Use of Expert Systems in Closed Loop Feedback Control

Årzén, Karl-Erik

1986

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7320/1-6/(1986)

Use of Expert Systems in Closed Loop Feedback Control

Karl-Erik Årzén

Department of Automatic Control
Lund Institute of Technology
May 1986

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
INTERNAL REPORT

Date of issue
May 1986

Document Number
CODEN: LUTFD2/(TFRT-7320)/1-6/(1986)

Author(s)
Karl-Erik Årzén

Supervisor

Sponsoring organisation
STU

Title and subtitle
Use of expert systems in closed loop feedback control

Abstract

Different uses of expert systems in process control are discussed. The paper concentrates on expert systems for closed loop control. Motivation for this is given and a prototype experiment is described. The experiment is evaluated and an expert control architecture is proposed. The system can be compared with a real-time operating system. An implementation based on YAPS and object-oriented programming is described.

This paper is presented at the American Control Conference, Seattle, USA, June 1986.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
6

Recipient's notes

Security classification

USE OF EXPERT SYSTEMS IN CLOSED LOOP FEEDBACK CONTROL

Karl-Erik Årzén

Department of Automatic Control,
Lund Institute of Technology,
Box 118, S-221 00 Lund, Sweden.

Abstract: Different uses of expert systems in process control are discussed. The paper concentrates on expert systems for closed single loop control. Motivation for this is given and a prototype experiment is described. The experiment is evaluated and an expert control architecture is proposed. The system can be compared with a real-time operating system. An implementation based on YAPS and object-oriented programming is described.

1. INTRODUCTION

Expert systems have many potential applications in process control. The application domain stretches from the entire plant to the single control loop. Both off-line and real-time problems exist. In this paper the expert system is used in real-time as a part of a single control loop. Many applications incorporate expert operator knowledge into the system. Our application is instead focused on incorporating more control knowledge into the controller. Good control requires high-quality process knowledge. This can be achieved in two ways, either directly from the operator or through experiments with the controlled process. The goal of the expert system is to build up the necessary process knowledge required for good control.

The paper is organized as follows. General expert system applications in process control are discussed in Section 2. Section 3 focuses on the single control loop. A prototype experiment is described. In Section 4 an expert control architecture is proposed and its implementation is discussed in Section 5.

2. EXPERT SYSTEMS IN PROCESS CONTROL

Expert systems is an area of Artificial Intelligence (AI) that has expanded rapidly during the last years, (Hayes-Roth *et al.*, 1983; Waterman 1986). Expert systems are used to solve problems that normally require human expertise and where traditional computer solutions are infeasible. The successful applications have all been in areas where high-quality knowledge dominates over common sense. This is true for many aspects of process control. The idea of adding heuristics to process control is not new. In Crossman and Cooke (1962), a heuristic decision program that used experience to enhance its performance was proposed for manual control of systems with slow dynamics.

Examples of possible off-line applications in process control are process design and control design. Process design has no clean analytical solution. Expert system assistance has a high potential value since process design strongly affects the achievable control quality. In control design expert systems can guide the selection of appropriate control structures, (Umeda and Niida, 1986), at the global level. Expert systems have also been integrated with computer aided control design packages, (James *et al.*, 1985, Birdwell *et al.*, 1985, Larsson and Åström, 1985). It has also been suggested to use expert systems to assist in the parameter settings of adaptive controllers, (Sanoff and Wellstead, 1985).

With few exceptions, existing expert systems are based on propositional logic or first-order predicate calculus. Sometimes they also allow multi-valued logic, e.g. expressions with the values true, false or unknown. This is however not enough for a real time expert system. Such a system may have to draw conclusions based on incomplete facts. Facts may also change after conclusions have been drawn from them. This means that the system has to backtrack and reconsider these conclusions. Several non-standard logics exist e.g. non-monotonic logic, (McDermott and Doyle, 1980) and temporal logic, (McDermott, 1982, Allen, 1984), that partially solve these problems but they have not yet been applied successfully.

In spite of these problems real-time expert systems exist. These systems tend to circumvent the fundamental problem in different ways. The method used in PICON, (Moore *et al.*, 1984, 1985), is to attach a duration time to all database elements and to test the rules periodically. When the duration of an element ends all concluded elements are withdrawn. It is also possible to assume that database elements are valid only in a certain context. The usual way, however, is to use the engineering, ad hoc method and take

care of these issues explicitly in the rules.

Monitoring and diagnosis are the most common real time applications. Plant-specific knowledge e.g. cause-effect relations, is used to locate a fault which caused a process upset or an alarm and to give advice on corrective action. Work is being done e.g. on nuclear power plants, (Nelson, 1982), and chemical plants, (Palowitch and Kramer, 1985). A monitoring expert system could also give advice on control optimization.

In these applications the expert system is used as a tool for the operator. The expert system loop can also operate in closed loop, i.e. affect the controlled plant directly. On a global level the expert system could e.g. be used for set-point control to optimize the system. Process start-up and production changes. is another possible area. In the aircraft industry expert systems are suggested that reconfigure the flight control system in case of damage, see e.g. Trankle and Markosian (1985). The topic of the rest of this paper is the use of expert systems in closed single loops. The ideas were first outlined in Åström and Anton (1984).

3. CLOSED LOOP EXPERT CONTROL

The present project aims at incorporating a rule based expert system in a feedback control loop. The goal of the expert system is to enhance the performance of the single loop controller and to learn as much as possible about the controlled process. This is achieved by orchestrating the application of different numerical algorithms to the process in an "intelligent" way. The numerical algorithms can be of different types: control algorithms, identification algorithms and monitoring algorithms. The control algorithms may be of varying complexity, from simple PI or relay controllers to more complicated optimal or pole-placement algorithms. The identification algorithms may range from simple algorithms for estimation of static gain to more complex algorithms such as the Least-Squares algorithm, (Åström and Wittenmark 1973). Supervisory algorithms should detect e.g. static errors, alarm level crossings and ringing in the controller output. The expert system should decide in which order the algorithms are applied and calculate their parameter settings. The application of one algorithm increases the knowledge about the physical plant and affects the application of further algorithms.

Existing single loop controllers consists of a combination of a control algorithm and a "safety jacket" of logical conditions. Safety jackets are often logical networks which dominate program code. They can be difficult to modify and often make the code less readable. The goal is to implement as much as possible of this as rules in the expert system. This gives a clean separation between the numerical algorithms and the branching logic that simplifies development and maintenance. It is often desirable to combine different algorithms. Some examples are different identification algorithms in self-tuning controllers, one controller for steady state operation and one robust controller for startup etc. The combination of algorithms imposes additional requirements on the correctness of the logics and inforces the need for a structured implementation.

High quality control requires good process knowledge. Adaptive controllers extract some of this knowledge from the process but they still need much *a priori* information such as system order, number of time delays etc., to perform well, (Isermann, 1982, Åström, 1983, Clark, 1984). One idea behind the expert system approach is to include as much process knowledge as possible in the controller. This knowledge can be collected in two ways: by asking the process operator or by performing experiments on the process. An experienced process operator has knowledge about the physical process. This knowledge is often qualitative. Examples may be estimates of dominant time constants and static gain, the nature of non-linearities etc. Simple identification experiments exist, (Åström and Hägglund, 1984) that can be used to extract knowledge from the process. The information obtained in these ways are diverse and sometimes uncertain or contradictory. The expert system approach gives a possibility to exploit and refine this knowledge.

In most of the examples given in the previous section, the expert system was used to incorporate the knowledge of the process operator. In this paper the emphasis is more on the expert knowledge of the control engineer. A controller of this kind has two possible uses. As an actual industrial controller or, perhaps more interestingly, as a testbench for rapid prototyping of new control structures.

From an AI point of view the on-line control application includes both planning and monitoring. The system should plan how the numerical algorithms should be applied to the process and monitor both the execution of the plan and the actual control. The expert system and the algorithms must be implemented as parallel processes having different priorities. The reason for this is that the different processes operate in different timescales. The response time of the algorithms must match the physical process while the rule interpretation in an expert system is a comparatively slow process.

An environment for experiments with expert control has been developed on a VAX 11/780 running VMS. This is described in more detail in Årzén (1986a). It consists of three parts: the expert system, the numerical algorithms and the man-machine interface. These parts are implemented as subprocesses that communicate by sending messages through mailboxes. The process structure is described in Fig 1. The numerical algorithms are implemented in Pascal and could be viewed as a library of algorithms. This process is connected to A/D and D/A converters. The algorithms works as filters that extract symbolic, qualitative information from the numerical signal flow. The expert system is not involved unless something significant has been detected by the algorithms. The expert system and the man-machine interface are implemented in Lisp. A simulation program, Simnon (Elmqvist, 1975), has been interfaced to the system as an alternative to controlling a physical process. Simnon can simulate

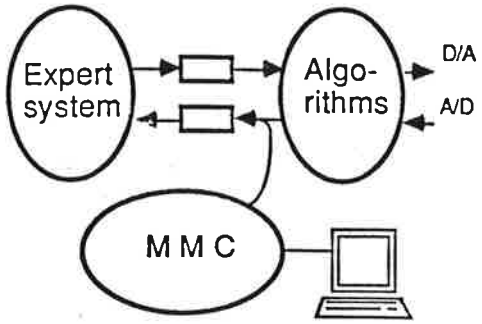

Fig. 1: Process structure. The ellipses represent processes and the rectangles represent mailboxes.

nonlinear differential or difference equations. The program has been modified so that it operates in real time.

This environment has been used in a prototype system where the expert system was implemented with the conventional expert system shell OPS4 (Forgy, 1979). The prototype system is described in more detail in Årzén (1986b). OPS4 is a pure forward chaining system. The database consists of arbitrarily nested list expressions. The condition parts of the rules are patterns that are matched against the contents of the database. There is no possibility of grouping rules according to context. This system was interfaced to the real-time environment by a rule that read new messages from the mailbox and entered them into the database. This rule was executed when no other rules were matched. In this way the rule execution was restarted.

The prototype system was used for experiments with a "smart" PID controller with auto-tuning and gain scheduling. The tuning was based on the Ziegler-Nichols auto-tuner (Åström, 1983). Relay oscillations are used to determine the PID parameters. When the loop is closed with the relay the controlled signals starts to oscillate. This oscillation corresponds to the point where the Nyquist curve of the process crosses the negative real axis. This point gives the ultimate period and the ultimate gain which are then used to compute the PID parameters. The algorithms needed to implement the system were a PID algorithm, a relay algorithm, an oscillation analyzer and a noise estimator. The controller worked in three different modes: manual, tuning and PID. When in PID mode a table was used to schedule the PID parameters for different operating conditions. The operator could change modes and enter new control parameters. It was possible to change the contents of the database and edit the rules on-line. The rules in the system could be divided in the following groups: noise-estimation rules, relay oscillation rules, parameter computation rules, PID supervision rules and command decoding rules. The system contained about 70 rules.

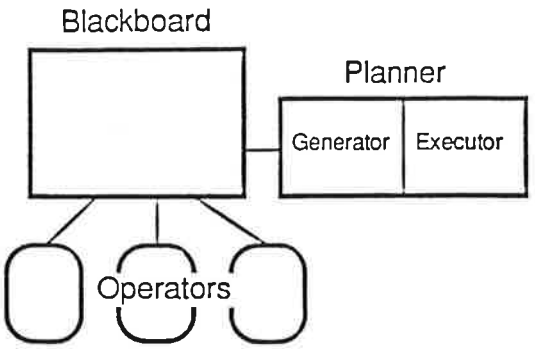

Fig. 2: An expert control architecture.

4. AN EXPERT CONTROL ARCHITECTURE

The experiments performed with the prototype system have been promising. The expert system approach gave a clean implementation that clearly showed the benefits of separating the of logic from the algorithms. As an example, the addition of gain-scheduling required only the addition of about 10 new rules to the system. This approach is thus very promising as a testbench for rapid prototyping.

The experiments also showed that a conventional expert system shell is poorly suited for real-time operation. Expert control contains a large element of planning that is not well supported by conventional expert system shells. An example is the tuning phase of the controller that contains a large sequential element. First a noise estimator algorithm is used to collect noise information which is then used to set the relay parameters. A detector is applied to determine that a steady state oscillation is obtained. The PID parameters are determined from the oscillation wave form and PID control is initiated. It is natural to group the rules according to the different stages in this plan. Production systems are generally weak at sequencing problems. The sequencing has to be explicitly expressed in the rules. This often gives the effect that the actual domain knowledge is obscured by the control knowledge i.e. when to apply the rules.

Another disadvantage with the prototype system was the uniform inference strategy. The problem is basically of the data-driven, forward chaining type with data in the form of significant events being sent from the algorithms to the expert system. The monitoring phase can, however, be stated as a diagnosis problem where backward chaining is more appropriate. The same is valid in the phase where the system tries to extract process knowledge from the operator. The lack of structure of the database was a third drawback. A database that allows objects with attributes would be preferable to the nested list structures of OPS4.

A better expert control architecture might be built around a blackboard architecture (Erman *et al.*, 1981) and a planning module, see Fig.2. The blackboard corresponds to a global database that is available to the different operators. The blackboard should allow information to be represented as objects with associated attributes. The

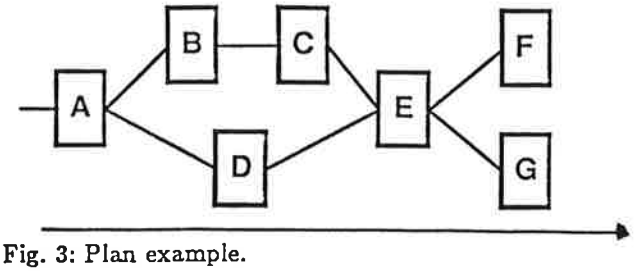

Fig. 3: Plan example.

operators contain the domain knowledge for a certain task. It should be possible to have different problem-solving strategies for different tasks. The operators could be procedural or rule based with different inference strategies depending on their task. This means that the operators could be tailored to their task. They should also be allowed to have their own local databases. The computation of PID parameters from oscillation measurements is an example of an operator task. An operator is often associated with one or more algorithms. An example might be the operator for the relay experiment that contains the domain knowledge for the relay algorithm and the oscillation analyzer algorithm.

The order in which the operators should be used is determined by the planning module. To reach a certain goal state e.g. safe steady-state control, requires in general that a sequence or plan of different operators is used. This plan often contains parallel parts. An example of this is the last part of the plan that contains one operator that takes care of the steady-state control algorithm and one or more operators that handle the monitoring algorithms. An example of a plan is shown in Fig. 3. A plan could be generated in three different ways;

- **Operator-based activation:** The operators start and stop other operators themselves. No separate plan generator is needed in this case.
- **Stored plans:** A finite number of plans for different initial and final goals are stored in the database.
- **Dynamic plan generation:** The plans are dynamically generated by a plan generator. Each operator has an associated set of preconditions that must be fulfilled for the operator to be applicable and a set of goals that will be met by the operator. A plan is generated by comparing the final goal and the initial state with the goals and the preconditions of the operators. This approach can be combined with operator-based activation.

The actual plan is executed by the plan executor. After each stage in the plan the outcome is compared with the expected outcome. Replanning is performed in case of inconsistencies. The dynamic and stochastic nature of control makes this close interaction necessary. The uncertainty in the outcomes of an operator application violates the assumptions of existing domain-independent planning systems (Cohen and Feigenbaum, 1982, Wilkins, 1983). The goal for the separation between the operators and the planning module is to separate the domain knowledge about different tasks from the control information.

The expert control architecture described can in many respects be compared with an ordinary real-time operating system, (e.g. Brinch Hansen, 1973). The operators are the equivalents of concurrent processes and the plan executor is the equivalent of a scheduler. This is especially true in the parallel phases of a plan. In an operating system the processes can wait for a certain time or for a certain event. Similar features can be provided in this architecture by assigning a state with the values running, ready or waiting to each operator. When an operator calls the function "waittime" in a rule it will be marked waiting by the plan executor. The operator is activated when the time is over. An operator can also wait for a specified element to be inserted in the database, e.g. a certain message from the algorithm part.

The operators could be implemented in two ways. The first is to implement them as concurrent Lisp processes that share a global database. This would probably require a Lisp machine. The second way is to implement the system in a single Lisp process. This has the drawback that the operators can not be interrupted.

5. IMPLEMENTATION

The implementation of a system along the lines of Section 4 will now be described. The system is based on the environment described in Section 3 with a new expert system part. This part is built around the forward chaining production system shell YAPS, (Allen, 1983) and the object-oriented Flavors system (Cannon, 1982).

Object-oriented programming provides a convenient way to implement highly modular systems, see e.g. Stefik and Bobrow (1986). Objects consist of a local state and a behavior. Objects are asked to perform operations by sending appropriate messages to them. The objects have associated methods that handles the messages. Generic algorithms can be implemented using object-oriented programming. A protocol i.e. a set of messages is defined, which specifies the external behavior of the object. This protocol does not define the internal implementation. Objects or instances are created by instantiating their descriptions. There are several different object-oriented add-ons to Lisp such as for example Flavors.

YAPS is a forward chaining shell implemented in Flavors. The database contains arbitrarily nested lists of atoms, integers and flavor objects. The condition part of the rules contains patterns that are matched against the contents of the database. Pattern matching variables are allowed. The condition part may also contain predicates acting on the database that must evaluate to true for the rule to be applicable. The action part of the rules contains ordinary Lisp functions. The inference strategy is event-driven forward chaining. The key feature of YAPS for our purposes is the possibility to use flavors objects in the

database and the fact that YAPS is itself a flavor object. This means that YAPS systems can be used as parts of the database of other YAPS systems. These "internal" systems can be controlled from the rules, In this way expert systems can be used "inside" other expert systems. The internal expert systems could also be of other types e.g. backward chaining systems, as long as they are implemented as flavor objects. Operators are implemented in this way.

The flavor objects that YAPS allows in the database can only be matched as object units. It is not possible to access the attributes of an object in the pattern matching. To allow for objects that can be accessed from the rules according to attributes, YAPS has been extended with static objects. A description of each object type has to be given. This includes the attributes with possible default values and an inheritance order. YAPS has also been extended with a limited explanation facility. A description of how the fact was derived is connected to each fact in the database and to each attribute of the facts that are objects. This gives the possibility to ask HOW questions.

The global database and the planning module are implemented in a YAPS system. The planning module is implemented as rules. The operators are implemented as objects in the database. Only YAPS forward chaining operators are presently allowed. The most important attributes of the operator objects are the following;

- **Status:** Take the values active or inactive. The value is active if the operator is used in the current phase of the plan.
- **State:** Takes the values running, ready or waiting. It is used by the plan executor when the operators is active.
- **Instance:** The flavor instance for this operator.
- **Goal:** The goals which the operator should achieve.
- **Preconditions:** The preconditions that are required for the operator to be applied.

The goal and the preconditions are collections of patterns in disjunctive normal form. The possible uncertainty of an operator shows up in the goal expression. Other key elements of the planning module are the actual plan and the goal stack. The actual plan contains the current plan and the goal stack is a stack of goal entries. A goal entry is a conjunction of patterns to be fulfilled. A goal entry may also consist of a collection of conjunction patterns with associated priorities. The plan generator takes the top goal element and attempts to generate a plan for it. If no plan is found then a new attempt is made with a goal of lower priority. This mechanism could be used to implement backup control. When a plan has been generated its execution starts. There is no guarantee that the plan is unique nor that it is the shortest possible plan. After each stage in the plan execution the goals achieved are compared to the planned goals. When a goal entry has been satisfied the next element of the goal stack is selected. A rule in the plan executor could look as follows.

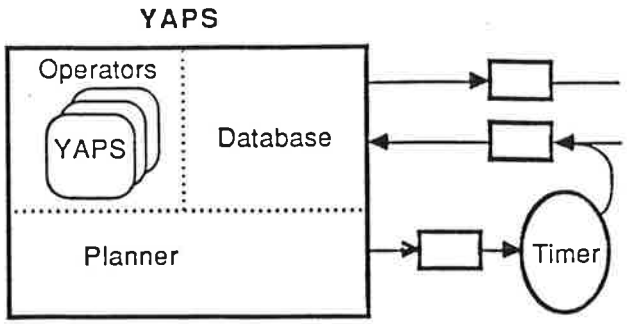

Fig. 4: Implementation structure.

```
(rule schedule1
    (object operator
      status active
      state  ready
      instance -x)
(not (object operator
      state  running))
-->
    (modify 1 state running)
    (<- -x 'run))
```

In natural language this rule says: if there is an operator which is active and ready and no operator is running then this operator is changed to running and a run message is sent to the actual flavor instance.

The plans are represented as nested list structures of operator names. They can be expressed in the following EBNF syntax where p and s denote parallel and sequential, respectively.

```
PLAN     ::= (OPERATOR ARGUMENT [ARGUMENT..])
OPERATOR ::= <p> | <s>
ARGUMENT ::= <operator-name> | PLAN
```

The example in Fig. 3 looks like (s A (p (s B C) D) E (p F G)) in this notation. The rest of the elements in the global database are used for domain information. The knowledge about the actual control loop is well suited to be represented as objects with different attributes.

The operators built on YAPS have predefined functions for activation and deactivation of other operators. They have also functions for waiting a certain time or for a certain element to be inserted in the global database. Each waiting function has two different versions. One that requests a wakeup and suspends the rule execution and one that only requests a wakeup. Furthermore, there are functions for adding and deleting facts locally as well as globally and for pushing and popping elements on the goal stack.

The waittime requests are handled by a separate timer process. When a waittime is requested a message is sent to an associated mailbox. The requests are queued by the process and a message is returned to the expert system when the requested time is over. The structure of the implementation is shown in Fig. 4.

The next step in the implementation will be to add other operators than YAPS. Work on this is currently under progress and will be described in further papers.

6. CONCLUSIONS

The expert system approach simplifies the implementation of controllers based on process knowledge. The knowledge required can be acquired from the process operator or from the process. Extracting knowledge from the process requires experiments. This means that different algorithms are applied to the plant. An expert system is well suited for implementation of the logic needed in this process.

A prototype environment has been built up. Experiment with conventional expert system shells have shown that they are not well suited for expert control. An architecture that is better suited is described. This architecture is under implementation using YAPS and object-oriented programming.

ACKNOWLEDGEMENT

I would like to thank my supervisor Prof. Karl Johan Åström for many useful discussions. The project is part of the Computer-Aided Control Engineering Project at the Department of Automatic Control, Lund Institute of Technology. It is supported by STU, the National Swedish Board for Technical Development under contract 85-3084.

REFERENCES

Allen, E.M. (1983): YAPS: Yet another production system. TR-1146, Department of Comp. Sc., Univ. of Maryland.

Allen, J.F. (1984): Towards a General Theory of Action and Time. *AI Journal* 23 pp. 123-154.

Årzén, K-E. (1986a): A Real-Time Environment for Expert Control. Report TFRT-7314, Dep. of Automatic Control, Lund Institute of Technology, Sweden.

Årzén, K-E. (1986b): Expert Systems for Process Control. In Proc. 1st Int. Conf. Appl. of AI in Engng. Pract., Southampton, U.K.

Åström, K.J. (1983): Theory and applications of adaptive control. *Automatica* 19, pp. 471-486.

Åström, K.J. and J.J. Anton (1984): Expert Control, Proc. 9'th IFAC World Congress, Budapest, Hungary.

Åström, K.J. and T. Hägglund (1984): Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica* 20, pp. 645-651.

Åström, K.J. and B. Wittenmark (1973): On self-tuning regulators. *Automatica* 9, pp. 185-199.

Birdwell, J.D., J.R.B. Cockett, R. Heller, R.W. Rochelle, A.J. Laub, M. Athans and L. Hatfield (1985): Expert systems techniques in a computer based control system analysis and design environment. Proc. 3rd IFAC/IFIP Int. Symp. CADCE '85, Lyngby, Copenhagen, Denmark.

Brinch Hansen, P. (1973): *Operating System Principles.* Prentice Hall, Englewood Cliffs, N.J.

Cannon, H.I. (1982): Flavors: A non-hierarchical approach to object-oriented programming, unpublished paper, Artificial Intelligence Laboratory, MIT, Cambridge, MA.

Clark, D.W. (1984): Implementation of adaptive controllers. In Harris and Bildings. (Eds), *Self-tuning and Adaptive Control.* Peter Peregrinus, U.K.

Cohen, P.R. and E. Feigenbaum (1982): STRIPS and ABSTRIPS. Chapter 15B of *The Handbook of Artificial Intelligence* Vol. 3, William Kaufman Publishing, Los Altos, Ca.

Crossman, E.R.F.W. and J.E. Cooke (1962): Manual Control of Slow-Response Systems, in E. Edwards and F.P. Lees, *The human operator in process control*, London: Taylor and Francis, Ltd., 1974.

Elmqvist, H. (1975): SIMNON, An Interactive Simulation Program For Nonlinear Systems, Report TFRT-3091, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Erman, L.D., P.E. London and S.F. Fickas (1981): The design and an example use of HEARSAY-III. Proc. IJCAI 7 pp. 409-415.

Forgy, C.L. (1979): OPS4 User's Manual. Tech. Rep. CMU-CS-79-132, Department of Comp. Sc., Carnegie-Mellon Univ., USA.

Hayes-Roth, F., D. Waterman and D. Lenat (1983): *Building expert systems.* Addison-Wesley, Reading, MA.

Isermann, R. (1982): Parameter adaptive control algorithms – A tutorial. *Automatica* 18, pp. 513-528.

James, J.R., J.H. Taylor and D.K. Frederick (1985): An expert system architecture for coping with complexity in computer-aided control engineering. Proc. 3rd IFAC/IFIP Int. Symp. CADCE, Lyngby, Copenhagen, Denmark.

Larsson J.E. and K.J. Åström (1985): An Expert System Interface for IDPAC, Proc. of 2nd IEEE Control Systems Society Symposium on CACSD, Santa Barbara, CA.

McDermott, D. (1982): A temporal logic for reasoning about processes and plans. *Cognitive Science* 6 pp.101-157.

McDermott, D. and J. Doyle (1980): Non-monotonic logic I *AI Journal* 13.

Moore, R.L., L.B. Hawkinson, C.G. Knickerbocker and L.M. Churchman (1984): A Real-Time Expert System for Process Control, Proc. First Conf. on AI Applications, pp. 529-576, IEEE Computer Society, Denver, Colorado.

Moore, R.L., L.B. Hawkinson, M.E Levin and C.G. Knickerbocker (1985): Expert Control. In Proc. ACC. pp. 885-887, Boston, MA.

Nelson, W.R. (1982): REACTOR: An expert system for diagnosis and treatment of nuclear reactor accidents, in Proc. of the National Conference on Artif. Intell., pp. 296-301, Pittsburgh, PA.

Palowitch Jr., B.L. and M.A. Kramer (1985): The application of a knowledge-based expert system to chemical plant fault diagnosis. In Proc. ACC pp. 646-651, Boston, MA.

Sanoff, S. P. and P. E. Wellstead (1985): Expert Identification and Control. Proc. IFAC Identification and System Parameter Estimation pp. 1273 - 1278, York, UK.

Stefik, M. and D.G. Bobrow (1986): Object-oriented Programming: Themes and Variations. *AI Magazine* Vol. 6, No. 4.

Trankle, T.L. and L.Z. Markosian (1985): An expert system for control system design. Proc. IEE Int. Conf. Control 85, Cambridge, UK.

Waterman, D.A. (1986): *A Guide to Expert Systems,* Addison-Wesley.

Wilkins, D. (1983): Domain-Independent Planning: Representation and Plan Generation. Tech. Note 266R, SRI International, Menlo Park, CA.