KNOWLEDGE BASED ROBOT CONTROL ON A MULTIPROCESSOR IN A MULTISENSOR ENVIRONMENT[1]

Bir Bhanu and Nils Thune
Department of Computer Science, University of Utah
Salt Lake City, Utah 84112

Abstract - Knowledge based robot control for automatic inspection, manipulation, and assembly of objects will be a common denominator in most of tomorrow's highly automated factories. These tasks will be handled routinely by intelligent, computer-controlled robots with multiprocessing and multisensor features which contribute to flexibility and adaptability. In this paper we discuss the ongoing work with CAOS which is a knowledge based robot control system. The structure and components of CAOS are modeled after the human brain using neuroschemata as the basic building blocks which incorporate parallel processing, hierarchical, and heterarchical control.

Index terms: Hierarchical Control, Intelligent Robot Control, NeuroSchemata, Parallel Processing

I. Introduction

There is an increasing demand from the industry to automate tasks such as recognition, assembly, inspection, and manipulation of objects. The availability of hardware for creating *intelligent robots* that can handle such tasks creates the need for robot control programs that are capable of controlling vast amounts of complex data provided by the multiple sensors of the robot. Such control programs must be capable of achieving high level goals and must also be flexible, adaptable, and be able to learn from past experience.

The control program 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. The processing involved in a control system used for robots in automated environments 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 a single processor.

Most of todays robots operate in a very restricted environment due to the limited functionality of their control systems [1, 4]. This constrains the usefulness of the *expensive* robot to simple and repetitive tasks.

To enable robots to interact intelligently with their environment, we need an artificial brain that can control the robot. Knowledge about the intelligent aspects for such a *robot brain* can be drawn from the neurosciences where studies of the most intelligent system we know, the brain and nervous system, indicate some important and basic factors of our intelligence [1]. These factors are also important for a robot control system.

---

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

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

The human brain and nervous system controls the activities of the human body. It coordinates the various activities, receiving thousands of bits of information from multiple sensor organs and interneurons [7, 9, 10]. Computational responses to the environment, which the body exists in, originates from sensory experience enabled by various tactile, auditory, and visual receptors. Neurons, the fundamental units of the brain and nervous system, encode and decode complex information through a network of interconnections between millions of other neurons. The interconnections and processing between the neurons forms the intelligent system that controls the human body.

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, 5, 6]. Many neurons operate in parallel, receiving input, processing it, and propagating the results to many other neurons.

In developing an intelligent control system for a robot it is natural to use ideas found in studying the brain. With this in mind, we are developing a robot control system called *Control using Action Oriented Schemata* or *CAOS*. The action oriented schemata are termed *neuroschemata* because of their similarity to *neurons*, which are the basic building blocks of the brain, and *schemata* [3, 8, 12] which are a basic kind of representation. Each neuroschema is able to activate several other neuroschemata in parallel, and they are the basic building blocks of the control system. The neuroschemata 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 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. Both the C and the LISP serial versions of CAOS are developed keeping in mind that they should be easily transportable to a parallel processor.

II. CAOS: A Hierarchical Control System

An intelligent control system for a robot needs to be able to achieve multisensory coordination. To make a robot intelligent it must be capable of the following:

- It must be able to perceive. That is, it should be able to become aware of things through receptors (multisensors) and previous knowledge.
- It must be able to learn and know; It should have the capability to learn by receiving information from the environment and use this as knowledge later.
- Finally, it must be able to understand and reason. That is, it should understand the meaning and/or nature of a problem and be able to use logic to solve it.

The ability to process sensory data in a consistent way is important, and to fully utilize the resulting information it must be integrated into the control of the robot. A control system for a robot can be viewed as having two parts. The first part is related to the structure which controls the sensory and

motor behavior of the robot and the second part is concerned with the processing and representation of the sensory and control data.

CAOS is a framework for developing goals and representing knowledge, and is able to coordinate such goals based on multisensory information and basic environmental knowledge. The overall system structure of CAOS is shown in Figure 1.

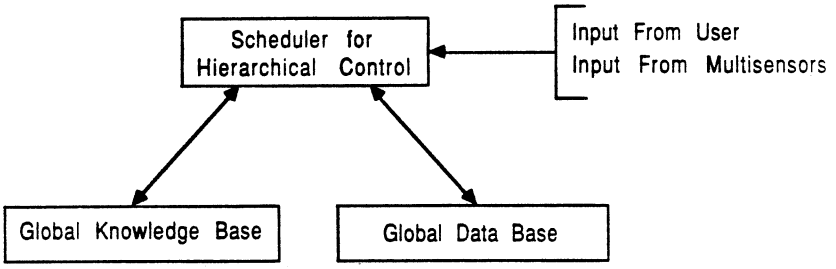

Figure 1: The CAOS System Structure

With CAOS, one is able to represent a dynamic world in which the robot interacts with the environment through planning and action based upon goals the robot is capable of performing. The approach used in CAOS provides control at all levels extending from specifying low level motion, position, velocity, and force commands on joints and end effectors to high level planning.

Action Oriented Control: As control programs grow exponentially in complexity with the number of sensors and branch points in the plan for achieving a goal, the choice of using hierarchical control with action oriented schemata (neuroschemata) enables the system to be partitioned into levels of limited complexity. Each level in the hierarchically organized tree have goals which decrease in complexity with decreasing level in the hierarchy. Each level in the tree produces "a jump in the intelligence" of the system.

Direct evidence of the hierarchical subdivision of control is found in the brain and is called the *Neuronal Hierarchy* [1]. At the bottom, neuronal computations can be concerned with only a single muscle group. Moving up one level, the computation now involves several muscle groups. Several levels above this again the motion of an entire limb, and action between limbs, are computed. Finally at the highest level of the neuronal hierarchy, the entire human body is coordinated towards future goals.

CAOS is structured in a hierarchical way and is action oriented (the system's goals know which subgoals to activate to achieve a specific goal state). It is based upon basic building blocks - neuroschemata - which incorporate mechanisms for planning, stable control, sensory data processing, and representation of the world. The CAOS shell controls the user interface and goal achievement. It enables the user to describe goals which the system should be able to achieve.

The Global Knowledge Base: The 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* [4] 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 knowledge base contains three *goal* types. They are LISP objects classified as *and-nodes*, *or-nodes*, and *program-leafs*. These three types represent goals that CAOS can achieve. They are directly comparable to the goal states neurons in the brain can be in. The main difference is that each goal/subgoal (neuron) of the brain is constantly active, monitoring its environmental input, while the goals/subgoals of the CAOS control system are passive until a particular goal state is desired which involves activating neuroschemata with these goals/subgoals.

Each node and leaf 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, the average run time to achieve the goal, the probability of success of obtaining the goal, and an output function. Figure 2 shows the node and leaf structures.

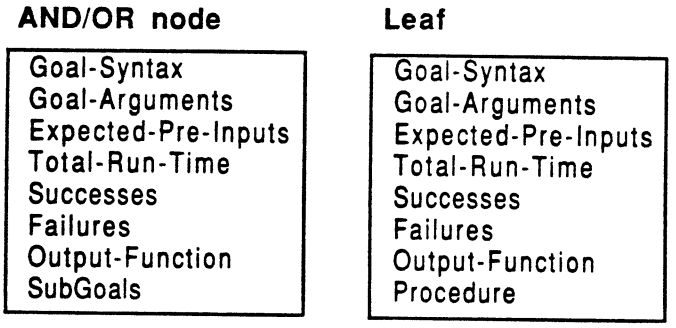

Figure 2: Structure of Node and Leaf Objects/Nodes

Information found in these node and leaf goals are specified by the user. This is similar to specifying rules in an expert system. The *goal-syntax* specifies the goal state and the pre-input (if any) needed to obtain this goal state. The goal arguments are simply the pre-input that the goal was activated with (needed input for achieving the goal state). *Expected pre-inputs* are functions (if any) which are used for checking the range and type of the pre-input that the goal was activated with. The *total run time* is the run time that the goal uses to achieve the goal state. *Successes* and *failures* are just what they say, the total number of previous successes and failures in trying to obtain the goal state. The *output function* is used to produce the output of the goal which implies that the desired goal state has been achieved. Finally, the *subgoals* are the goals needed to achieve the desired goal state. This is usually referred to as *goal/problem reduction.* In the case of a leaf the subgoals would have been replaced with a single function.

The Global Data Base: The data base contains the known set of facts associated with a particular domain and is the model of the *world.* The database used in the C implementation of CAOS was very simple, it contained information about polyhedral objects only. Figure 3 shows an example of the information stored in the data base.

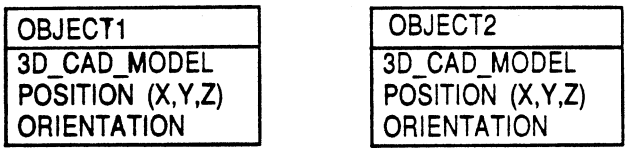

Figure 3: Data Base Objects

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

The Neuroschema: Due to the lack of parallel processing being available to the extent needed to fully simulate all the neurons in the brain, a more rigid approach to control is needed. Some processing can be done in parallel on existing computers, but not enough compared to the amount needed for simulating the brain, and hence a different basic building block than the neuron is expected for the control system if it is to be functionally implemented on todays existing parallel processors.

The neuroschema provides this basic building block. It is a strict processing element for controlling input and output of a specific function. The neuroschema can be viewed as corresponding to the cell body of the neuron, with the difference that it becomes active only in certain situations, compared to the neuron which is active constantly.

The neuroschema consists of three sections: an *Activation Section*, an *Event Section*, and a *Learning Section*. Each neuroschema is a LISP *method*, simulating the general functions of a neuron by receiving input, processing it, and producing output based upon the function (goal) it controls. Figure 4 shows the major components of the neuroschema.

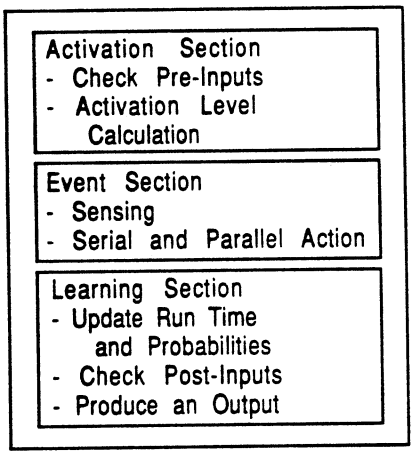

Figure 4: The Neuroschema Components

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 is activated with a goal (a LISP object) and uses the information in this goal to determine how to process the input and output. An active neuroschema with its goal in the robot control system is the counterpart to the neuron of the human nervous system.

Information flowing in the control system is provided by the user, multi sensors [11], and computations done internally by the neuroschemata. In contrast to a neuron, which constantly monitors its dendrites (inputs), the neuroschema becomes active only when a particular goal state is requested. To achieve this particular goal state a neuroschema is activated with a goal object which contains information about how to obtain this goal. The *activation section* of a neuroschema becomes active at the moment the neuroschema is activated. Along with the goal object the neuroschema is also activated with some particular input which we call *pre-input* (as opposed to post-input discussed later). Using the information found in the goal object the neuroschema checks the pre-input for acceptable range and type. Depending on this result the event section of the neuroschema is activated.

The *Event Section* of the neuroschema uses the information in the goal object to determine how the goal is to be achieved. The neuroschema, activated with a goal, obtains this goal by achieving its subgoals. The subgoals are obtained by either activating a new neuroschema for each of them, or if the subgoal simply is a program/function, it is executed. Subgoals of a goal can be directly compared to interneurons of the brain while programs/functions are analog to motor neurons in the nervous system controlling receptors and joints.

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 highest expectation of success is chosen. Success means that the (sub)goal has been obtained with satisfactory output and failure indicates that the (sub)goal

was not achieved. This is comparable to the function of the cell body of a neuron which produces an output when necessary input satisfies it (the goal state is met).

If a (sub)goal has never been active before, the expectation 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 expectation for obtaining the goal). In all other cases we have:

E(s) = (Average run time of 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 were achieved, divided with S (the number of successes for obtaining this goal). P(s) is given by S divided with N (the total number of trials for this goal). If there is more than one alternative for obtaining a goal, the control system choses the subgoal with the highest expectation of success. If the expectation values are equal, the leftmost branch is chosen, or in a parallel version, both would be pursued when the expectation values are within a specified 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 knowledge base, indicated by the scheduler. It is also activated 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 these. Post-inputs are the output of the neuroschemata controlling the subgoals of a goal. Hence, post-inputs and pre-inputs together are comparable to all the inputs received through the dendrites of a neuron.

The structure of the neuroschema resembles the schema of Arbib, Iberall and Lyon, and Overton [3, 8, 12], but is very different in functionality. In contrast to their schemata, which have preconstructed plans for achieving a goal, the neuroschema is a control environment which can be activated with any plans for goal achievement from the knowledge base. Another difference is found in the approach to learning, which, in the case of schemata, is done by instantiating new schemata which better fit a new situation. When the CAOS system is learning, new information about how to achieve the goal is used to update the knowledge base. This new information includes the probability and average run time measures, and is used to achieve the goal the next time. Furthermore, neuroschemata is implementationally possible on current equipment, using parallel processing as available, while schemata are directly restricted in operation depending on the number of processors currently available on the machine they are running on.

Parallelism in the Control System: Exploiting parallelism in the control system involves activating several neuroschemata on different processors. This requires complex communication and synchronization between various processes. The control system uses the neuroschemata in the hierarchically organized tree 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 expectation 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 neuroschemata). 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. When a child is done, a special message informs the parent. However, if no processor is available 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 neuroschemata 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.

Parallelism in Programs: In addition to parallelism in the control system, inherent parallelism can be exploited in programs such as low level image processing. 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. 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 are 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 always a limit on the number of processors available at any time 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 *bottlenecks* 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, 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 for simple problems even the serial version of the control does not take much execution time.

III. Conclusion and Future Work

In this paper we have discussed our ongoing work with CAOS, an approach to robot control. It is being 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 knowledge base, the data base, and the scheduler for hierarchical control. The scheduler and the neuroschemata use information found in the knowledge base and data base to determine how to obtain a goal given by the user. The neuroschemata provide the functionality of neurons, incorporating parallelism as it is available, and is easily implementable with no requirements to the system it is implemented on. We have implemented C and LISP serial versions of CAOS.

Future work on CAOS will include a control structure with neuroschemata that uses both *hierarchical* and *heterarchical* control. Furthermore, a parallel version using extensive inherent parallelism in the control system and the programs will be implemented.

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*. Department of Computer and Information Sciences, COINS 83-25, University of Massachusetts, August, 1983.

[4] D.H. Ballard and C.M. Brown. *Computer Vision*. Prentic-Hall, Inc, 1982.

[5] 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.

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

[7] Arthur C. Guyton. *Human Physiology and Mechanisms of Disease*. W.B. Saunders Company, 1982.

[8] T. Iberall and D. Lyons. *Towards Perceptual Robotics*. Department of Computer and Information Sciences, COINS 84-17, University of Massachusetts, August, 1984.

[9] Eric R. Kandel and James H. Schwartz. *Principles of neural science*. Elsevier Science Publishing Co, Inc., 1985.

[10] Stephen W. Kuffler, John G. Nicholls, A. Robert Martin. *From Neuron To Brain*. Sinauer Associates Inc. Publishers, 1984.

[11] Amar Mitiche and J. K. Aggarwal. Multiple Sensor Integration/Fusion through Image Processing. *Optical Engineering* 25(3):380-386, March, 1986.

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