-6, 1976, pp. 420-433.

8. Albus, J. S., A. J. Barbera, R. N. Nagel, "Theory and Practice of Hierarchical Control," Proc. National Computer Conf., Chicago, IL, May 4-8, 1981.

9. Barbera, A. J., A. S. Albus, and M. L. Fitzgerald, "Hierarchical Control of Robots Using Microcomputers," Proc. 9th Int. Symp. Indus. Robots, March 13-15, 1979, p.405-422.

10. Albus, J. S., A. J. Barbera, M. L. Fitzgerald, R. N. Nagel, G. J. VanderBrug, T. E. Wheatly, "A Measurement and Control Model for Adaption Robots," Proc. 10th Int. Symp. Indus. Robots, March 5-7, 1980.

11. Barbera, A. J., "An Architecture for a Robot Hierarchical Control System," National Bureau of Standards Special Pub. 500-23, 1977.