CAOS: A HIERARCHICAL ROBOT CONTROL SYSTEM[1]

Bir Bhanu, Nils Thune, and Mari Thune

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

ABSTRACT

Control systems which enable robots to behave intelligently is a major issue in todays process of automating factories. This paper presents a hierarchical robot control system, termed CAOS for Control using Action Oriented Schemas, with ideas taken from the neurosciences. We are using action oriented schemas (called neuroschemas) as the basic building blocks in a hierarchical control structure which is being implemented on a BBN Butterfly Parallel Processor.

Serial versions in C and LISP are presented with examples showing how CAOS achieves the goals of recognizing three-dimensional polyhedral objects. We also describe a simulation on how it manipulates objects. Moreover, the ongoing implementation of a parallel version of the system is discussed.

Index terms: Hierarchical Control, Intelligent Robot Control, Parallel Processing, Schemas, 3-D Object Recognition

I. INTRODUCTION

As computers become cheaper and more compact, and the availability of high quality sensors increases, it becomes attractive to create intelligent robots for use in automated environments for recognition, inspection, assembly, and manipulation of objects [7]. Due to safety hazards, repeatability of tasks, or economic constraints, it is an attractive notion to replace humans with robots in many of the tasks mentioned. The availability of hardware for intelligent robots creates a need for designing control programs which have the capability of intelligent goal seeking. The control needs to be concerned with goal achievement guided by diverse information from multiple sensors such as TV cameras, range finders, and tactile-, force-, and torque-sensors. Control becomes crucially important as the tasks, enabled by multisensors, become more complex and involved.

The processing involved in a control system used for robots in automated environments often needs to be done in real time, and it is therefore natural to bring parallel processing into the picture, enabling considerable speedup in execution time when compared to sequential processing on conventional processors. For example, low level image processing, involving large amounts of data, is often accomplished in real time using parallel processors. Furthermore, the control can also experience speedup by running independent parts in parallel.

[1]This work was supported in part by NSF Grants DCR-8506393, DMC-8502115, ECS-8307483 and MCS-8221750

Many existing robot control systems assume a very restricted operational environment [1] limiting the usefulness of the system to a small domain or to tasks which follow a particular pattern in a repetitive fashion. In many cases, for example spray-painting, this is quite adequate. For many other tasks in less structured environments the robots need to be more sophisticated.

Knowledge about the intelligent aspects of a control system can be drawn from the neurosciences where studies of the most intelligent system we know, the human brain and nervous system, indicate some important and basic factors of our intelligence [1]. These factors are also important for a robot control system. They are:

1. The brain is made of basic building blocks, called neurons.
2. The brain is structured in a hierarchical manner.
3. The brain operates in parallel.

The neurons process and produce information which is used to make intelligent decisions about tasks to be done. Even though there are many categories of neurons, such as motor neurons and sensory neurons, almost all of them have the same general structure [8]; multiple *dendrites* carry the input to the *cell body* where the information is processed, and a single *axon* carries the output to other neurons in the nervous system. All of the neurons, with their dendrites and axons, are organized into a complex network which is probably the key to our intelligence, since it provides the necessary links between parts of our brain [1, 2, 8].

It is believed that the neurons, with their complex network of interconnections, are organized in a hierarchical fashion [1]. Commands are issued at the top, and are split into subgoals as they propagate down the hierarchy. In addition to the hierarchical organization, the brain makes extensive use of parallelism in carrying out its tasks [1, 2, 4, 5]. Many neurons operate in parallel, receiving input, processing it, and propagating the results to many other neurons. The brain is quite slow compared to digital computers, being able to carry out only about 100 serial time steps per second [4, 5]. The normal reaction time for a human being is approximately 0.5 to 1.0 second, and the tasks which the brain carries out during this time often require a substantially higher number of computations than 100, leading to the conclusion that parallelism is essential for our ability to react as fast as we do.

In developing an intelligent control system for robots, it is desirable to include the three important aspects of the brain already discussed. With this in mind, we present an approach to robot control called *Control using Action Oriented Schemas* or CAOS. The action oriented schemas are termed *neuroschemas* because of their similarity to *neurons*, which are the basic building blocks of the brain, and *schemas* [3, 6, 9] which are a basic kind of representation. Each neuroschema is able to activate several other neuroschemas in parallel, and they are the basic building blocks of the control system. The neuroschemas are

organized in a hierarchical manner for each goal the system can achieve. Hence, three of the main aspects of the brain have their analogs in CAOS: Basic building blocks, hierarchical organization, and parallel processing.

The purpose of CAOS is to achieve high level goals, specified by a user, through planning and action. The goals which can be achieved depend upon the system's global knowledge base, and are restricted by existing rules, facts, and procedures which the system can consult.

Currently, we have two serial versions implemented in C and LISP respectively. The C version is the preliminary version and its knowledge is sufficient to locate and recognize simple polyhedral objects in range images. Due to implementation difficulties with the C version and the prospect of a LISP compiler for the BBN Butterfly Parallel Processor, the next version was written in LISP. Both the C and the LISP serial versions of CAOS were developed keeping in mind that they should be easily transportable to the BBN Butterfly.

II. SYSTEM ORGANIZATION

CAOS consists of three main parts as shown Figure 1: The scheduler for the hierarchical control, the global knowledge base, and the global data base. The scheduler receives commands (goals) from the user and decides how to obtain a goal by interpreting the rules in the global knowledge base which describe how to subdivide goals. In addition to containing all of the rules and facts on how to obtain goals in a particular domain, the global knowledge base includes experience from previous achievements which are used in planning. The global data base is the world model of the system, containing a known set of facts associated with a particular domain. CAOS is goal driven (backward chaining), eliminating the problem of creating too many unrelated hypotheses as seen in purely data driven systems. Instead, it focuses on the best way of obtaining a goal.

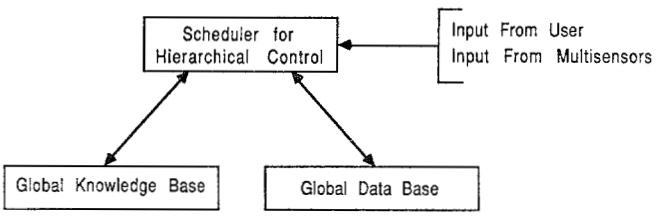

Figure 1: The Basic Building Blocks of CAOS

To achieve a goal, the scheduler uses neuroschemas (implemented as LISP *methods*) to create a tree, organized as a hierarchy, which controls how the goal will be obtained. This tree is an analog to the short term memory in the human brain; the knowledge of how to obtain the main goal exists while it is being achieved and is deleted when obtained, after updating the global knowledge base and the global data base if any new facts or experiences were gained.

One example of a goal to be achieved is (PUT-ON (OBJECT1 OBJECT2)), where OBJECT1 and OBJECT2 might be polyhedral objects (Note: The command syntax in CAOS is LISP syntax).

Three steps are required to obtain the main goal after the scheduler has received the request from the user:

1. Get information from the global knowledge base about the main goal.
2. Get input information from the global data base.
3. Activate a neuroschema for the main goal.

When the scheduler has retrieved the information, it activates a neuroschema which will become the root of the control tree. In this example, the information found about the main goal is that it can only be achieved by obtaining five subgoals: (FIND-IN (OBJECT1 IMAGE)), (FIND-IN (OBJECT2 IMAGE)), (GET (OBJECT1)), (MOVE-TO (OBJECT1 OBJECT2)), and (UNGRASP (OBJECT1)). Furthermore, no information other than the 3-D CAD model of each polyhedral was found in the global data base. Figure 2 shows the control tree of the neuroschemas involved with achieving this main goal.

To obtain the main goal the neuroschema at the root activates new neuroschemas for the subgoals that need to be achieved. In this example both OBJECT1 and OBJECT2 can be found simultaneously (in the parallel version), and hence neuroschemas for these two subgoals can be activated at the same time.

Three cases can arise when the neuroschema tries to achieve the goal it controls:

1. If the neuroschema controls an AND node, it activates new neuroschemas for each subgoal in sequential order. The new neuroschemas can be started in parallel if necessary inputs are available and the respective subgoals are otherwise independent.
2. If the neuroschema controls an OR node, it activates a neuroschema for the subgoal with the highest likelihood of success, or activates several alternatives in parallel if their expectations of success are within a threshold.
3. If the neuroschema controls a leaf in the hierarchically organized control tree, it executes the procedure associated with this leaf.

As seen in Figure 2, the third subgoal, (GET (OBJECT1)), of (PUT-ON (OBJECT1 OBJECT2)) is itself a goal that can be achieved by either obtaining (GRASP (OBJECT1)) or by obtaining (CLEAR-AND-GRASP (OBJECT1)). Which of the two alternatives to be tried first, depends on the previous experience with GRASP and CLEAR-AND-GRASP as explained later. The leaves in the tree are all programs which are executed.

When a goal is successfully obtained, the part of the control tree (short term memory) below that node (goal) is deleted after any new information is stored in the global knowledge base and the global data base.

The Global Knowledge Base: The global knowledge base is a part of the *world model* of the control system, and can be viewed as an analog to the *long term memory* of the human brain. This is in contrast to the *short term memory*, which uses information found in long term memory to obtain a goal, and then disappears. The global knowledge base contains three types of LISP objects classified as AND nodes, OR nodes, and PROGRAM leafs (goals). Each type of node (goal) has slots for storing information about the syntax of the goal, the arguments of the goal, the expected type and range of pre-inputs and post-inputs (see below), the average run time to achieve the goal, the probability of success of obtaining the goal, and an output function. Figure 3 shows the node and leaf structures.

The Global Data Base: The global data base contains the known set of facts associated with a particular domain and is the model of the "world". Currently the global data base is very simple. It only contains information about polyhedral

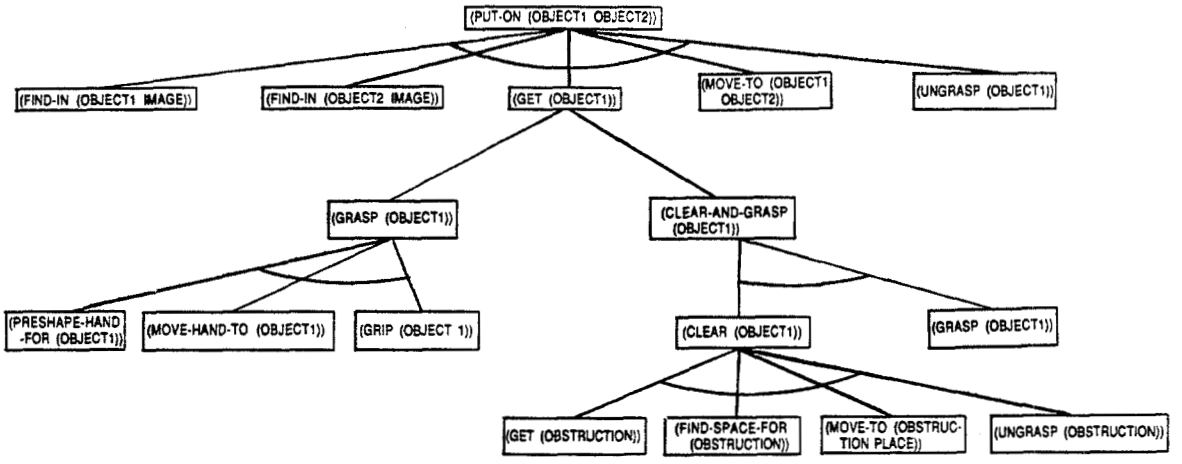

Figure 2: Example - (PUT-ON (OBJECT1 OBJECT2))

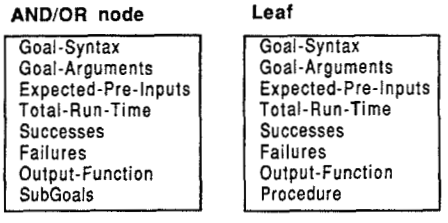

Figure 3: Node and Leaf Structure

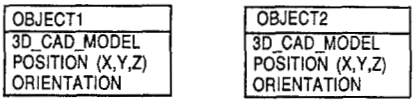

Figure 4: Global Data Base Example

objects. Figure 4 shows an example of the information stored in the global data base.

With each polyhedral object known to the global data base, its three-dimensional model and current position and orientation in the environment are stored. In a later LISP version of CAOS, *frames* will be used to represent information known to the system.

The Neuroschema: The basic unit in the control structure is the *neuroschema.* Each node in the tree in Figure 2 is controlled by a neuroschema. They can only be activated by other neuroschemas, which are already active, and the system is therefore action oriented (goal driven). In addition to activating new neuroschemas, the parent neuroschemas provide decision making (using probability and average run time measures) at their respective nodes in the hierarchy, thus deriving a sequence of steps. This planning is based upon previous experience, and achieves a goal with least risk and cost.

The neuroschema consists of three sections: An *Activation Section*, an *Event Section*, and a *Learning Section.* Each neuroschema is implemented as a LISP *method*, simulating the general functions of a neuron by receiving input, processing it, and producing output. Figure 5 shows the major components of the neuroschema.

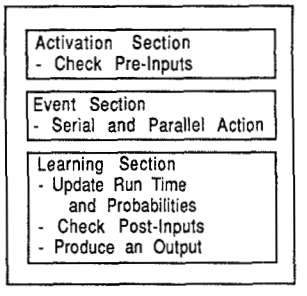

Figure 5: The Neuroschema Components

A neuron can take on any number of inputs and produces an output. The same is true for the neuroschema. In addition, *any type* of input and output can flow through it, making it flexible. The neuroschema makes use of the knowledge with which it has been activated to determine how to process the input and output.

The *Activation Section* checks all the pre-inputs and activates the event section if all of the pre-inputs were satisfactory. Pre-inputs are the inputs given by the user before the main goal is activated or the inputs found in the global data base when the main goal is activated.

The *Event Section* determines how the goal is to be achieved. This corresponds to the functions a neuron performs on its input resulting in its output. The neuroschema, which is activated with a goal, obtains this goal by achieving its subgoals. The subgoals are obtained by either activating new neuroschemas at the level below in the hierarchy, or, if the subgoal is simply a program, it is executed. One particular goal can sometimes be obtained by achieving either one of several different, alternative subgoals (an OR node). Using statistics and average run time from previous successes in obtaining the different goals, the goal with the highest likelihood of success is chosen. Success, s, means that the goal has been obtained with satisfactory output and failure indicates that the goal was not achieved. The goal's probability of success , P(s), is the number of successes, S, for this goal divided by the total number of trials , N (= sum of successes and failures), for this goal.

If a (sub)goal has never been active before, the expectation, E(s), of obtaining it is said to be 0.0 (Note: E(s) is defined over the range 0.0 - ∞ where 0.0 denotes the highest likelihood for obtaining the goal). In all other cases we have:

E(s) = (Average run time of a goal)/P(s)

The average run time of a goal is given as the total sum of all run times for the goal, when the goal was achieved, divided with S.

If there is more than one alternative for obtaining a goal, the control system choses the subgoal with the highest likelihood of success. If the expectation values are equal, the leftmost branch is chosen, or in the parallel version, both would be pursued when the expectation values are within a threshold.

The third section in the neuroschema, the *Learning Section*, is activated when no information about how to obtain the main goal can be found in the global knowledge base, indicated by the scheduler. It is also activated by the activation section each time a neuroschema has obtained all its subgoals, or has executed its procedure, to update the average run time and probability of success. Finally, this section checks the post-inputs and produces the output of the goal based on them. Post-inputs are the outputs of the neuroschemas controlling subgoals of a goal.

The structure of the neuroschema resembles the schema of Arbib, Iberall and Lyon, and Overton [3, 6, 9]. But, in contrast to their schemas, which have *preconstructed plans* (hard wired) for achieving a goal, the neuroschema is a control environment which can be activated with *any* plans for goal achievement from the global knowledge base. Another difference is found in the approach to learning, which, in the case of schemas, is done by instantiating new schemas which better fit a new situation. When our system is learning, new information about how to achieve the goal is used to update the global knowledge base. This new information also takes the form of probability and average run time measures, and is used to achieve the goal next time. Thus, we have a uniform mechanism for building and maintaining the global knowledge base, which is missing in other approaches. Moreover, we have only one type of schemas, called neuroschemas, unlike the work of others [3, 6, 9] who use many different kinds of schemas.

III. EXPLOITING PARALLELISM

The BBN Butterfly: The serial C version of our robot control system is partially transported from a VAX 11/780 to a Butterfly Parallel Processor [10] (parallel version). The Butterfly is a multiple instruction, multiple data (MIMD) machine, and is connected to a host machine which in our case is a VAX 11/780 host. The Butterfly may have up to 256 processor nodes interconnected by a switching network called the *Butterfly Switch*. Each processor node has a co-processor called the Processor Node Controller (PNC) which is responsible for all memory references and transfers. The Butterfly at The University of Utah has 19 Motorola MC68020 processor nodes, each having a Motorola MC68881 co-processor and 1 Mbyte of memory, except two, which have 4 Mbyte of memory. The processors operate at 16 MHz, due to a frequency doubler. References over the Butterfly Switch, to remote memory, usually takes about 4 microseconds round trip.

All code for the Butterfly is developed and compiled on the host machine (VAX 11/780). The executable code is then downloaded to the Butterfly, where it is run. There are two approaches we use to program the Butterfly: *Chrysalis* functions and *Uniform System* functions.

Each processor runs one copy of the operating system Chrysalis. This operating system is mainly written in C and supports communication and synchronization between processes running on different processors. This is done by means of dual queues which pass messages between these processes, and an event mechanism (similar to *signals* in UNIX). Chrysalis does not provide automatic resource allocation, load balancing or process migration, however [4]. Each user-developed program has to set up the data, create all necessary processes, and decide on which node(s) they will run. Five analogs to UNIX's seek-, open-, close-, read-, and write-functions enable access to files residing on the host machine.

Compared to Chrysalis, the Uniform System approach to programming the Butterfly provides the user with easier resource management. The Uniform System is built on top of Chrysalis and consists of several subroutines which take care of, for example, allocation of memory and processors, and generation of new tasks (processes). The user does not allocate memory space or processors explicitly, since the Uniform System takes care of the distribution of tasks on processors and provides special memory allocation routines. The Uniform System is especially suitable for homogeneous problems often found in low level computer vision programs.

Parallelism in the Control System: Exploiting parallelism in the control system involves activating several neuroschemas on different processors, requiring complex communication and synchronization between various processes. This is implemented using Chrysalis. The control system uses the neuroschemas in the hierarchically organized tree, described earlier , to decide if subgoals can be started up in parallel. This occurs when different alternative subgoals can achieve the same goal with approximately the same likelihood of success. In addition, subgoals can be started up in parallel when all needed inputs are provided, and any use of end effectors will not result in conflicts.

One of the advantages of using multiple processors to simultaneously execute alternative goal paths is to prevent time delay due to an alternative's failure to obtain the goal. If one of the alternatives fails, or the results are not satisfactory, the result of another can be used instead. If the alternatives were not executed in parallel, and the most promising one failed, it would take longer to achieve a goal; the next alternative would be executed only after the first had failed.

When the hierarchical control allows parallelism, the parent neuroschema has to check if there are any processors available on which to start up "child processes" (new neuroschemas). If this is the case, the parent must also set up all the necessary data on the respective processors before it can initiate any child processes. The parent and child communicate using a dual queue, on which messages are posted. When a child is done, a special message informs the parent. If no processor is available, however, the child process must be started up on the same processor as the parent. Moreover, if there is only one way of obtaining a goal, the child will always be started on the same processor node as the parent, since there are no alternatives which can be started up in parallel.

The possibility of executing several alternative or independent neuroschemas simultaneously, can speed up the system considerably compared to executing it on a uniprocessor. How much faster it will actually run, depends on how well parallelism can be exploited in each particular case. When the system is entirely transported to the Butterfly, the speedup and utilization of processors, compared to linear execution, will be measured.

Parallelism in Programs: In addition to parallelism in the control system, inherent parallelism can be exploited in programs such as low level image analysis. These programs

deal with image data which requires extensive and time consuming operations. Implementing such programs has no complex control aspects because the processing is homogeneous, enabling the processors to run the same code on different data. This implies that the Uniform System is the best programming approach. One example is edge detection. In this case, the data (the image) can be split into several "chunks" and put onto the available processors, which all run the same edge detection program on their part of the image (a homogeneous problem). There is no complex control aspects involved, like starting up different programs on different processors and taking care of dual queues for message passing between the processes.

Processor Utilization: The two categories of parallelism in the system, discussed above, could use as many parallel processors as there are possible processes. However, there is a limit on the number of processors, 19 in our case, and therefore the problem of processor utilization arises. There has to be a balance between the number of processors the two categories are allowed to occupy. Obviously, the most time consuming processes should use the maximum number of processors, thus reducing the number of "bottle necks" in the system. Since programs such as low level image analysis will be the most expensive part with regard to execution time, it is preferable that these processes occupy most of the processors on the Butterfly, so as to prevent unecessary serial execution. The total execution time for achieving a goal will then be minimized. If the hierarchical control programs occupy just a few nodes, this will not hurt the overall performance significantly, since even the serial versions of the control do not take much execution time.

IV. CAOS AND EXPERT SYSTEMS

In developing CAOS it is important that it has similarities to the human brain and also existing expert systems [11]. These similarities are evident in that CAOS has basic building blocks (neuroschemas), it is controlled in a hierarchical manner, and it is goal driven.

As in an expert system, CAOS has a global knowledge base and a global data base with facts and rules (rules for obtaining goals). Furthermore, like the inference engine (interpreter and scheduler), our neuroschemas contain the general problem solving knowledge. Moreover, in order to do anything "intelligent", both expert systems and our control system need a domain expert to provide them with knowledge on how to obtain goals. The neuroschemas in CAOS provide a consistent way of interpreting how goals should be obtained. They use metarules in the form of probabilities and average run time when deciding which subgoals to pursue to achieve a main goal. Making CAOS goal driven (backward chaining) prevents problems with generating too many hypotheses, but does not avoid the problem of restricting the hypotheses to too narrow a range. In both systems the knowledge is permanent and consistent, and can be goal directed.

One main difference between CAOS and expert systems in general, is that our system can easily receive sensory information from the environment provided programs exists in its global knowledge base enabling interaction with sensors. Implementing the system in LISP or C and using the neuroschemas as the basic building blocks, enable us to easily acquire this sensory information, and also control end effectors. In addition, the system can more easily be exported to the BBN Butterfly Parallel Processor than a system written using expert system tools, since this machine currently supports only C and will support LISP sometime soon.

The conventional expert system relies on symbolic information provided by the user to achieve a goal, and usually cannot interact with the environment through sensors. This seems to be the most important difference. But this does not mean that an expert system with the right expert-system-building tool could not provide this capability.

We conclude that although CAOS is similar to expert systems, essential differences are found in the use of basic building blocks, neuroschemas, and easy multisensor integration due to implementation in LISP and C.

V. EXPERIMENTAL RESULTS

To demonstrate how the control system works, we describe two categories of examples. One, taken from the serial C version, involves recognition of polyhedral objects using range images, and the other, taken from the serial LISP version, involves a simulation of object manipulation. At each level in the hierarchically organized control trees shown in the figures, the goals are executed from left to right.

Experiment 1: In the first experiment we used a CAD based three-dimensional object recognition algorithm for simple polyhedral objects. The steps involved in this algorithm are as follows:

1. Selecting pertinent features
2. Building a data base
3. Generating hypotheses
4. Verifying hypotheses

In the particular goal (FIND-IN (OBJECT IMAGE)) the above mentioned steps are divided into subgoals as shown in the control tree for this goal in Figure 6. In this particular case, the goal can be obtained by either identifying a polyhedral or a cylinder. The first two subgoals (from left to right) of FIND-POLYH-IN are the pertinent feature selection. The next three subgoals are the hypotheses generation, the hypotheses verification, and lastly its position determination. The determined information is stored in the global data base and is thereby available for direct access from other parts of a more extensive control tree (as in Example 2). The subgoals for FIND-CYLINDERS would look similar, but are currently not a part of the global knowledge base.

Each of the (sub)goals were initially assigned an expectation value of 0.0 for success, as described in the subsection "The Neuroschema". This means that the number of trials (N) were set to 2, the number of successes (S) to 1, and the run time to 0.0 (the expectation value is then E(s) = 0.0/(1/2) = 0.0). Since FIND-POLYH-IN has only one alternative way of achieving the goal, updating of expectation values would never cause another alternative to be executed. Nevertheless, this example helps illustrating how the expectation values are updated. As each subgoal was executed, the success or failure was indicated by updating S, N, and the run time for the subgoal. For example, if VERIFY-HYPOTHESIS failed and all the other subgoals succeeded the first time, the first three subgoals were assigned S values of 2 and N values of 3, whereas the fourth obtained an N value of 3 and a S value of 1. The goal FIND-POLYH-IN also received N = 3 and S = 1, since the overall goal failed when VERIFY-HYPOTHESIS failed. When a detection paradigm for cylinders is included in the control tree, the two OR branches at the top can be executed in parallel, since they are independent.

Experiment 2: The second experiment involved simulation of object manipulation by a robot. Figure 2 shows a partial control tree for this goal. To obtain the goal (PUT-ON (OBJECT1 OBJECT2)), there are several possibilities for goal achievement which are evident in that there are

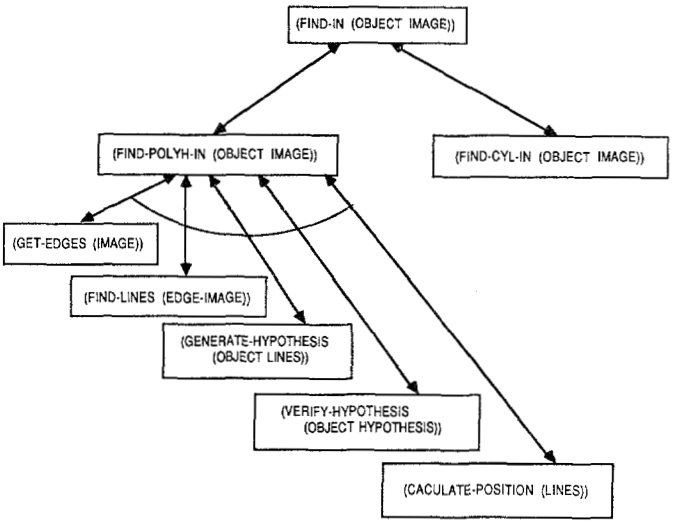

Figure 6: Example - (FIND-IN (OBJECT IMAGE))

several OR nodes in the tree. This experiment can thereby show how the decision making is done depending on success or failure at different nodes.

Just as in the above example, all (sub)goals were assigned an initial expectation of 0.0 by setting N = 2 and S = 1 for each of them and setting run time = 0. Since this experiment was a simulation, some of the programs (at the lowest level in the hierarchy, or as leaves on the tree) just returned "success" or "failure" more or less randomly.

In the first execution a failure occurred in the third subgoal for (PUT-ON (OBJECT1 OBJECT2)). Its subgoal (MOVE-HAND-TO (OBJECT1)) failed, let us say because an obstruction was found on OBJECT1 (actually, the program just returned "failure" as described earlier). The alternative subgoal (CLEAR-AND-GRASP (OBJECT1)) was then tried with success by clearing OBJECT1 before getting it. The two last subgoals of PUT-ON succeeded, resulting in the final values for trial and success values (N and S respectively).

If the main goal were to be executed again, and *all* subgoals were to succeed the first time they were tried, the current expectation values would provide the necessary information for a neuroschema's decision about which alternative (subgoal) to try first when encountering an OR node. The third subgoal, (GET (OBJECT1)), is an OR node with two alternative subgoals (note that the positions of OBJECT1 and OBJECT2 are stored in the global data base after executing FIND-IN at the second level of the control tree). The leftmost alternative has a lower likelihood of success than that of the rightmost one, because it has failed once whereas the other has not failed at all. The execution time for the rightmost is higher than for the leftmost, however, and when this is taken into account we get the expectation values (= average run time/probability of success) for the two alternatives. The one with expectation value closest to 0.0 will be chosen and tried first.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have discussed our work with the robot control system CAOS. This system is developed with three of the important aspects of human intelligence in mind: Basic building blocks, hierarchical organization, and parallel processing. The system consists of three basic parts which include the global knowledge base, the global data base, and the scheduler for hierarchical control. This scheduler and the neuroschemas use information found in the global knowledge base and the global data base to determine how to obtain a goal given by the user.

Two examples showing how the system functions are presented, including recognition of polyhedral objects and a simulation of object manipulation by a robot. We also conclude that although CAOS is similar to expert systems in some respects, they are set apart by differences such as using neuroschemas as a basic unit, and the easy intergration of sensory input and output.

Future work on the control system will include a complete implementation on the BBN Butterfly Parallel Processor in C, or LISP if a reliable LISP compiler becomes available in the near future. We will obtain results for speedup when comparing parallel versus serial processing, and will explore processor utilization and process synchronization.

REFERENCES

[1] J.S. Albus. *Brains, Behavior, & Robotics*. BYTE Publications Incorporated, 1981.

[2] J.A. Anderson. Cognitive and Psychological Computation with Neural Models. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13(5):799-815, September/October, 1983.

[3] M.A. Arbib, T. Iberall and D. Lyons. *Coordinated Control Programs for Movements of the Hand.* COINS 83-25, University of Massachusetts, August, 1983.

[4] C.M. Brown, C.S. Ellis, J.A.Feldman, T.J. LeBlanc and G.L. Peterson. *Research with the Butterfly Multicomputer*. Technical Report, BBN Laboratories Incorporated, 1986.

[5] J.A. Feldman. Connectionist Models and Parallelism in High Level Vision. *Computer Vision, Graphics, and Image Processing* 31(2):178-200, August, 1985.

[6] T. Iberall and D. Lyons. *Towards Perceptual Robotics*. COINS 84-17, University of Massachusetts, August, 1984.

[7] C. Jacobus, W.D. Lee and J. Norton. Flexible Assembly and Inspection of a small Electric Fuel Pump. *Proc. SPIE, Intelligent Robots and Computer Vision* 579:528-536, 1985.

[8] E.R. Lewis. The Elements of Single Neurons: A Review. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13(5):702-710, September/October, 1983.

[9] K.J. Overton. *The Acquisition, Processing, and use of Tactile Sensor Data in Robot Control.* COINS 84-08, University of Massachusetts, May, 1984.

[10] B. Thomas, R. Gurwitz, J. Goodhue, D. Allen and M. Beeler. *Butterfly Parallel Processor Overview.* BBN 6148, BBN Laboratories Incorporated, March, 1986.

[11] D.A. Waterman. *A Guide to Expert Systems.* Addison-Wesley Publishing Company, Inc., 1986.