INTERNATIONAL JOURNAL OF ADAPTIVE CONTROL AND SIGNAL PROCESSING, VOL. 5, 21–39 (1991)

AN AUTOSYNTHESIZING NON-LINEAR CONTROL SYSTEM USING A RULE-BASED EXPERT SYSTEM

JAMES H. TAYLOR AND PHILIP G. STRINGER*

GE Corporate R & D Control Systems Lab., Bldg KW, Room D209A, PO Box 8, Schenectady, NY 12301, U.S.A.

SUMMARY

An *advanced non-linear control system* is obtained by combining recent developments in non-linear control system synthesis with a rule-based system approach to real-time control. The basic problem to be addressed is the control of a non-linear plant which is sufficiently sensitive to both operating point and input amplitude that the desired control system performance can be obtained only with a non-linear controller that is 'retuned' or 'resynthesized' whenever the operating point changes significantly. (The term 'changes significantly' in this context signifies an operating point change that causes the non-linear control system input/output behaviour to change substantially in an undesirable way.) This is accomplished by a *hierarchically organized intelligent control system* with a conventional reprogrammable controller under the direction of a rule-based expert system. The function of the expert system is to

(i) monitor the behaviour of the non-linear control system to determine when retuning or resynthesis is required; if the behaviour is satisfactory, then continue passive monitoring; else
(ii) when retuning or resynthesis is required: set up and execute experiments to derive the model information required to tune or synthesize a new non-linear controller in terms of a given structure and parametrization; execute the retuning/resynthesis procedure; reprogramme the controller (download the parametrization); and recommission the updated non-linear control system and return to normal operation.

In essence, the rule-based system provides supervisory control ('meta-control') for a conventional reprogrammable non-linear controller that will autotune or autosynthesize as required. Autosynthesis takes place to accommodate variability in plant behaviour due to operating point changes, and the non-linear controller thus synthesized corrects for amplitude sensitivity. This concept represents one way to combine artificial intelligence with control; it will be discussed and illustrated by example below.

KEY WORDS AI in real-time control Intelligent control Expert systems for control Heuristic programming

1. INTRODUCTION

This presentation focuses on developing *advanced control systems* via the use of *rule-based systems for real-time control*. The approach and methodology described below are the results of several exploratory studies in which we pursued the following questions: how can one make effective use of expert systems technology in control, and what are the costs and benefits of this methodology? We have carried out this investigation by adopting a specific architecture

* Present address: James River Corp., Camas, Washington, U.S.A.

0890–6327/91/010021–19$09.50

Received May 1989
Revised May 1990

for AI-based control, developing software to support this architecture, and modelling and simulating several examples; these aspects are discussed in detail in the main sections of this paper. First, some general concepts and comments are given.

Rule-based systems are software environments that can be used in developing real-time control systems, relieving the system engineer of much of the burden of taking a 'textbook design' (e.g. non-linear control algorithm) and making it work in a real-world application. The specific problem addressed is the well-known fact that obtaining a control algorithm is often a small percentage of the control engineer's overall effort; implementing it (making it work in terms of system interfaces, initialization procedures, logic, exception handling, operator interface, etc.) is usually the more challenging part of the engineering task. This is especially true with advanced systems designs where complicated features such as adaptation algorithms, failure detection and isolation schemes, parameter identification software, etc. must be embedded as an integral part of the control system software.

There is often no standard or even systematic approach to implementing a controller in this sense; rather, the engineer adds logic and other code 'patchwork' to the controller software until it works in the real system. If the system is complicated, the resulting logic and algorithmic modifications may become a large and unwieldy mass of 'spaghetti code' that is difficult to implement, test, validate, document and maintain.

The rule-based systems approach can be used to provide a great deal of support for the above task, resulting in a more flexible system implementation with less iteration and greater likelihood of success. In essence, the rule-based system provides the environment in which to develop and test all of the required heuristic logic and control, using a programming paradigm (rule-based systems) and language (rule writing) that is ideally suited for the task. An appropriate expert system shell then provides the software framework in which the rule-based system operates to carry out the overall control system functionality.

The application of expert systems methodology outlined above represents one type of built-in expertise, namely embedded design knowledge. A second usage involves situations where AI-based approaches are used to alleviate problems that presently arise from closing or supervising control loops via the actions of human operators. The problems solved by expert systems or other AI techniques in this case include excessive operator workload, requirements for speed of response that stress or exceed human capability, human sensory or decision-making overload in the case of emergencies in complicated systems, etc. We have primarily considered problems associated with implementation/design knowledge, although the same general methodology is applicable in both situations.

The resulting rule-based real-time control (RBRTC) system may actually be implemented as a rule-based system or it may be 'compiled' into a lower-level language if the flexibility of the rule-based environment is no longer needed in the target application. Compiling the meta-controller results in a system that is typically faster in execution and that can be installed on a less expensive host processor. Either way, our experience has been that the development and implementation effort has been greatly simplified and the full control system implementation can be maintained and documented effectively in rule-based form.

The balance of this paper is organized as follows. Section 2 deals with a general architecture for rule-based real-time control. Section 3 outlines the development environment used in this and related studies including the expert system knowledge representation and inference mechanisms. Section 4 summarizes the theoretical bases for this research, namely a linear autotuning scheme and several non-linear control synthesis approaches based on describing function methods. Section 5 describes rule-based systems for autotuning and autosynthesizing non-linear control systems. Section 6 illustrates the performance of an autosynthesizing non-linear control system. Finally, Section 7 provides a discussion of our findings and conclusions.

2. AN ARCHITECTURE FOR RULE-BASED REAL-TIME CONTROL

There are various models for the combination of expert systems and control technology. Our paradigm [1,2] is a *rule-based real-time control (RBRTC) system* having the following component parts:

(i) *sensors* (monitors) to determine the state of the process in terms of 'signals', e.g. sensor outputs.
(ii) *pattern recognizers* or feature extractors to process signals and create 'symbols' or linguistic representations of the information, e.g.
(TRANSIENT %OVERSHOOT . EXCESSIVE), (TRANSIENT RISETIME . 1·253)
(iii) *conventional programmable controllers* with interfaces permitting gain changing, state resetting, reconfiguration, etc.
(iv) *rule bases* that contain the overall control strategy for all regimes (embedded knowledge of the control system designer) and
(v) *an expert system shell or 'inference engine'* that exercises the higher-level control ('meta-control') of the system.

An RBRTC system architecture that incorporates these elements was described in detail in References 1 and 2 and is portrayed here in Figure 1. This configuration provides a natural control hierarchy that embeds the expertise of the system designer (or, in some contexts, a capable human operator) in the higher layer while making the best possible use of conventional control technology at the lower layer. Note that there may be many levels in such a hierarchical scheme (more levels in the meta-control layer and/or more levels in the conventional control layer); for example, the systems described in Section 5 are partitioned with a meta-control layer that is comprised of a supervisory rule base and a number of lower-level rule bases.

Note that there are varying degrees of 'realness' in RBRTC systems. The fastest and most difficult implementations are those in which the AI portion of the system must respond in time intervals comparable to the time constants of the plant dynamics or sampling interval of the conventional controller. More often the rule-based system is performing supervisory tasks that permit much slower operation in comparison with plant dynamics and sampling times. In the example described here the autosynthesis RBRTC is dealing with real-time operations and signals but does so when the process is off-line.

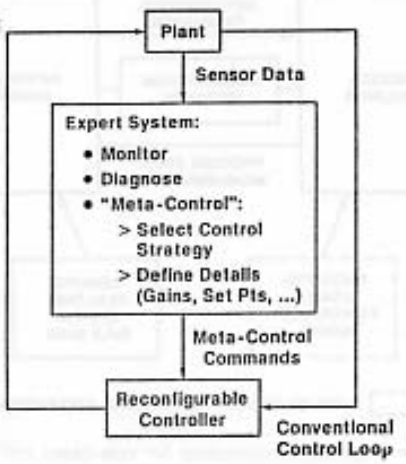


Figure 1. Architecture for rule-based real-time control

This general concept is quite similar to other approaches; see References 3–6 for a sampling of related work. It is also distinctive from other approaches to 'intelligent' control, for example fuzzy control (see e.g. Reference 7) and the use of neural networks (see e.g. Reference 8).

3. A DEVELOPMENT ENVIRONMENT FOR RULE-BASED REAL-TIME CONTROL

3.1. Overall software environment

The architecture outlined above was developed first as a vehicle for studying a failure detection and isolation methodology.[9] In that same effort[1] we developed a software framework for experimenting with rule-based control applications; for more detail see Reference 2. An overview of this framework (a real-time control simulation environment with an embedded rule-based system) is depicted in Figure 2. It incorporates a standard simulation environment for continuous- and discrete-time systems (in which one can model the plant and conventional reconfigurable controller) coupled to an expert system shell (inference engine) where the rule-based system is implemented in terms of generic real-time control logic and application-specific meta-control rules.

The simulation environment used in this framework is Simnon.[10] The capabilities that we needed for this application are generally the same features required for any flexible, general-purpose non-linear simulation environment. In particular, Simnon provides a powerful modelling capability for continuous- and discrete-time non-linear systems and a command-driven interface that facilitates the control of the simulation itself (start time, end time, integration step and algorithm), manipulation of the model (e.g. changes to the plant model, downloading controller parameter values) and monitoring the variables in the model.

The expert system environment used in Figure 2 is a GE-developed shell called Delphi. This environment supports production rules with inference performed via a programmable combination of forward and backward chaining and basic numerical operations (e.g. testing variables against threshold values). Delphi is an interpretive Lisp-based environment that is completely open (customizable via the introduction of additional Lisp code); we found this

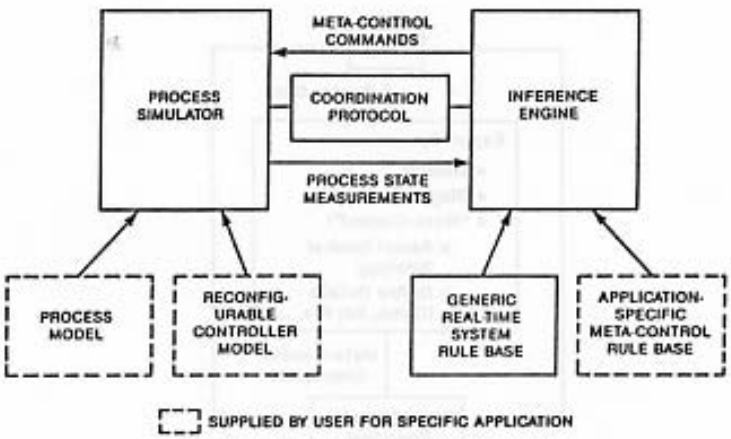


Figure 2. Development environment for rule-based real-time control

attribute to be extremely valuable in constructing the pseudo-real-time environment shown in Figure 2. For example, we took advantage of this capability to extend the shell to accommodate multiple rule bases that may be interchanged and managed by a supervisory rule base.

The coupling of these software packages, as portrayed in Figure 2, is managed via a simple co-ordination protocol so that the simulator stops at each meta-control sample time, control passes to the rule-based system which samples the signals in the simulation and performs meta-control tasks, and then the simulation continues when the expert system has completed its part of the cycle. This protocol is a direct adaptation of the approach used in a study of the use of AI methodologies for computer-aided control engineering (CACE).[11,12] From the controls simulation point of view the expert system is just another discrete-time module. We emphasize that this development environment is not a real-time implementation, just an extended simulation tool for RBRTC.

3.2. *Rule-based expert system environment*

An expert system provides a *structure* and *machinery* for organizing and manipulating (processing) knowledge. The Delphi expert system shell has the architecture depicted in Figure 3, which is comprised of a *knowledge processor* or 'inference engine' which receives information from *knowledge sources* (the user, data from storage, sensor data and perhaps data generated by running other software). This information is processed according to heuristic *inference strategy* and a set of *production rules*. This processing involves a dynamic data structure called a *list of facts* which changes in size and content as knowledge is processed.

More specifically, the constituents and operation of the Delphi knowledge-based system may be summarized as follows.

1. *Facts* — information processed by Delphi is represented in the form of a three-tuple, designated (OBJECT ATTRIBUTE . VALUE), e.g.
   (LIN_PID DERIV-GAIN . 15·2)
   (AUTO-TUNING TERMINATION . SUCCESSFUL)

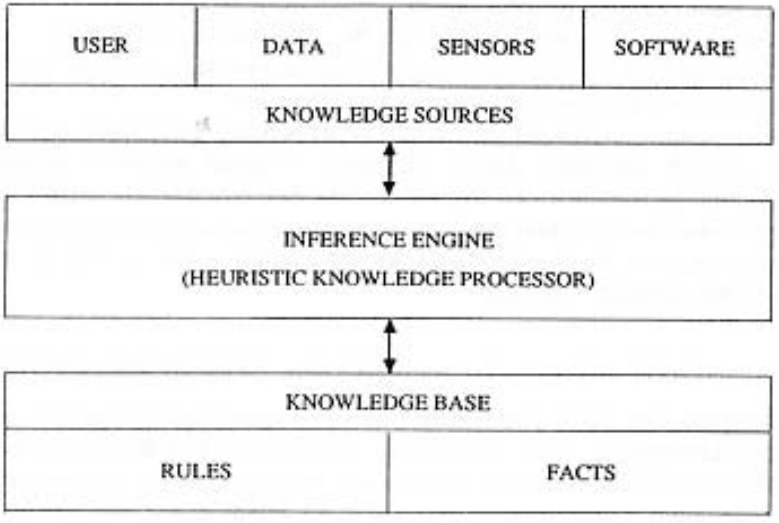


Figure 3. Delphi expert system shell architecture

The input from any knowledge source has to be translated into this format (a process called 'signal-to-symbol transformation'). Facts are also written, modified and/or cleared by the inference engine.

2. *Rules* — facts are manipulated by rules, which are of the form

   (IF(PREMISE$_1$, PREMISE$_2$, ...)
   THEN (ACTION$_1$, ACTION$_2$, ...))

   where

   (a) PREMISE = (⟨PREDICATE⟩ ⟨PATTERN⟩);
   PREDICATE = TRUE, FALSE, NTRUE, NFALSE, TEST, ...
   PATTERN = THREE-TUPLE

   (b) ACTION = (⟨VERB⟩ ⟨PATTERN⟩)
   VERB = WRITE, CLEAR, PROVE, ASK, DO, PROG, ...
   PATTERN = THREE-TUPLE

3. The following is an example rule. Assume that previous processing via the expert system has established the facts %__OSHOOT VALUE . 17·9 and then LIN__PID %__OSHOOT . GT·15 — this would cause the following rule to fire

```
(RULE 110 ("Overshoot is excessive")
  (IF
    (TRUE (LIN__PID %__OSHOOT . GT·15)
    "linear PID design yields %overshoot > 15%")
  THEN
    (WRITE (LIN__PID %__OSHOOT . EXCESSIVE))))
```

4. *Uncertainty* may also be included in the Delphi production rule formalism by the use of certainty factors. We have not used this feature so far in our applications. One open question is how uncertainty is to be propagated in the inference process.

5. Knowledge processing is done via various 'inference' modes.

   (a) *Forward chaining* or *data-driven processing* (e.g. OPS-5)
   Given: the current *list of facts*; rules are searched to find those whose premises are satisfied and such rules are executed (*'fired'*). The inference engine may continue until exhaustion or there may be limiting mechanisms to constrain the search.

   (b) *Backward chaining* or *goal-driven processing* (e.g. MYCIN)
   Given: a *goal* (e.g. write (LIN__PID %__OSHOOT . OK)); find rule(s) that can accomplish it; if the corresponding premises are satisfied, fire the rule; if not, define its unsatisfied premises as subgoals and continue.

   Some environments implement one mode of deduction; others use both with suitable inference control strategies. Delphi supports a mixed inference strategy. We use backward chaining to control an overall process; for example, the RBRTC 'Supervisor' rule base invokes the secondary rule bases to carry out monitoring, autosynthesis and recommissioning (see Section 5) via backward chaining. The secondary rule bases are designed to use forward chaining.

4. LINEAR AUTOTUNING AND NON-LINEAR SYNTHESIS APPROACHES

The basic idea behind non-linear autotuning[13] and non-linear autosynthesis is that there is a natural symbiosis between the linear autotuning scheme (LAS) of References 14 and 15 and certain non-linear controller synthesis approaches (NCSAs) based on describing function methods.[16,17] The main connection is that the LAS cited above involves identifying the frequency response of an unknown linear plant as the basis for controller gain adjustment and

that the NCSAs require frequency domain input/output characterizations of non-linear systems as the basis for non-linear controller synthesis. Therefore this LAS provides the basis for automatically executing an NCSA for an unknown non-linear plant.

4.1. Linear autotuning

A generic definition of an automatic tuning process may be stated as follows. The system is first configured so that the unknown plant is excited by a suitable signal generator, relevant plant dynamic descriptors are estimated from the response of the plant to the excitation and regulator parameters are calculated on the basis of these descriptors. The regulator parameters are then downloaded into a controller, the signal generator, estimator and module for design calculation are disconnected and the control system commences to operate like an ordinary fixed regulator. This process may be repeated as necessary (e.g. whenever it has been determined that the fixed regulator is not performing adequately).

Linear autotuning schemes based on features of the Nyquist locus of the open-loop transfer function can be extended most readily to the non-linear case using the frequency domain NCSAs mentioned above. Typically, knowledge of the critical point, i.e. the first point where the Nyquist curve intersects the negative real axis, is used. This point can be characterized by the critical gain K_c and the critical frequency ω_c. The Ziegler–Nichols algorithm is a typical example of a design method based on the critical point. Other points on the Nyquist locus may be useful for autotuning design as well.

To use such design methods for autotuning, it is necessary to find an identification or estimation method which determines the critical point. This could be done by supplying a sinusoidal input to the plant and sweeping over frequency until a phase shift of $-\pi$ radians is obtained. Such an approach is, however, time-consuming and difficult to implement.

A major contribution of Reference 14 is a new method for automatically determining points on the Nyquist curve. It is based on the observation that a system with a phase lag of at least π radians at high frequencies will usually oscillate when a relay is introduced in the control loop. To determine the point, the system is connected in a feedback loop with a relay in series with the plant, as shown in Figure 4. The error e is then a periodic signal and K_c and ω_c can be determined approximately from the first harmonic component of the oscillation:

Let D be the relay output level and let a be the amplitude of the first harmonic of the error signal. Then[18] the describing function of the relay is $N(a) = 4D/\pi a$ and the describing function condition for oscillation at frequency ω_c is $N(a) * G(j\omega_c) = -1$. Since a and thus $N(a)$ are known, the critical gain for the plant is simply given by

$$K_c = \frac{-1}{G(j\omega_c)} = N(a) \tag{1}$$

The critical frequency ω_c is also known from observation of the oscillation period T_0, i.e. $\omega_c = 2\pi/T_0$.

The period of the oscillation can easily be determined by measuring the times between zero crossings. The amplitude may be determined by measuring the peak-to-peak values. These estimation methods are very easy to implement because they are only based on counting and comparisons. More elaborate estimation schemes may also be used to determine the amplitude and the period of the oscillation. The method outlined above has, however, been shown to work well in practice.[14]

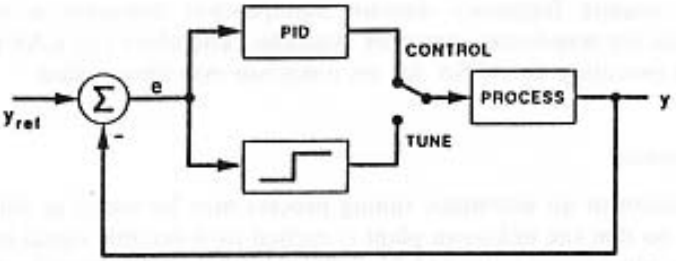


Figure 4. Linear autotuning system schematic

This estimation method will automatically generate a plant input signal with significant frequency content at ω_c, ensuring that the point can be determined accurately. The only *a priori* information required is that the process engineer should select D and thus the input signal amplitude to excite the plant properly without creating dangerous excursions in the plant variables.

Other points on the Nyquist curve can be estimated by introducing known dynamics and/or hysteresis (positive or negative) in the relay. This may be useful either in cases where the plant does not oscillate with just a pure relay in the closed-loop system or to obtain further information about the frequency domain behaviour of the plant. The introduction of hysteresis was used in the algorithm in Reference 13; the required condition for oscillation[19] is then

$$\frac{-1}{N(a)} = \frac{-\pi}{4D}\,[\sqrt{(a^2 - h^2)} + \mathrm{j}h] = G(\mathrm{j}\omega_0) \tag{2}$$

where $\pm h$ defines the hysteresis break points (the hysteresis width is $2h$). Knowing the relay parameters and the oscillation amplitude a thus allows the direct calculation of $G(\mathrm{j}\omega_0)$.

The use of relay-induced oscillations to determine the frequency domain characteristics of a non-linear system was proposed in Reference 13. This is done via relay experiments with different relay levels D_i and different hysteresis points $\pm h_i$. If the system is nearly linear, the amplitude of the oscillation is approximately proportional to D_i and the LAS will suffice for autotuning; if not, the various levels will generally yield different behaviour and the application of an NCSA may be required. These experiments are thus diagnostic as well as supporting non-linear controller synthesis if necessary. In the context of non-linear systems this scheme for autosynthesis may be used to accommodate changes in behaviour due to (slowly) varying *operating points* as well as *uncertainty* in the plant dynamics or (slow) changes with *time* or *environmental effects*.

4.2. Non-linear controller synthesis

The basic idea exploited in References 16 and 17 is that sinusoidal input describing function (SIDF) models of a non-linear plant provide a good basis for the design of linear and non-linear compensators. Two methods have been suggested for obtaining such models: the derivation of analytic SIDF models of the dynamics plus the solution of equations of harmonic balance, and simulation of the non-linear plant with sinusoidal inputs plus Fourier analysis of the output signal to determine the input/output (I/O) gain and phase relations. Further references on SIDF modelling for non-linear systems include References 18–20.

Using either technique, the designer models the plant by assuming that the input to the plant is sinusoidal with amplitude a and frequency ω and obtains a transfer function representing the I/O relation in the form $G(j\omega;a)$. Several compensator synthesis methods based on SIDF I/O models are given by Taylor and Strobel.[16,17]

The non-linear compensator synthesis method of Reference 16 was used in Reference 13; it proceeds as follows.

(a) Select a nominal input amplitude a_0, obtain $G(j\omega;a_0)$ and design a nominal linear compensator C_0 based on that model. Denote the combined I/O relation CG_0.

(b) Determine a set of SIDF I/O models for a set of error signal amplitudes $\{e_i, i = 1, 2, \ldots\}$ for the compensated plant, denoted $\{GC(j\omega;e_i)\}$ or, more simply, $\{CG_i\}$.

(c) Inspect the behaviour of $\{CG_i\}$ near the critical point; if there are large differences in CG_i for various error amplitudes, then synthesize a non-linearity that minimizes the variation as much as possible. A particular implementation of this step is as follows.[16]

(i) Determine the M-circle that is just tangent to CG_0.

(ii) For each i determine the gain K_i such that K_iCG_i is just tangent to that same M-circle.

(iii) Use the information $\{K_i(e_i)\}$ as the basis for synthesizing the non-linearity $f_c(e)$ that must be placed before the compensator to desensitize the overall open-loop I/O relation in the frequency domain. Synthesis is carried out by SIDF inversion, i.e. by finding the parameters of a piecewise-linear function so that the SIDF of that non-linearity fits the points $\{K_i(e_i)\}$ as closely as possible (a minimum mean square error criterion is used in Reference 16).

The procedures in step (c) are illustrated in Figures 5 and 6, which are taken directly from Reference 16.

It is noteworthy that the NCSAs of References 16 and 17 involve desensitizing the *open-loop frequency response*; validation of these approaches in the time domain by simulation showed that the *step response* of the resulting closed-loop system is also much less sensitive to input amplitude than in the case of linear control of the same plant. This may be observed in the example provided in Section 6 (Figure 11).

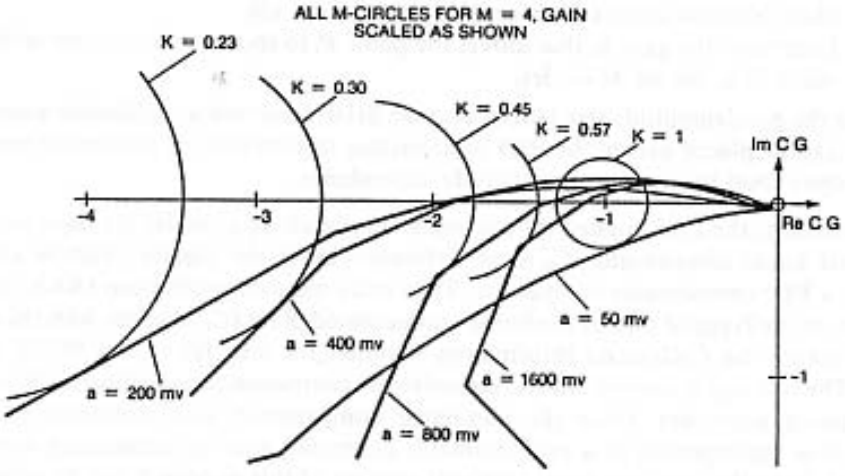


Figure 5. Non-linearity synthesis—determining gain/amplitude

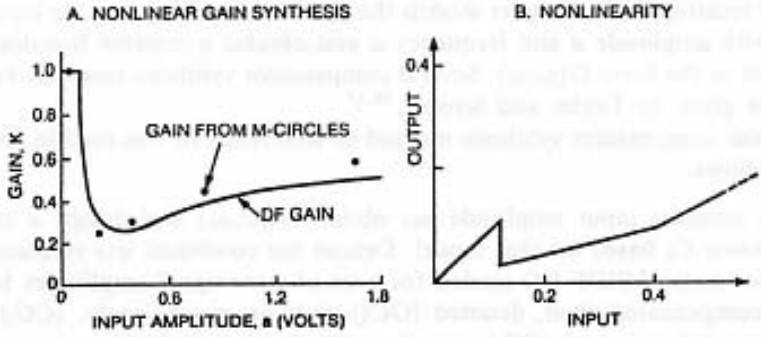


Figure 6. Non-linearity synthesis—SIDF inversion

4.3. *Non-linear autotuning*

One way in which the linear autotuning scheme (LAS) presented in References 14 and 15 can be extended to the non-linear case is by combining it with the non-linear compensator synthesis approach (NCSA) outlined in the preceding paragraphs. This leads to the following non-linear autotuning algorithm from Reference 13.

(a) Install a relay of nominal output level D_0 and hysteresis h_0 in series with the unknown plant to be controlled; note that the square wave has first-harmonic amplitude $a_0 = 4D_0/\pi$; close a unity-gain feedback loop around this system.

(b) Generally the close-loop system will exhibit limit cycle oscillations; determine the frequency domain descriptors (critical gain and frequency, equations (1, 2)) and design a nominal linear PID controller C_0 as in the above LAS.

(c) Apply the previously described NCSA; obtain $\{K_i(e_i)\}$ for the open-loop system with linear control for a reasonable number of cases, $i = 1, 2, \ldots, m$, as follows.

  (i) Install a relay output level D_i plus hysteresis h_i in series with C_0 and the unknown plant; close a unity-gain feedback loop around this combination.

  (ii) Adjust the hysteresis to obtain limit cycles; the frequencies and amplitudes of the oscillation at the plant output give points P_i on the linear-compensated non-linear plant Nyquist locus CG_i according to equation (2).
  Determine the gain K_i that moves the point P_i to the desired location in the Nyquist plane (e.g. on an M-circle).

(d) Take the gain/amplitude sets $\{K_i(e_i)\}$ and use SIDF inversion to synthesize a non-linearity that can be placed before the PID compensator to eliminate or reduce the sensitivity of the open loop to non-linear amplitude dependence.

More generally, the LAS applied for a nominal amplitude a_0 provides the basis for designing the nominal linear compensator C_0 as in Reference 16; if the Ziegler–Nichols algorithm is used, then a PID compensator is obtained. Then relay-induced oscillations (RIOs) are used to find points on the Nyquist plot of the linear-compensated plant (C_0 in series with the non-linear plant) to obtain the $C_0G(j\omega;e_i)$ information required for step (c) of the NCSA previously outlined. The last step is carried out to synthesize the compensator non-linearity f_c to complete the autotuning algorithm. Once the non-linear compensator $f_c(e)$ followed by $C_0(j\omega)$ is designed, it is implemented in a programmable controller and the autotuning mechanism is deactivated. Note that the non-linear synthesis portion of this approach can be applied to any compensator type, not merely PID.

4.4. Non-linear autosynthesis

The NCSA summarized in Section 4.2 is based on the general idea that we first determine a set of linear controllers which achieve a single frequency domain objective for a variety of input amplitudes and then we coalesce these linear controllers into a single amplitude-desensitizing non-linear controller by the use of SIDF inversion.[18] The methods presented in References 16 and 17 do this in a 'batch' process, i.e. the linear gains are determined from previously evaluated frequency domain models of the plant or open-loop system.

During the course of testing and refining the non-linear autotuning algorithm presented in Section 4.3 in our RBRTC development system (Section 5.1), it occurred to us that we could determine the linear gains via on-line adjustment, i.e. that we could induce RIO and adjust controller gains until the RIO limit cycle conditions meet a specified frequency domain objective directly. This has the major advantage that far fewer RIO experiments need to be performed during on-line parameter adjustment than would be needed to implement the NCSA in Reference 17 using a batch-processing approach. This idea was the basis for the algorithm presented in Section 5.2.

The SIDF basis for the non-linear autosynthesis approach described in Section 5.2 is identical to that presented above. The only SIDF results required in that algorithm are provided in equation (2). The details of this NCSA algorithm, however, differ substantially from those in References 16 and 17.

5. RULE-BASED REAL-TIME NON-LINEAR CONTROL

Our most ambitious applications of AI-based controls began with the development of a rule-based system implementation of the non-linear PID autotuning algorithm from Section 4.3.[13] This non-linear controller tuning approach was selected because it was well known to us (we had the 'designer expertise' required to programme the expert system) and it was clearly of sufficient complexity to challenge the developers. An outline of this system is provided in Section 5.1.

The completion and study of the rule-based non-linear autotuning controller provided us with new ideas for non-linear controller synthesis. The high-level AI programming environment in our development system encouraged us to programme these ideas to demonstrate their feasibility. The resulting system is described in Section 5.2

5.1. Rule-based non-linear autotuning

An overview of the rule-based real-time control (RBRTC) system implementation for non-linear autotuning is shown in Figure 7. It features a supervisory RBRTC rule base (RB) plus lower-level RBs partitioned as follows.

(i) A *Performance-monitoring rule base (unbuilt)* which samples the transient response of the control system and decides if returning is needed.

(ii) An *autotuning rule base* which implements the algorithm in Reference 13 as follows.

(a) *Plant identification set-up* — reconfigure the system by replacing the previous PID controller with a hysteretic relay (toggle the control/retune switch in Figure 7 to the 'retune' position and reprogramme the controller to be a linear unity gain), thus preparing for the production of relay-induced oscillations (RIOs) for plant model identification in the frequency domain; a nominal value of the relay output level D_0 is used to obtain the plant frequency response at a nominal amplitude $a_0 = 4D_0/\pi$.

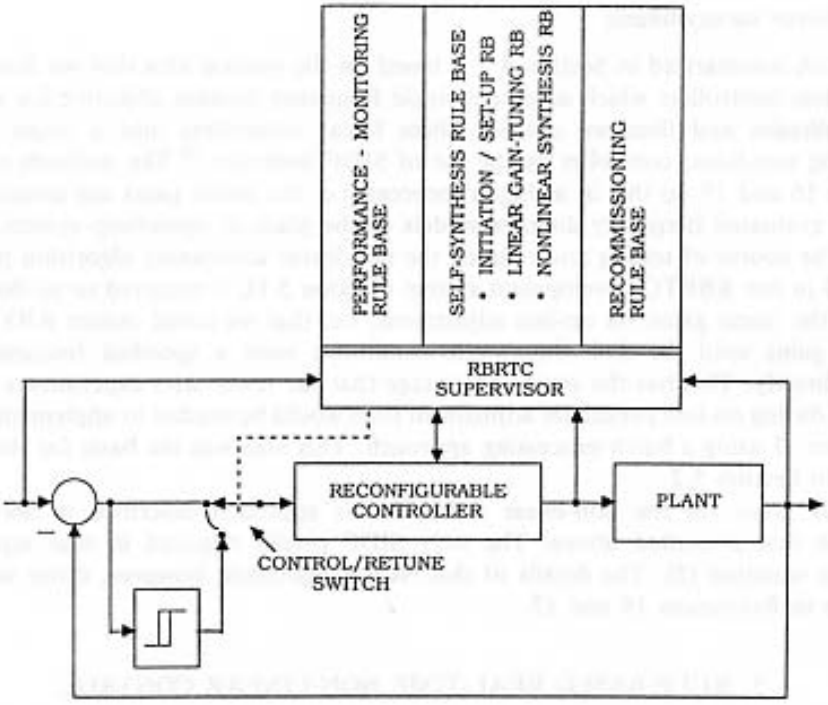


Figure 7. RBRTC for non-linear autotuning

(b) *Plant identification execution* — carry out one set of RIO experiments for this nominal amplitude to obtain the corresponding magnitude and phase of the non-linear plant frequency response for several values of hysteresis (this determines several points on the nominal plant SIDF 'Nyquist' locus).

(c) *Linear controller synthesis* — synthesize the corresponding nominal linear controller using the above SIDF data (e.g. design a PID controller via Ziegler–Nichols tuning for the nominal amplitude); note that the linear controller type is arbitrary.

(d) *Open-loop identification set-up* — download the parameters for the new linear controller and prepare to induce RIOs (keeping the control/retune switch in the retune position) for open-loop system identification in the frequency domain (different relay output levels D_i are used to obtain the plant frequency response at different amplitudes a_i).

(e) *Open-loop identification execution* — carry out a set of RIO experiments to obtain the corresponding magnitude and phase of the linear controller in series with the non-linear plant as the open-loop system responds to different amplitude periodic inputs (different relay output levels).

(f) *Non-linearity synthesis* — determine the *desired* gain versus amplitude relation $K_i(a_i)$ that cancels the sensitivity of the open-loop response with respect to amplitude; this is done by determining the gain that causes the open-loop frequency response to be just tangent to a given M-circle.[16] Then synthesize a compensating piecewise-linear function that can be placed before the controller to *reduce* the open-loop amplitude sensitivity determined above by fitting the gain/amplitude

relation $K_i(a_i)$ in a minimum mean square sense using SIDF inversion; reprogramme the controller (download the parameters that define the controller non-linearity).

(iii) A *recommissioning rule base* which puts the new non-linear PID controller into service once retuning is complete via the retune/control switch.

The conceptual functionality of the performance-monitoring rule base is to determine when autotuning becomes necessary. The idea was to process the plant input and output signals to extract transient response metrics (e.g. % overshoot) and to compare these with their desired values or ranges; autotuning would be commanded whenever the metrics were out of tolerance. Similar signal feature extraction had been performed in an earlier study of RBRTC;[1] we thus considered this rule base to be of secondary interest and never built it.

Given a (hypothetical) determination that the control system is not performing satisfactorily, the control system goes through a period of retuning under the supervision of the autotuning rule base. This was programmed to be an exact duplication of the procedure presented in Reference 13, as outlined in more general terms above, so the details of this algorithm are not repeated here. We note, however, that the non-linearity synthesis rules are quite involved.

(i) The determination of a gain K_i such that K_iCG_i is just tangent to a specified M-circle entails programming rules that process the data defining CG_i and use geometric analysis of line segments to determine tangency conditions. There is a great deal of heuristic logic required to take pairs of points defining CG_i and determine if a gain exists that makes the corresponding line segment tangent to or touch the M-circle; then one must process the tangency/touching conditions for all pairs/line segments to arrive at the final gain values K_i. This is especially complicated for conditionally stable systems.

(ii) Given the data set $\{K_i(a_i)\}$, the SIDF inversion process is no less complicated. The symmetric piecewise-linear function used in these studies (similar to that depicted in Figure 6) is defined by four parameters: the slope at the origin, m_1, the input-variable break point δ, the discontinuity at that break point, D, and the slope beyond the break, m_2. The derivative of the SIDF for this non-linearity is a discontinuous function of these parameters, so the gradient of the SIDF to a given parameter may not exist and the fitting process for SIDF inversion may fail. For example, if all the amplitudes a_i are less than the initial guess for the break-point δ, then perturbing δ, D and m_2 will yield zero gradients for the fitting cost as a function of δ and the fitting procedure will terminate with the best linear fit in terms of m_1. This is a very obvious observation; we found that there were a number of other 'traps' that would defeat the conventional routine for the SIDF inversion process, and some of them were not so evident. In short, good initial guesses for these parameters were critical to the success of such an automated procedure.

The programming of the non-linearity synthesis rules profited a great deal from our efforts to implement the M-circle and SIDF inversion algorithms in our conventional CACE environment[18] — the code for CACE tools contained a large body of heuristic logic implemented in Fortran. The translation of this logic into a Delphi rule base was a straightforward task and the result is not particularly profound. For example, the rules to infer non-linearity parameter initial guesses prior to SIDF inversion involved the following.

1. Guess m_1 = average of the gains for the two smallest error amplitudes.
2. Guess δ = amplitude where the gain has the biggest % change.
3. Guess m_2 = average of the gains for the two largest error amplitudes.
4. The amount of 'dip' is determined by fitting a straight line through the two smallest and

two largest error amplitude points and finding the maximum deviation of $K_i(e_i)$ from that line.

5. Guess D = 'dip'.

Once these guesses are calculated, the conventional routine for SIDF inversion is run to obtain the best fit.

The third rule base in Figure 7 (the recommissioning rule base) was implemented in a very simplistic form. The new controller definition (parameters for PID gains and the parametrization of the piecewise-linear desensitizing non-linearity) is downloaded to the reprogrammable controller modelled in Simnon and is tested by subjecting the non-linear control system to a sequence of prespecified step inputs (the amplitudes of the steps ranging from 'small' to 'large' according to the physics of the non-linear plant) to determine if the transient response is satisfactory. Again the results were effectively identical to those reported in Reference 13, so the step responses are not repeated here.

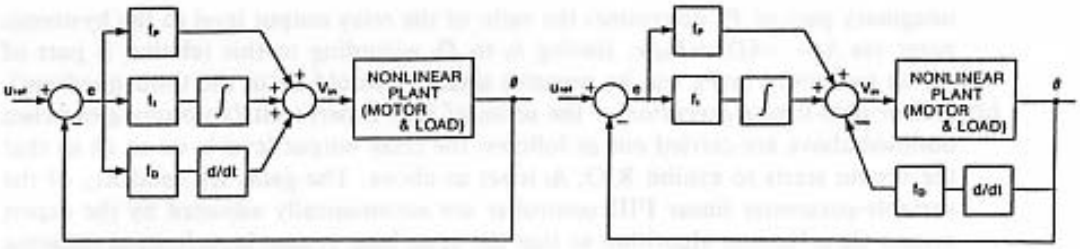
The outcome of this study was very encouraging, although we recognized that the final result implemented in our development environment was still a long way from a real-world implementable system. It demonstrated the power of a rule-based system to execute a very complicated and heuristic algorithm (the RBRTC implementation carried out an autotuning scenario in about 5 min CPU time on a VAX 785 compared with roughly half a day for the manual execution of the same effort that went into the study reported in Reference 13, and later experience has shown that the RBRTC system is very maintainable and extendable (see Section 5.2, for example).

5.2. *Rule-based non-linear autosynthesis*

After the above system was realized in simulation form (Figure 2) and validated, we invented a new, more general non-linear control autosynthesis methodology that approaches the generality of the synthesis method described in Reference 17. In addition, the new approach provides the means for autosynthesis with fewer experiments than the algorithm outlined in Section 5.1.

The main difference between the synthesis in References 16 and 17 is that the method in Reference 16 synthesizes a *single* non-linearity to precede an *arbitrary* linear controller to attempt to reduce the sensitivity of the open-loop system to input amplitude; in Reference 17 a *non-linear PID controller* is synthesized that has *three independent non-linearities* in the controller paths (proportional, integral, derivative), thus allowing more degrees of freedom in desensitizing the open-loop system and hence a less sensitive closed-loop system behaviour as demonstrated in the example in Reference 17. The benefit of added degrees of freedom can be appreciated by noting that this configuration allows the independent synthesis of desensitizing non-linearities for low frequency (via the 'I'-term), mid-frequency (the 'P'-term) and high frequency ('D').

The details of our second non-linear RBRTC autosynthesis approach differ from those in Reference 17 in several ways. The most noteworthy aspect of this new technique is that the structure of the controller is more realistic. The algorithm in Reference 17 required that the derivative feedback path be parallel to the proportional and integral paths as shown in Figure 8(a); in our more recent autosynthesis approach the rate term may be in the feedback path (Figure 8(b)). The latter configuration is highly preferable since it avoids overdriving the plant when the reference input changes abruptly. The second major difference is that the new autosynthesis approach requires far fewer experiments to obtain frequency domain plant



(a) Parallel PID paths (b) PID with Rate Feedback

Figure 8. Two non-linear PID controller configurations

model characterizations than would be required by a direct use of the algorithm in Reference 17.

The corresponding RBRTC system differs from that outlined in Section 5.1 and defined in Reference 13 primarily in replacing the autotuning rules with an *autosynthesizing rule base*. (The first three parts of the autosynthesis RB are also the same as those for autotuning.) In summary, this rule-based system works as follows.

(i) The *performance-monitoring rule base* (still unimplemented) determines when autosynthesis is required; if the behaviour is satisfactory, then it continues passive monitoring; else

(ii) the *autosynthesis rule base* is invoked to perform the following:

(a) *Plant identification* — reconfigure the control system as in Section 5.1 in preparation for producing relay-induced oscillations (RIOs) for plant model identification at a nominal amplitude; carry out RIO experiments to obtain several points on the non-linear plant 'Nyquist' locus $G(j\omega; a_0)$.

(b) *Linear PID controller synthesis* — synthesize the corresponding nominal linear PID controller using the above frequency response data (e.g. design a PID controller via Ziegler–Nichols tuning for the nominal amplitude); note that the linear controller type *must be PID* in this algorithm.

(c) *Linear gain-tuning set-up* — download the parameters for the new linear controller and set up a series of RIO experiments as in Section 5.1 in which a *variable-parameter linear PID controller* and plant will be excited by sinusoids of some frequency ω_{RIO} and various prespecified amplitudes $\{a_i\}$. The controller gains from the previous step are *initial guesses* for this procedure. The amplitude set $\{a_i\}$ ranges from 'small' to 'large' according to the designer's knowledge of the plant and is governed by the corresponding set of relay output levels via $D_i = \pi a_i/4$.[19] In the course of these experiments the two forward path linear controller gains $K_{P,i}$ and $K_{I,i}$ will be adjusted automatically (autotuned) for each amplitude so that the open-loop system input/output response $C_iG(j\omega_0; a_i)$ will pass through a specific point P_0 in the Nyquist plane at a specific frequency ω_0. (Generally ω_0 would be near the desired control system bandwidth and the point P_0 is a design specification selected by the control engineer; in classical control terms this point has obvious significance in relation to gain and phase margins or to touching a given M-circle. Assigning values to these design parameters as well as the amplitude set $\{a_i\}$ is part of set-up.) Note that the SIDF limit cycle condition is $C_iG(\omega_0; a_i) \triangleq R_{CG,i} + jI_{CG,i} = -1/N(a_i)$ and that for this non-linearity the imaginary part of $-1/N(a_i)$ is independent of the input amplitude (equation (2)). Referring to equation (2), this means that the

imaginary part of P_0 determines the ratio of the relay output level to the hysteresis point via $h_i = -4D_i \mathrm{Im} P_0/\pi$; slaving h_i to D_i according to this relation is part of set-up (generally $\mathrm{Im} P_0$ will be negative since P_0 would be in the third quadrant).

(d) *Linear gain-tuning execution* — the series of RIO experiments/autotuning exercises outlined above are carried out as follows: the relay output level is set to D_i so that the system starts to exhibit RIO; h_i is set as above. The gains $K_{P,i}$ and $K_{I,i}$ of the variable-parameter linear PID controller are automatically adjusted by the expert system via a Newton algorithm so that the open-loop system input/output response $C_i G(j\omega_0; a_i)$ will pass through the point P_0 in the Nyquist plane at a frequency ω_0. Specifically, $K_{I,i}$ is adjusted so that $R_{CG,i} = \mathrm{Re} P_0$ and $K_{P,i}$ is adjusted so that $\omega_{RIO} = \omega_0$. This set of experiments provides the gain/amplitude data $K_{I,i}(a_i)$ and $K_{P,i}(a_i)$. Note that K_D is left fixed in this algorithm.

(e) *Autosynthesis* — invoke the SIDF inversion procedure to synthesize the desensitizing controller non-linearities f_I and f_P that best fit the gain/amplitude data $K_{I,i}(a_i)$ and $K_{P,i}(a_i)$; reprogramme the controller (download the parameters that define the autosynthesized controller non-linearities).

(iii) The *recommissioning rule base* places the autosynthesized non-linear control system into service and returns to normal operation.

The complexity of the non-linear auto-synthesis rule base is roughly comparable to that of the autotuning case. There was a small amount of simplification achieved by replacing the M-circle algorithm rules with the gain-tuning logic. The main improvement compared with the rule base in Section 5.1 was that there are a smaller number of experiments to be performed in the non-linear synthesis step, so the second RBRTC system can execute its task more efficiently.

6. ILLUSTRATIVE EXAMPLE OF AUTOSYNTHESIZING NON-LINEAR CONTROL

As mentioned previously, primary emphasis was placed on the autosynthesis rule base in the above studies; the monitoring rule base has not been built and the recommissioning rules simply perform step response tests to determine the performance of the closed loop system after autosynthesis. The autosynthesis rule base was fully implemented, however, and performed well on the sample problems it was given.

The non-linear plant from References 13 and 17 was modelled in the Simnon-based simulation environment shown in Figure 2, along with a non-linear controller with parameters that can be specified by the expert system. The Supervisor rule base (Figure 7) was loaded into the Delphi inference engine, with the set goal of validating the response of the non-linear controller. By backward chaining, the first subgoal of autotuning or autosynthesis was then set and thus the corresponding secondary rule base was loaded (the *self-synthesis* rule base in Figure 7).

To achieve this sub-goal, the RBRTC set up and executed the series of simulation runs specified in one of the algorithmic 'recipes' in Section 4.3 or 4.4, i.e. the feedback loop was closed with a relay of specified height, RIO oscillations were excited and the resulting signals were processed to generate the gain/amplitude data for non-linearity synthesis. For each run the expert system specified the controller parameters, relay heights, simulation times, etc. and then told Simnon to simulate and stop. 'Sensors' were simulated by commanding Simnon to write out the required data; these data were read and translated into symbolic form by routines invoked by Delphi. The expert system also ran conventional software to carry out major numerical operations such as SIDF inversion. At the end of this process the controller non-

linearities were defined and downloaded to the Simnon model and the first subgoal was declared to be achieved; so the Supervisor sets a second subgoal; perform step response tests.

Another secondary rule base was invoked (a simplistic version of the *recommissioning* rule base in Figure 7) to set up and execute the specified step response simulations. The results are displayed to the user, who acknowledges the information. This establishes the fact that the response of the non-linear controller has been validated and thus terminates the simulated retuning process.

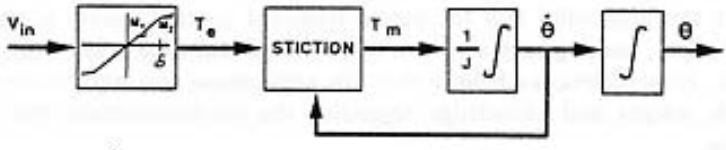
The above simulation study confirmed the effectiveness of the rule-based system. In brief:

(i) The plant was a simple but notoriously difficult model of a position control system with torque motor saturation and stiction (Figure 9).

(ii) The behaviour of the plant in combination with a *linear* controller is depicted in Figure 10, where it can be noted that the resulting feedback system exhibits 'sticking' for low-amplitude inputs and excessive overshoot for large reference input steps owing to integral wind-up.

(iii) The behaviour of the plant in combination with a *non-linear* controller *autosynthesized* by the methodology outlined above is depicted in Figure 11, where it can be noted that the resulting feedback system is much less sensitive to the amplitude of the reference step input.

Much more would have to be done to achieve a complete working RBRTC. The monitoring task is not difficult since a rule base for diagnosing the properties of real-time signals is



$$T_m = \begin{cases} T_e - f_v\dot{\theta} - f_c\,\mathrm{sign}(\dot{\theta}), & |T_e| > f_c \quad \text{or} \quad \dot{\theta} \neq 0.0 \\ 0.0, & |T_e| < f_c \quad \text{and} \quad \dot{\theta} = 0.0 \end{cases}$$

Figure 9. Plant for positioning servo problem

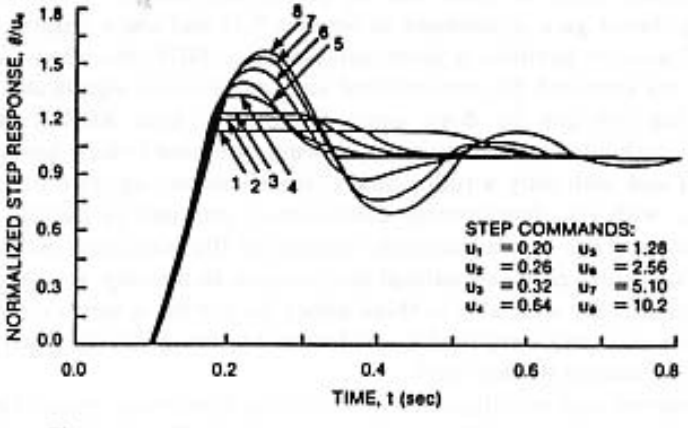


Figure 10. Positioning servo performance with linear control

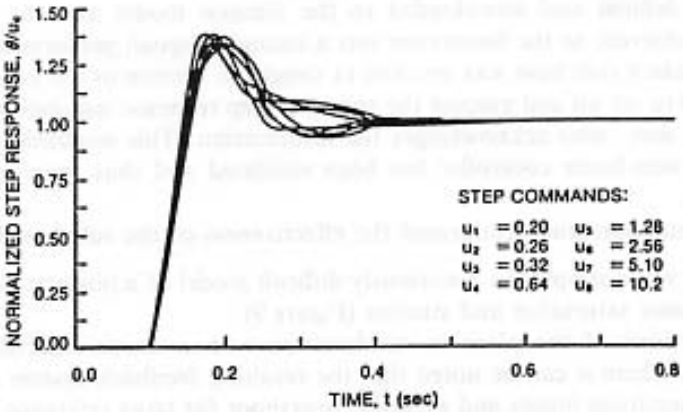


Figure 11. Positioning servo performance with non-linear control

straightforward and has been done in other contexts.[1] The self-synthesis rule base would have to be made more 'robust' for use in real field applications and would have to be extended with the ability to 'diagnose' the amplitude sensitivity of the plant if it is to deal with a variety of application areas. Validating the redesigned control system (testing its response and determining if specified performance measures are satisfactory) and recommissioning it properly is another major task which we have not considered. Also, we do not have a strategy for dealing with the possibility that the autosynthesized control system might not satisfy its requirements. Finally, making such a system work in *real* real-time may be difficult if the plant dynamics is fast. Nonetheless, we believe that this application and study have revealed a good deal of valuable insight and knowledge regarding the implementation and effectiveness of RBRTC systems.

7. CONCLUSIONS

A full implementation of a non-linear autosynthesizing control system as described in Sections 4 and 5 would clearly be a difficult task without the powerful environment provided by the rule-based systems approach described in Sections 2 and 3. There are many algorithms that need to be executed; some of these can be performed entirely within the expert system environment (e.g. linear gain adjustment in Section 5.2) and some require the expert system invoke other software to perform a given subtask (e.g. SIDF inversion, Section 5.1). Even those tasks that are executed by conventional software require significant heuristic decision making before the job can be done successfully. Our final RBRTC system was quite complicated even without a performance-monitoring rule base (which would not be trivial to programme well) and with only a rudimentary 'recommissioning' rule base.

Our experience with the development environment outlined in Section 3 has been very positive. Programming the logical/heuristic portion of the non-linear auto-synthesis control system in production rules gave us a natural environment to develop, test, extend and maintain this part of the system. We were able to think about the system in terms of high-level synthesis approaches, which was very beneficial as evidenced by the ability to generate new synthesis methods during the course of this work.

That such a system can be effective in controlling non-linear plants (assuming that the control objective is minimizing the sensitivity of the feedback system to input-amplitude

dependence without unnecessarily sacrificing performance) has also been shown. This study thus serves to demonstrate both the efficacy of the non-linear synthesis approach and the rule-based meta-control concept.

REFERENCES

1. Taylor, J. H., R. W. Gerhardt and E. C. Luce, 'Rule-based real-time control systems', *Proc. Conf. on Decision and Control*, Los Angeles, CA, December 1987, pp. 1923–1928.
2. Gerhardt, R. W. 'An expert system for real-time control, *M.Sc. Thesis*, RPI, Troy, NY, May 1986.
3. Åström, K. J. and J. J. Anton, 'Expert control', *Proc. IFAC 9th World Cong.*, Budapest, 1984, Vol VI, p. 240.
4. Åström, K. J., 'Adaptation, auto-tuning, and smart controls', *Proc. Chemical Process Control III*, Asilomar, CA, 1986, pp. 427–466.
5. Handelman, D. A. and R. F. Stengel, 'An architecture for real-time rule-based control', *Proc. American Control Conf.*, Minneapolis, MN, June 1987, pp. 1636–1642.
6. Antsaklis, P. J., K. M. Passino and S. J. Wang, 'Autonomous control systems: architecture and fundamental issues', *Proc. American Control Conf.*, Atlanta, GA, June 1988, pp. 602–607.
7. Tong, R., 'A control engineering review of fuzzy systems', *Automatica*, **13**, 559–570 (1977).
8. Barto, A. G., R. S. Sutton and C. W. Anderson, 'Neuron-like adaptive elements that can solve difficult learning control problems', *IEEE Trans. Syst., Man Cyber.*, **SMC-13**, 834–846 (1983).
9. Viswanadham, N., J. H. Taylor and E. C. Luce, 'A frequency-domain approach to failure detection and isolation with application to the GE-21 turbine engine control system', *Control — Theory and Advanced Technology (C-TAT)* **3**, 45–72 (1987).
10. Elmqvist, H., 'SIMNON — an interactive simulation program for non-linear systems', *Proc. Simulation 77*, Montreux, 1977.
11. Taylor, J. H. and D. K. Frederick, 'An expert system architecture for computer-aided control engineering', *Proc. IEEE*, **72**, 1975–1805 (1984).
12. James, J. R., J. H. Taylor and D. K. Frederick, 'An expert system architecture for coping with complexity in computer-aided control engineering', *Proc. Third IFAC Symp. on Computer-aided Design in Control and Engineering Systems*, Lyngby, August 1985, pp. 47–53.
13. Taylor, J. H., and K. J. Åström, 'A nonlinear PID autotuning algorithm', *Proc. American Control Conf.*, Seattle, WA, June 1986. pp. 2118–2123.
14. Åström and T. Hägglund, 'Automatic tuning of simple regulators', *Proc. IFAC 9th World Congr.*, Budapest, 1984, Vol. III, p. 267.
15. Hägglund, T. and K. J. Åström, 'A new method for design of PID regulators', *Report TFRT-7273*, Department of Automatic Control, Lund Institute of Technology, 1984.
16. Taylor, J. H. and K. L. Strobel, 'Applications of a nonlinear controller design approach based on quasilinear system models', *Proc. American Control Conf.*, San Diego, CA, 1984, pp. 817–824.
17. Taylor, J. H. and K. L. Strobel, 'Nonlinear compensator synthesis via sinusoidal-input describing functions', *Proc. American Control Conf.*, Boston, MA, June 1985, pp. 1242–1247.
18. Atherton, D. P., *Nonlinear Control Engineering*, Van Nostrand Reinhold, London and New York, full edition 1975, student edition 1982.
19. Taylor, J. H., 'A systematic nonlinear controller design method based on quasilinear system models', *Proc. American Control Conf.*, San Francisco, CA, 1983, pp. 141–145.
20. Taylor, J. H., 'Computer-aided control engineering environment for nonlinear systems', *Proc. Third IFAC Symp. on Computer-aided Design in Control and Engineering Systems*, Lyngby, August 1985, pp. 38–43.