LUND UNIVERSITY

Computer Aided Control Engineering CACE

Proceedings of a Full-Day Seminar in Stockholm, March 25, 1987

Mattsson, Sven-Erik

1987

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

Link to publication

Citation for published version (APA):
Mattsson, S.-E. (Ed.) (1987). *Computer Aided Control Engineering CACE: Proceedings of a Full-Day Seminar in Stockholm, March 25, 1987*. (Technical Reports TFRT-7359). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

CODEN: LUTFD2/(TFRT-7359)/1–140/(1987)

Computer Aided Control Engineering CACE

Proceedings of a full-day seminar
Stockholm, March 25, 1987

Sven Erik Mattsson (Ed)

Department of Automatic Control
Lund Institute of Technology
May 1987

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

Document name
Report

Date of issue
May 1987

Document Number
CODEN:LUTFD2/(TFRT-7359)/1-140/(1987)

Author(s)
Sven Erik Mattsson (Ed)

Supervisor

Sponsoring organisation
The Swedish Board of Technical Development

Title and subtitle
Computer Aided Engineering, CACE
Proceedings of a full-day seminar, Stockholm, March 25, 1987

Abstract

This report contains the proceedings of a full-day seminar held in Stockholm on March 25, 1987. The seminar was arranged by the Swedish National Board for Technical Development, STU, Stockholm and the Department of Automatic Control, Lund Institute of Technology, Lund. The purpose was to present results and experiences from the STU supported research program Computer Aided Control Engineering, CACE. About 50 persons from university and industry attended the seminar.

Key words
Computer Aided Control Engineering

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English and Swedish

Number of pages
140

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

Computer Aided Control Engineering

Datorstödd konstruktion av reglersystem

Seminarium 25 mars 1987
Stockholm

Edited by Sven Erik Mattsson
Dept. of Automatic Control, Lund Institute of Technology

STYRELSEN FOR TEKNISK UTVECKLING
SWEDISH BOARD FOR TECHNICAL DEVELOPMENT

Program and Table of Contents

March 25, 1987

9.00 **Introduction**

9.05 **Computer Aided Control Design – A Perspective**

K.J. Åström, Dept of Automatic Control, LTH: Viewgraphs 7

Computer Aided Modeling, Analysis and Design of Control Systems – A Perspective, Control Systems Magazine, May 1983 19

9.50 **Experiences of using Simnon and Idpac**

Jan-Erik Gustafsson and Thomas Östman, STFI: Viewgraphs 33

Claes Källström, SSPA: The use of hybrid model tests and computer simulations for offshore installations, IFIP/IFAC Symposium ASSOPO'85, Trondheim, Norway, 1985 37

Anders Weibull, Optimus Control: Viewgraphs 49

10.45 **Break**

11.00 **The CACE Project – An Overview**

S.E. Mattsson, Dept of Automatic Control, LTH: Viewgraphs 53

Published Papers, Conference Contributions and Reports 59

International CACE Software Efforts, (S.E. Mattsson and K.J. Åström) 63

11.30 **Hibliz – A Simulator Using Hierarchical Block Diagrams**

S.E. Mattsson, Dept of Automatic Control, LTH.

A Simulator for Dynamical Systems Using Graphics and Equations for Modelling, (Elmqvist and Mattsson), IEEE Control Systems Society Third Symposium on Computer-Aided Control System Design (CACSD), Arlington, VA, September 24–26, 1986 77

System Representations, (Åström and Kreutzer), IEEE Control Systems Society Third Symposium on Computer-Aided Control System Design (CACSD), Arlington, VA, September 24–26, 1986 83

12.00 **Lunch**

13.15 **Expert System Interfaces**

J.E. Larsson and *P. Persson*, Dept of Automatic Control, LTH: Viewgraphs 89

An Intelligent Help System for Idpac 95

14.00 **Symbolic Formula Manipulation**

Ulf Holmberg, Dept of Automatic Control, LTH: Viewgraphs 101

Combination of Symbolic Manipulation and Numerics, (Holmberg, Lilja and Mattsson), TFRT-3186 107

14.30 **Expert Control**

K-E. Årzén, Dept of Automatic Control, LTH: Viewgraphs 125

Knowledge Based Controllers 133

15.00 **Coffee**

15.30 **Related STU Projects**

Arne Otteblad, STU: Viewgraphs 135

16.00 **Continuation of the CACE Project and Discussion**

Karl Johan Åström, Dept of Automatic Control, LTH: Viewgraphs 139

17.00 **Closing**

Computer Aided Control Engineering

K. J. Åström

Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

1. Introduction
2. Interaction Principles
3. An Example
4. How It Developed
5. The Future
6. Conclusions

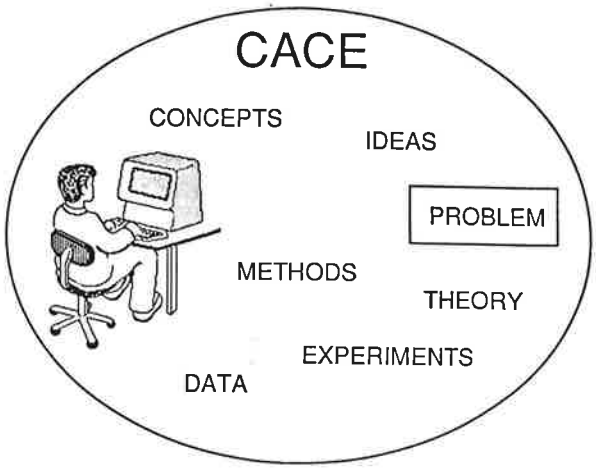
Introduction

Background

Many methods are conceptually simple but their details may be messy

Solution

Combine man's intuition with the computers Numeric, symbolic and reasoning capability



What?

Control engineering tasks

Modeling
Identification
Analysis
Simulation
Design
Implementation
Documentation
Validation
Evaluation

Why?

- Increased engineering productivity
  Facts: X Manmonth → X Manweeks
  No intermediaries
- Better solutions
- Improved teaching
- Good way to package theory
- Good Problems
  Depth / Breadth
  Theory / Applications

How?

The control engineers workbench

Interaction Principles

Question and answer
Command dialog
Dynamic menus
Natural language
Knowledge based interfaces

Command Dialog

General form:

```
COMMAND LARG1 LAG2 < RARG1 RARG2
```

Examples:

```
MATOP S < A*B + C
INSI U 100
   >PRBS 4 7
   >EXIT
SIMU Y < SYST U
OPTFB L CLSYS < LOSS SYS
SYSOPT CLSYS < SYS1 SYS2 FB
ML PAR < DATA ORDER
```

Advantage:

User keeps the initiative

Disadvantage:

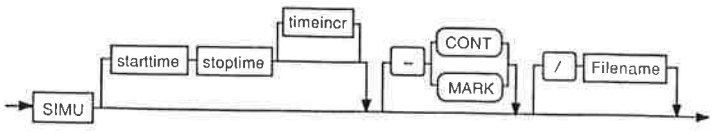
User must know

Command Dialog

Commands should be

- natural
- powerful
- self-explanatory
- predictable (consistent)
- short
- flexible

Command extensions
Short forms
Default parameters



Example

```
SIMU 0 100
SIMU
```

Synpac Example

SYSTEM FILE:

DYNAMICS	$dx/dt = Ax + Bu + v$
A,B,C,D,	$y = Cx + Du + e$
COVARIANCES	$R1 = cov(v)$
R1,R12,R2	$R12 = cov(v, e)$
LOSSFUNCTION	$R2 = cov(e)$
Q1,Q12,Q2	

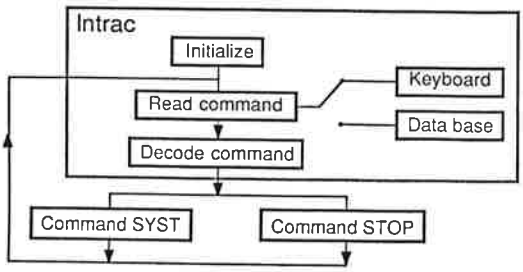
SAMPLING PERIOD H

$$J = \int_0^\infty [x^t Q1x + 2x^t Q12u + u^t Q2u]dt$$

```
MACRO DESIGN ALPHA
  ALTER Q1 3 3 ALPHA
  FOR H + 0.5 TO 5 STEP 0.5
    SAMP DSYS < CSYS H
    TRANS Q DSYS < CSYS H
    TRANS R DSYS < CSYS H
    OPTFB L < DSYS
    KALFI K < DSYS
    CONNECT CLSYS < DSYS K L
    SIMU Y X < CLSYS UREF
    PLOT X(1) X(7) X(8) XE(1) U
  NEXT H
END {MACRO}

EDIT SYSTEM FILE
INPUT UREF < STEP
DESIGN 3
DESIGN 8
```

Command Language Interpreter



Command procedures (Macro)
Conrol structures
Help systems

Interpreters (Lisp, Basic, Apl), incremental compilers and command language interpreters (VMS, Unix) are written in similar ways. Compare the "read-eval-print loop" in Lisp.

High Level Problem Solving Languages

- ❄ Vocabulary
  words, data objects
- ❄ Composition rules
  operators, grammar, syntax
- ❄ Meaning
  semantics

A language should be:

- ❄ Rich to solve many problems
- ❄ Simple and easy to learn
- ❄ Extendable

A CAD system is simply a language interpretor

More than a means for instructing the computer to perform tasks

Should serve as a framework within which we organize our ideas

- Primitive data types and operations
- Methods for combining them into compounds/objects
- Means of abstraction, by which compound objects can be named and manipulated as units

SIMNON

Simulation of Mixed Continuous and Discrete Time Systems

- Data structures
- Command language

Documentation aids

```
SWITCH SPY/ECHO/LOG ON
WRITE
```

Data Structures

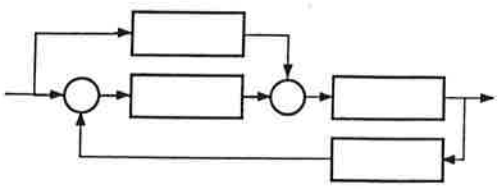
- Continuous system

$$\frac{dx}{dt} = f(x,u,t)$$
$$y = g(x,u,t)$$

- Discrete system

$$x(t_{k+1}) = f(x(t_k),u(t_k), t_k)$$
$$x(t_k) = f(x(t_k),u(t_k), t_k)$$

- Connecting system



SIMNON

Command Language

Utilities

EDIT
GET
LIST
PRINT
SAVE
STOP

Graphic Output

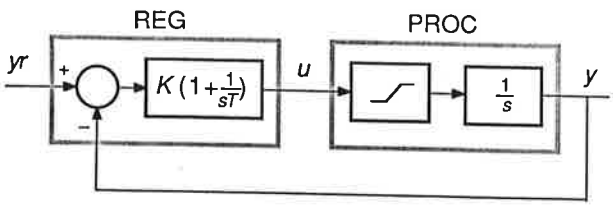
AREA
ASHOW
AXES
HCOPY
SHOW
SPLIT
TEXT

Simulation Commands

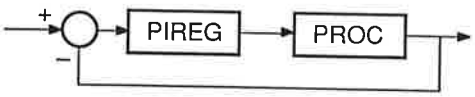
ALGOR
DISP
ERROR
INIT
PAR
PLOT
SIMU
STORE
SYST

EXAMPLE

PI Control with Antiwindup




```
CONNECTING SYSTEM conn
yr[reg] = 1
y[reg] = y[proc]
u[proc] = u[reg]
END
```




```
CONTINUOUS SYSTEM proc
INPUT u
INPUT y
.
.
.
END

DISCRETE SYSTEM pireg
INPUT yr y
OUTPUT u
.
.
.
END

CONNECTING SYSTEM conn
yr[pireg] = 1
y[pireg] = y[proc]
u[proc] = u[pireg]
END
```

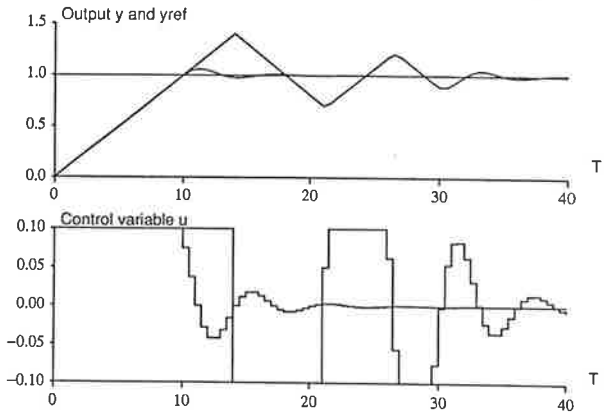
Discrete PI Regulator

```
DISCRETE SYSTEM pireg
INPUT yr y
OUTPUT u
STATE i
NEW ni
TIME t
TSAMP ts
```

} Declarations

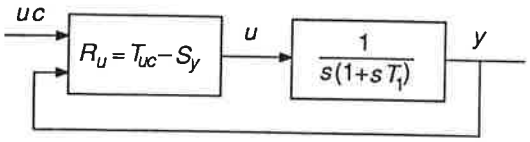
```
e = yr - y
v = k*e + i
u = IF v < ulow THEN ulow
    ELSE IF v < uhigh THEN v
    ELSE uhigh
ni = i + k*e*h/ti + h/t0*(u - v)
ts = t + h
h:0.2
ti:1
t0:1
ulow:-1
uhigh:1
END
```

} Parameter assignments




```
MACRO fig9
"Generates Fig. 9
SYST integr pireg conn
STORE yr y[proc] upr
SIMU 0 40/wup
PAR ulow:-0.1
PAR uhigh:0.1
SIMU /nowup
SPLIT 2 1
ASHOW y/wup
SHOW y/nowup
ASHOW upr/wup
SHOW upr/nowup
END
```

Influence of Sampling Rate in Pole Placement Control




```
CONNECTING SYSTEM conn
uc[reg] = 1
u[proc] = u[reg]
y[reg] = u[proc]
END
```

It is very hard to find out how the response is influenced by h by analysis!

```
DISCRETE SYSTEM polp
INPUT yc y
⋮
TSAMP ts
"Compute desired discrete char pol
p1 = -2*cos(w*h*sqrt(1-z^2))
p2 = exp(-2*z*w*h)
"Sampling continuous model 1/s(s+1)
a2 = exp(-h/t1)
a1 = -(1+az)
b1 = h-t1*(1-az)
b2 = t1-az*(t1+h)
"Solution of Diophantine eq. AR+BS=PT

"Output
u = t0*uc-s0*y-s1*yold-r1*uold
"Dynamics
nyold = y
nuold = u
ts = t+h
"Parameter assignments
⋮
END
```

History

Analog computing

Digital simulators

Dynasar DAS	1962
CSMP	1965
CSSL	**1967**
ACSL	1975
Simnon	1973

CACE

UMIST	1970
Lund	1972
Cambridge	1972
Etc.	

Proliferation

Research at many places	
Matlab	1981
Commercial packages	1982
Workstations CAD	

MatLab and Derivatives

Cleve Moler CS Univ. New Mexico 1981

Interactive matrix language

Expression parser

User defined functions

exp(A), sin(A)

CTRL-C	Systems Control Technology
Matrix	Integrated Systems
PC-Matlab	The Math Works
Impact	ETH
EAGLES & M	LLNL
Blaise	INRIA

Proliferation

IFAC Symposia

Zürich 1979
West Lafayette 1982
Lyngby 1985
Working Group

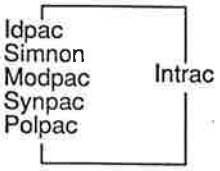
GE-RPI-NSF Workshop 1981

IEEE Symposia

Berkeley CA 1982
Boston MA 1983
Santa Barbara 1985
Arlington VA 1986
Technical Committe on CACSD

Review of Lund Experience

- Software libraries 1970
  Early experiments
- Synpac 0 1972
- Simnon 0 1973
- STU-project 1973–1980



- Use, maintenance, distribution
- Why did we stop 1980 ?

Lund Packages

Simnon Interactive simulation language for nonlinear continuous and discrete time systems with facilities for optimization and use of experimental data.

Idpac Interactive language for identification of linear systems using parametric and non-parametric (covariance spectra) methods.

Synpac State space oriented design package for linear systems which includes LQG, pole placement and robust designs for discrete and continuous systems.

Modpac Analysis and transformations of models.

Polpac Polynomial oriented analysis and design package.

Lispid System for off-line identification of nonlinear continuous time systems.

Dymola A new simulator concept with hierarchically structures and model types.

Program Sizes

	Commands	code k lines	size k bytes
Intrac	**17**	7	90
Idpac	**34**	37	470
Simnon	**22**	25	360
Modpac	**41**	41	570
Synpac	**47**	43	630
Lispid	–	9	260

Intrac

One module for interaction common to all packages

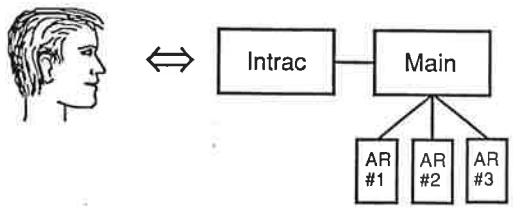
Command oriented

Initiative stays with user but may be transfered to computer when needed

Features

Arguments
Local and global variables
Input-output, numbers and graphs
Control of program flow
Simplified dialog
Macro facility

Structure



Experiences of Use

At our own department

Elementary courses
Advanced courses
Projects
Research
Book writing — Åström/Wittenmark, "Computer Controlled Systems –Theory and Design" Prentice Hall 1984
Model library

At other departments

Mathematics
Statistics
Biology
Chemistry

In industry

Portability
Hardware requirements
Educational requirements
Special features

Teaching Experience

- ❄ Come closer to reality with modest effort
- ❄ Model library
- ❄ Reproducable results
  - Easy to document
  - Easy to repeat
  - Easy to modify
- ❄ We have barely scratched the surface

Industrial Experience

Saab	Control laws and simulation JA37, **JAS**
STFI	Analysis and simulation of processes in pulp and paper
SSPA	Analysis and simulation of ships and platforms
GE	Federated systems
Sydkraft	Data analysis from power stations, Compact simulator for Barsebäck, Wind power plant
Asea	Robotics, Relay, Automation, Traction
Bofors	Design and simulation of fire control systems
SattControl	Simulation of auto-tuners

Experience of Design

- ❄ Unified user interface – Intrac
- ❄ Extensible command language
- ❄ High level language
- ❄ User feedback

The Lund Tradition

INTRAC may be viewed as a package to make FORTRAN routines interactive

GE Federated System

Rattles Delight, Delight–MIMO
U.C. Berkeley, Imperial College

Delight Mary-Lin
U.C. Maryland

Perspective

1970	1987
Numerics	
Poor libraries Wilkinson & Reinsch (1971)	Good libraries Eispac, Linpac, ODE Differential Algebraic Equations
Symbolic Calculation	
Very limited	Macsyma
Hardware	
64 kbyte RAM 0.5 Mbyte Disc Teletype Storage oscilloscope	4 Mbyte RAM 70 Mbyte Disc 1 MPEL Color Graphics
Interaction Models	
Terminal oriented Q & A Command languages Lisp, Apl, Basic	Personal computing workstations Dynamic menus Xerox Parc, Macintosh Spreadsheets Natural language Knowledge based
Software	
Fortran Basic	Lisp Ada Prolog
Experiences	
None	Plenty

Current Projects

Commercial Development
Matlab derivatives

England SERC 28 Msek
UMIST, Cambridge, Kingston
Salford, Swansea, Imperial College

France
INRIA, Grenoble SIRENA

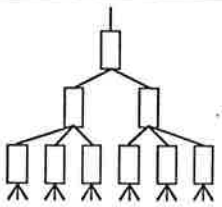
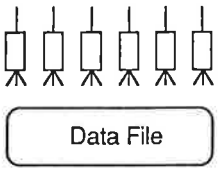
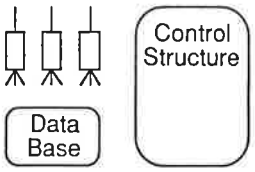
Benelux
SYCOT WGS

ETH / U Arizona
Impact

USA

LLNL	EAGLES & M
U Tennessee	CASCADE
UC Berkeley	DELIGHT
U Maryland	Delight Marylin
RPI	
US SantaBarbara	

Three Phases of CAE

Menu driven dialog	Command dialog	Expert guided dialog
		
Operators + fixed logic Easy to use for the problems it was designed for Straight jacket for other	Operators help command procedures Flexible but more difficult to use	Operators + flexible data driven logic Flexible and easy to use Learning facility

Why do we need more than matrices?

- Other datatypes needed
- Inconvenient to represent systems as

  A, B, C, D

  or

  $$\begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

  TAG
- Matrix transfer functions
- Hierarchical systems
- Nonlinear systems

Why start again?

- Help to realize the full potential of the new emerging hardware.
- New ideas are essential. They will not come from the commercial scene.
- Still a long a way to go towards the CE's workbench.
- Defining high level problem oriented languages is a good way to systematize control engineering methods and practice.

Computer Aided Control Engineering (CACE)

Research project 1985–1989
Aimed at developing a software environment for Computer Aided Control Engineering

Pilot Projects

1. Graphics based man-machine interaction
2. Expert system interfaces
3. Formula manipulation and numerics
4. High-level problem solving languages
5. Implementation languages
6. Expert control

Conclusion

Good research projects

- Unify control theory
- Realistic models
- Nonlinearities
- Complete design cycle

Student interest

New premises

- Knowledge
- Experience
- Computer hardware
- Computer software

Increased productivity

Computer Aided Modeling, Analysis and Design of Control Systems–A Perspective

K. J. Astrom

Department of Automatic Control, Lund Institute of Technology, S-220 07 Lund, Sweden

ABSTRACT: The paper summarizes experiences of development and use of interactive software for computer aided design of control systems. Different principles for interaction with users having wide ranges of experiences and knowledge are discussed. A comprehensive set of packages for modeling, identification, analysis, simulation, and design are described. Problems associated with structuring, portability, maintainability and extensibility are discussed. Experiences from development and use of the packages in teaching and industrial environments are discussed. Some views on future development of CAD for control systems are also given.

Introduction

Thirty years ago pencil, paper, slide rules and analog computers were the major tools for analysis and synthesis of control systems. The methods and the tools were so simple that an engineer could master both problems and tools. Many new methods for analysis and design of control systems have emerged during the last 30 years. These methods differ from the classical techniques. They are more sophisticated analytically and their use requires extensive calculations. An extensive subroutine library is required to apply these methods to a practical problem. Even if such a library is available, it is a major effort to write the software necessary to solve a particular problem. This means that modern control theory is costly to use. Another drawback is that the problem solver interacts with his tools (the computer) via intermediaries (programmers). This easily leads to confusion and mistakes. The intensive interaction between problem formulation and solution is also lost.

Received August 30, 1982; revised January 27, 1983. Accepted in revised form by Editor M. Jamshidi.

Expanded version of papers presented at the 18th and 20th IEEE Conferences on Design and Control, San Diego, CA, and the GE-RPI workshop, Schenectady, NY.

Based on experience from industrial application of modern control theory in the early sixties, it was clear to me that modern control theory could be used very successfully in a research laboratory or at a university. It was, however, equally clear that the methods would not be widely used in normal engineering practice unless the proper tools were developed. A number of projects were therefore carried out in order to explore the possibilities of developing the proper tools for using control theory cost effectively. This paper summarizes results and experiences from these projects.

The projects were based on the idea of combining an engineer's intuition and overview with digital computing power. The approach included development of design techniques and design of man-machine interfaces, for interactive use of the computer. Graphics were important for rapid man-machine communication.

This paper is organized as follows. A brief overview of the projects is given in Section 2. Interaction principles are discussed in Section 3. The comprehensive set of program packages which is one result of the projects is described in Section 4. Some special problems associated with large systems are discussed in Section 5. Experiences from use of the packages in university and industrial environments are presented in Section 6. Sections 7 and 8 give suggestions for future work and conclusions.

The Projects

A brief overview of the projects which were carried out are given in this section.

Goals

The objectives of the projects were to make advanced methods for modeling, analysis and design of control systems easily accessible to engineers, researchers and students, and to explore the potentials of interactive computing for control system design.

Background

When the projects were initiated around 1970 we had extensive experience in analog simulation, programming in FORTRAN, BASIC, and APL. There was consensus about the power of digital computation and the superiority of the man-machine interaction in analog simulation. We were familiar with the ease of debugging and running programs in an interactive implementation like APL. But we were also aware of the limited portability of such programs and of the difficulties of extending such systems.

Stepwise refinement

The software was developed in close interaction with the users. A system outline was sketched, and the ideas were discussed in seminars. A system of moderate size was implemented and tested by several users. The system was then modified. In the initial phases we were also quite willing to scrap a system and start all over again. As the projects progressed we acquired a much better feel for what could be done and how it should be done. It also became clear that a fairly comprehensive package was necessary to evaluate the ideas. Such packages were also developed. They went through many revisions to improve portability, modularity and efficiency.

Constraints

What can be done with interactive computing depends much on the available hardware. Since the hardware has undergone revolutionary development over the past ten years, it is useful to describe what was available in the projects. When the activity was started in 1971 we had access to a DEC PDP 15 with 32 kbytes of core memory, a 256 kbyte disk and a storage oscilloscope. After a few years the activity was moved to large mainframe computers. We are currently using a DEC Vax-11/780 with 2 Mbyte of fast memory and a 300 Mbyte disc for most of the work.

Our sponsoring agency (STU) also introduced constraints by insisting that the programs should be portable and useful to industry. One way to achieve this was to use standard FORTRAN.

Results

The projects have resulted in a comprehensive set of program packages for model-

0272-1708/83/0500-0004$1.00©1983 IEEE

ing, identification, analysis, simulation and design of control systems. We have several years experience in using these packages in different environments. Ideas on the use of graphics and interactive computing in future systems have also been developed. An overview of the results are given in the next sections.

Interaction Principles

When designing a system for man-machine interaction it is important to realize that there is a wide range of users, from *novices* to *experts*, with different abilities and demands. For a novice who needs a lot of guidance it is natural to have a system where the computer has the initiative and the user is gently led towards a solution of his problem. For an expert user it is much better to have a system where the user keeps the initiative and where he gets advice and help on request only. Attempts of guidance and control by the computer can lead to frustration and inefficiency. It is highly desirable to design a system so that it will accommodate a wide range of users, making it more universal. It also makes it possible to gradually shift the initiative from the computer to the user as he becomes more proficient.

To obtain an efficient man-machine interface it is desirable to have hardware with a high communication rate and a communication language with a good expression power. When our projects were started we were limited to a teletype and a storage oscilloscope. There were also limited experiences of design of man-machine interfaces. The predominant approach was a question-and-answer dialog. (See Rosenbrock, 1974.)

In our projects it was discovered at an early stage that the simple question-and-answer dialog was too rigid and very frustrating for an experienced user. The main disadvantage is that the computer is in command of the work rather than the user. This was even more pronounced because of the slow input-output device (teletype) which was used initially.

Our primary design goal was to develop tools for the expert. A secondary goal was to make the tools useful for a novice also. To make sure that the initiative would remain with the user it was decided to make the interaction command oriented. This was also inspired by experiences from programming in APL. Use of a command dialog also had the unexpected effect that it was possible to create new user defined commands easily. It was thus possible to use the packages in ways which were not anticipated when they were designed. The decision to use commands instead of a question and answer dialog thus had far reaching consequences. A more detailed discussion of the different types of dialogs and of our experiences of them is given in Wieslander (1979). Today there is a wide range of experiences of designing man-machine interfaces in many different fields. Our own conclusions agree well with those found in Newman and Sproull (1979) and Foley and van Dam (1981), although their conclusions are based on different hardware.

Examples of commands

The structure of the commands we introduced will now be described. The general form of a command is

```
NAME LARG1 LARG2 ···
←RARG1 RARG2 ···
```

A command has a name. It may also have left arguments and right arguments. The arguments may be numbers or names of objects in a data base. In our packages the objects are implemented as files because this is a simple way to deal with objects having different types. A few examples of commands are given to further illustrate the notion of a command.
The command

```
MATOPS S ← A * B + C
```

simply performs the matrix operation expressed to the right of the arrow.
The command

```
POLOP S ← A * B + C
```

performs the same operation on polynomials.
The command

```
INSI U 100
  >PRBS 4 7
  >EXIT
```

generates an input signal of length 100 called U. The command has options to generate several input signals. The options are selected by additional subcommands. PRBS is a *subcommand* which selects a PRBS signal. The optional arguments 4 and 7 indicate that the PRBS signal should change, at most, every fourth sampling period and that its period should be 2^7-1. The subcommand EXIT denotes the end of the subcommands.
The command

```
DETER Y ← SYST U
```

generates the response of the linear system called SYST to the input signal U.
The command

```
MLPAR ← DAT N
```

fits an ARMAX model of order N to the data in the file called DAT and stores the parameters in a file called PAR.
The command

```
OPTFB L CLSYS ← LOSS SYS
```

computes the optimal feedback gain L and the corresponding closed loop system CLSYS for the system SYS and the loss function LOSS.

Short form commands and default values

In a command dialog it is highly desirable to have simple commands. This is in conflict with the requirement that commands should be explicit and that it may sometimes be desirable to have variants of the commands. These opposite requirements may be resolved by allowing short forms of the commands. The standard form for the simulation command is SIMU. If no other command starts with the letter S it is, however, sufficient to type S alone. It may also be useful to have a simple way of renaming the commands. We have experimented with short form commands and renaming me-

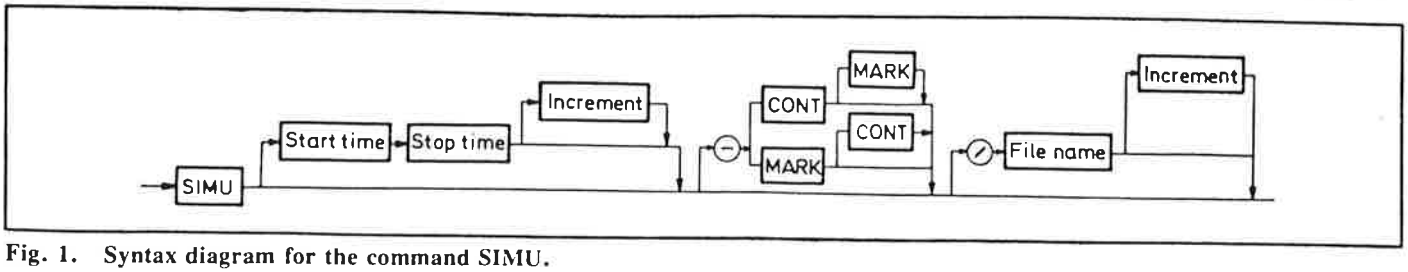


Fig. 1. Syntax diagram for the command SIMU.

Appendix A-INTRAC Commands

1. Input and output

READ	— Read string or variable from keyboard.
SWITCH	— Utility command.
WRITE	— Write string or variable on terminal.

2. Assignment

DEFAULT	— Assign default values.
FREE	— Release assigned global variables.
LET	— Assignment of variables and global parameters.
STOP	— Stop execution and return to OS.

3. Control of program flow

FOR..TO	— Loop.
NEXT V	
LABEL L	— Declaration of label.
GOTO L	— Transfer control.
IF..GOTO	— Transfer control.

4. Macro

END	— End of macro definition.
FORMAL	— Declaration of formal arguments.
MACRO	— Macro definition.
RESUME	— Resume execution of macro.
SUSPEND	— Suspend execution of macro.

chanisms. These functions are, however, not implemented in our standard packages.

A similar mechanism may be used for commands which use arguments by introducing a default mechanism so that previous values of the arguments are used unless new values are specified explicitly. The concept is illustrated by an example.

The syntax diagram for the command SIMU is shown in Fig. 1. The diagram implies that any form of the command which is obtained by traversing the graph in the directions of the arrows is allowed. For example, the command

SIMU 0 100

simulates a system from time 0 to time 100. If we want to repeat the simulation a second time with different parameters, it suffices to write

SIMU

The arguments 0 and 100 are then taken as the previously used values.

It follows from Fig. 1 that start and stop times and the initial time increment may be specified. It is also possible to mark curves by the argument MARK. A simulation may also be continued by using the end conditions of a previous simulation as initial values. This is done by the command extension CONT. The results of a simulation may also be stored in a file.

Macros

The commands are normally read from a terminal in a command driven system. It is, however, useful to have the option of reading a sequence of commands from a file in storage instead. Since this is analogous to a macro facility in an ordinary programming language, the same nomenclature is adopted. (See Wegner, 1968.) The construction

```
MACRO NAME
  Command 1
  Command 2
  Command3
END
```

thus indicates that the commands 1, 2 and 3 are not executed but stored in memory. The command sequence is then activated simply by typing NAME.

Macros are convenient for simplification of a dialog. Command sequences that are commonly used may be defined as macros. A simple macro call will then activate a whole sequence of commands. The macro facility is also useful in order to generate new commands. Macros may also be used to rename commands. This is useful in order to tailor a system to the needs of a particular user.

The usefulness of macros may be extended considerably by introducing commands to control the program flow in a macro, facilities for handling local and global variables and by allowing macros to have arguments. By having commands for reading the keyboard and for writing on the terminal, it is also possible to implement menu driven dialogs using macros.

An interactive CAD program based on a command dialog with a macro-facility may be viewed as an *extendable high level problem solving language*.

Error checking

It is important in interactive systems to have tests for avoiding errors. It is thus useful to check data types and to test problems for consistency whenever possible.

Implementation

It is straightforward to implement a command driven interactive program. The structure used in all packages is shown in Fig. 2. The main loop reads a command, decodes it and performs the required actions. All parts of Fig. 2 except the action routines are implemented as a package of subroutines called INTRAC. These subroutines perform command decoding, file handling and plot-

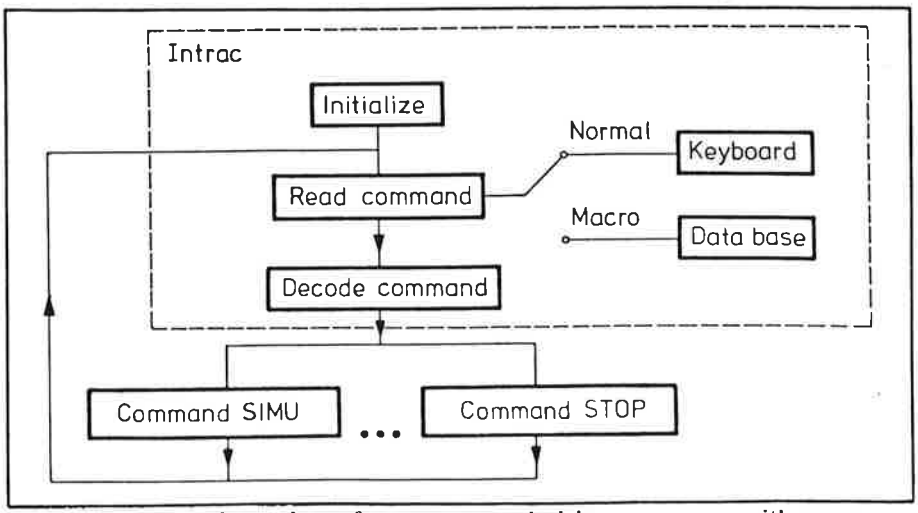


Fig. 2. Skeleton flow chart for a command driven program with a macro facility.

ting. INTRAC also contains the macro facility. Macros may have formal arguments, local and global variables. They permit conditional and repeated execution of commands as well as nested use of macros. There are read and write commands, which can be used to implement menu dialogs. It is possible to mix command mode and question mode, since the execution of a macro may be suspended and resumed later. A description of INTRAC is given in Wieslander and Elmqvist (1978) and Wieslander (1980a) The commands available in INTRAC are listed in Appendix A.

To build a package using INTRAC it is necessary to write the action routines, i.e. the subroutines that perform the desired tasks. The commands are then entered in the command table of the command decoder. It is also easy to add a command to a package, to move commands between packages and to create special purpose packages. INTRAC may thus be viewed as a tool for converting a collection of FORTRAN subroutines into an interactive package. INTRAC has also been used to implement other packages by other groups.

The structure with a common user interface for all packages is advantageous for the user because the interaction and the macro commands are the same in all packages. This simplifies learning and use of the packages.

How to choose commands

The selection of commands is one of the major issues when designing a CAD package. The commands determine how useful a package is and how easy it is to learn. It is important that commands are *complete* in the sense that they allow use of a wide range of techniques in an area. Otherwise the designer will only try those approaches for which commands are available. Commands should also have a considerable *expression power* so that a control system designer can do what he wants with a few commands. The commands should also reflect the *natural concepts* from a theoretical point of view. This would make it easy for a user well versed in control theory to use a package. The commands should also be *few and simple* so that they are easy to learn and remember. This is, of course, in conflict with requirements on completeness and expression power. Selection of commands is thus a good exercise in engineering design.

Based on experiences from our projects we have arrived at some design principles. A set of basic commands which correspond to the elements of the theory and which allow coverage of a certain problem area are first determined. Simplifications and extensions are then generated using the macro facility.

Program Packages

The initial experiments indicated that interactive program packages could be powerful problem solving tools. The necessity of considering a wide range of problems when developing the tools was also apparent. If this is not done it is easy to arrive at specialized solutions which are difficult to generalize and extend. To work with programs of reasonable size a family of interactive program packages for modeling, identification, simulation, analysis and design of control systems were developed. The packages are all based on the common user interface INTRAC which was discus-

Appendix B–IDPAC Commands

1. Utilities

CONV	— Conversion of data to internal standard format.
DELET	— Delete a file.
EDIT	— Edit system description.
FHEAD	— Inspect and change file parameters.
FORMAT	— Conversion of data to symbolic external form.
FTEST	— Check existence of a file.
LIST	— List files.
MOVE	— Move data in database.
TURN	— Change program switches.

2. Graphic output

BODE	— Plot Bode diagrams.
HCOPY	— Make hard copy.
PLMAG	— Magnify plot and allow changes of data.
PLOT	— Plot curves with linear scales.

3. Time series operations

ACOF	— Compute autocorrelation function.
CCOF	— Compute crosscorrelation function.
CONC	— Concatenate time series.
CUT	— Extract a part of a time series.
INSI	— Generate time series.
PICK	— Pick equidistant time points.
SCLOP	— Do scalar operations on a time series.
SLIDE	— Introduce relative delays between time series.
STAT	— Compute statistical characteristics.
TREND	— Remove a trend.
VECOP	— Do vector operations on a time series.

4. Frequency response operations

ASPEC	— Compute an auto spectrum.
CSPEC	— Compute a cross spectrum.
DFT	— Discrete Fourier Transform.
FROP	— Operate on frequency responses.
IDFT	— Inverse Discrete Fourier Transform.

5. Simulation and model analysis

DETER	— Deterministic Simulation.
DSIM	— Simulation with noise.
FILT	— Compute a filter system.
RANPA	— Pick parameters from a random distribution.
RESID	— Compute residuals with statistical test.
SPTRF	— Compute the frequency response of a transfer function.

6. Identification

LS	— Least Squares identification.
ML	— Maximum Likelihood identification.
SQR	— Least Squares data reduction.
STRUC	— Least Squares structure definition.

Table 1. Examples of program sizes

	Number of commands	Source code lines	Program size kbytes
INTRAC	17	7 000	90
IDPAC	39	37 000	470
MODPAC	37	41 000	570
SIMNON	24	25 000	360
SYNPAC	46	43 000	630
POLPAC	32	32 000	460

sed in Section 3. The different packages are listed in Table 1, which also summarizes some data about them. Brief descriptions of the different packages are given below. The commands used in the packages are listed in the appendices. There are also a large number of macros available for all packages.

IDPAC

IDPAC is a package for data analysis and identification of linear systems having one output and many inputs. Time series analysis of ARMA and ARIMA models is a special case. The package has commands for manipulation and plotting of data, correlation analysis, spectral analysis and parametric system identification. There are also commands for model validation and simulation. The basic techniques used for parameter estimation are the least squares method and the maximum likelihood method. By using the macro facility it is, however, possible to generate commands for most of the parameter estimation methods which are proposed in literature. It was actually in the development of IDPAC that the power of the macro concept became apparent. In the early IDPAC versions there were many commands necessary to cover the available identification methods. It was, however, discovered that almost all methods could be obtained by combinations of correlations analysis, spectral analysis, least squares and maximum likelihood estimation. Commands were thus constructed to give primitives for these operations and the special methods were then implemented as macros which used the primitive commands. This approach is also a pedagogical way to structure the problem area.

IDPAC can be viewed as a convenient way of packaging the research in systems identification that has been done at our department for 15 years. IDPAC has gone through several steps of development. It grew out of the software described in Åstrom *et al.* (1965). The latest version is described in Wieslander (1980b). The Åstrom paper (1980) gives the relevant theory for the parametric identification methods. It also contains a comprehensive set of examples of using IDPAC. A summary of the commands are given in Appendix B. Descriptions of some of the IDPAC macros are given in Gustavsson (1979). Typical examples of using IDPAC are given in Gustavsson and Nilsson (1979).

MODPAC

There are many ways to describe a control system. Nonparametric methods in the time and frequency domain can be used. Parametric descriptions like state equations, rational transfer functions and fractions of matrix polynomials may also be used. There are also many ways in which state equations can be transformed. For digital control it is necessary to go between continuous time and discrete time representations. All these problems can be handled by MODPAC. The package also has facilities for finding the Kalman decomposition of a system and for calculating observers. MODPAC is described in Wieslander (1980c). A list of the commands in MODPAC is given in Appendix C.

SIMNON

SIMNON is a package for interactive simulation of nonlinear continuous time systems with discrete time regulators. The package also includes noise generators, time delays, a facility for using data files from IDPAC as inputs to the system, and an optimizer.

SIMNON allows a system to be described as an interconnection of subsystems. There are two types of subsystems, continuous time systems and discrete time systems. This makes SIMNON well suited for simulation of digital control systems. The characteristics of SIMNON are illustrated by an example.

Listing 1 gives a description of a feedback loop consisting of a continuous time process called PROC and a digital PI regulator called REG. The process is an integrator with input saturation. The interconnections are described by the connecting system CON.

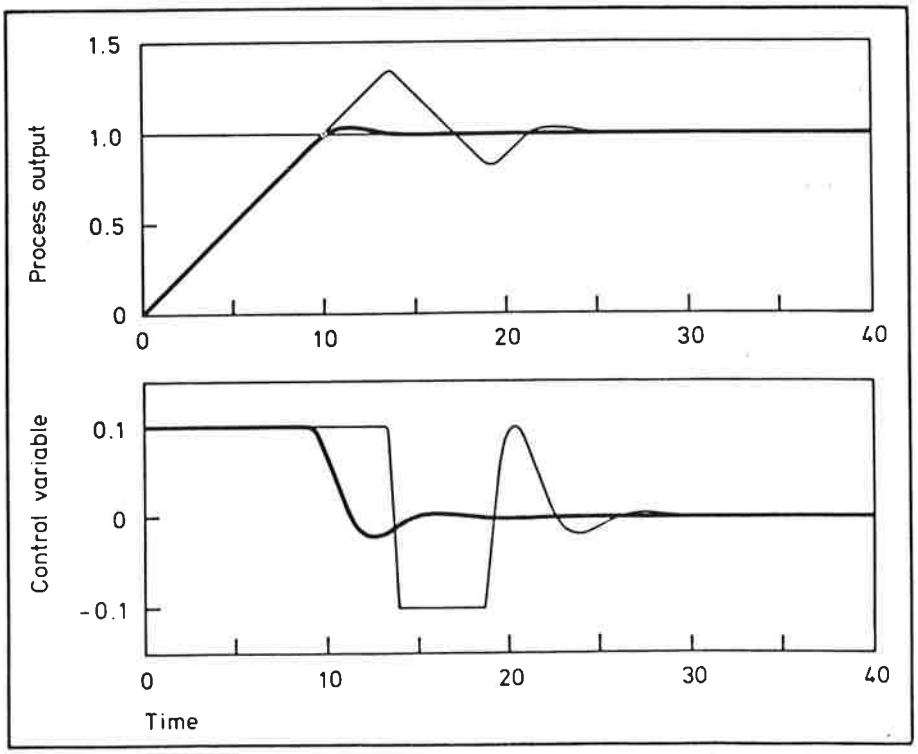


Fig. 3. Results of simulation of process with a PI regulator. Thin lines show results with ordinary regulator and thick lines show results for regulator with anti-windup.

The following annotated dialog illustrates how SIMNON is used.

Command	*Action*
SYST PROC REG CON	Activate the systems.
AXES H 0 100 V −1 1	Draw axes
PLOT yr y[proc] u[reg]	Determine variables to be plotted
STORE yr y[proc] u[reg]	Select variable to be stored
SIMU 0 100	Simulate
SPLIT 2 1	Form two screen windows
ASHOW y	Draw y with automatic scaling and yr with the
SHOW yr	same scales in first window.
ASHOW u	Draw u with automatic scaling in second window.

The results are shown by the curves in the thin lines shown in Fig. 3. These curves show that there is a considerable overshoot due to integral windup. The regulator REG has anti-windup. The state of the regulator is reset when its output is equal to ulow or uhigh. The limits were set to ulow=−1 and uhigh=1 in the simulations shown with thin lines in Fig. 3. These values are so large that the integral is never reset. The simulation shown in thin lines in Fig. 3 thus corresponds to a regulator without wind-up. The actual actuator limitations correspond to ulow=−0.1 and uhigh=0.1.
The command

```
PAR ulow: −0.1

PAR uhigh: 0.1
```

change the parameters and the command SIMU now generates the curves shown in the thick lines in Fig. 3. Notice the drastic improvements due to the nonlinearity in the regulator.

The first version of SIMNON was implemented in an MS project. SIMNON has gone through several stages of development. (See Elmquist, 1975 and 1977.) A list of the commands in SIMNON is given in Appendix D.

SYNPAC

SYNPAC is a state space oriented design package. It includes facilities for calculating state feedback and Kalman filters for continuous and discrete time LQG problems. It also has facilities for transforming continuous time problems into discrete time problems.

An example illustrates some features of SYNPAC. Consider the standard LQG problem

$$dx = Axdt + Budt + dv$$

$$dy = Cxdt + de$$

```
CONTINUOUS SYSTEM PROC
''Integrator with input saturation
Input u
Output y
State x
Der dx
upr=if u<−0.1 then −0.1 else if u<0.1
  then u else 0.1
dx=upr
END

DISCRETE SYSTEM REG
''PI regulator with anti-windup
Input yr y
Output u
State i
New ni
Time t
Tsamp ts
e=yr−y
v=k*e+i
u=if v<ulow then ulow else if v<uhigh
  then v else uhigh
ni=i+k*h*e/ti+u−v
ts=t+h
k:1
ti:1
h:0.5
ulow:−1
uhigh:1
END

CONNECTING SYSTEM CON
''Connecting system for simulation of
  process PROC
''with PI regulation by system REG
yr[REG]=1
y[REG]=y[PROC]
u[PROC]=u[REG]
END
```

Listing 1. SIMNON description of a simple control loop consisting of a continuous time process and a discrete PI regulator.

where {v} and {e} are Wiener processes with joint incremental covariances

$$\operatorname{cov}\begin{bmatrix} dv & dv \\ de & de \end{bmatrix}^T = \begin{bmatrix} R_1 & R_{12} \\ R_{12}^T & R_2 \end{bmatrix}.$$

Let the control problem be to minimize

$$J = \lim_{T\to\infty} \frac{1}{T} E \int_0^T [x^T(t)Q_1 x(t) + 2x^T(t)Q_{12}u(t) + u^T(t)Q_2 u(t)]\, dt.$$

Furthermore assume that a digital regulator will be used and that sampling periods from 0.5 to 5 s are of interest.

Assume that a system description which contains the matrices A, B, C, R_1, R_{12}, R_2, Q_1, Q_{12} and Q_2 has been introduced in a file called CSYS, and that the design parameter, i.e. the parameter which will be modified in the design, is the 3,3 element of the matrix Q_1. The following macro then executes the design

```
 1: MACRO DESIGN ALPHA
 2:   ALTER Q1 3 3 ALPHA
 3:   FOR H = 0.5 to 5 STEP 0.5
 4:     SAMP DSYS ← CSYS H
 5:     TRANS Q DSYS ← CSYS H
 6:     TRANS R DSYS ← CSYS H
 7:     OPTFB L ← DSYS
 8:     KALFI K ← DSYS
 9:     CONNECT CLSYS ← DSYS K L
10:     SIMU Y X ← CLSYS UREF
11:     PLOT X(1) X(7) X(8) XE(1) U
12:   NEXT H
13: END {MACRO}
```

Line numbers have been introduced only to be able to describe the algorithm. A macro with the name DESIGN and the parameter ALPHA is defined on line 1. The macro definition ends on line 13. The 3,3 element of the matrix Q1 is assigned the value ALPHA on line 2. Line 3 is a repetition statement which repeats the commands 4 through 11 for sampling periods 0.5 to 5 with an increment of 0.5. The system description is sampled on line 4 and the criterion and the covariances are transformed on lines 5 and 6. The command on line 7 computes the optimal state feedback matrix L and the command on line 8 computes the Kalman filter gain. The command on line 9 forms a closed loop system composed of the original system, the Kalman filter and the state feedback. The command on line 10 simulates the closed loop system with a reference input UREF. The command on line 11 plots statevariables 1, 7 and 8, the estimate

Appendix C–MODPAC Commands

1. Utilities

AGR	— Edit an aggregate file.

The rest are the same as in IDPAC.

2. Graphic output

BODE	— Plot Bode diagrams.
HCOPY	— Make hard copy.
NIC	— Display a frequency response in a Nichols diagram.
NYQ	— Display a frequency response in a Nyquist diagram.
PLEV	— Display eigenvalues, etc. in the complex plane.
PLOT	— Plot curves with linear scales.

3. Matrix operations

ALTER	— Alter elements in a matrix.
EIGEN	— Compute eigenvalues of a matrix.
ENTER	— Enter a matrix element by element.
EXPAN	— Generate a matrix from sub-matrices.
MATOP	— Perform matrix operations.
REDUC	— Extract a submatrix.
UNITM	— Generate a unit matrix.
ZEROM	— Generate a zero matrix.

4. Polynomial operations

POCONV	— Polynomial image—polynomial file conversion.
POLY	— Generate or edit a polynomial.
POLZ	— Compute and plot the zeroes of a polynomial.
ZERPOL	— Create a polynomial from its zeroes.

5. System operations

CONT	— Convert to continuous time form.
KALD	— Do a Kalman decomposition.
SAMP	— Convert to discrete time form.
SPSS	— Compute the frequency response.
SSTRF1	— Convert from state space to tranfer function.
SYST	— Generate a system description.
SYSTR	— Do a general coordinate transformation.
TBALAN	— Transform to balanced form.
TCON	— Transform to controllable form.
TDIAG	— Transform to diagonal form.
THESS	— Transform to Hessenberg form.
TOBS	— Transform to observable form.
TRFSS1	— Convert from transfer function to state space.

of the first state and the control signal. The following dialog illustrates how the macro may be used:

```
EDIT FILE CSYS

INPUT UREF ← STEP

DESIGN 3

DESIGN 8.
```

The system file is first edited. A step is generated as a command signal and the macro design is executed with parameters 3 and 8.

The example illustrates some SYNPAC commands. It also shows how a macro may be used to create a special purpose command.

SYNPAC was the first package that was implemented. It was based on the FORTRAN programs described in Åstrom (1963). A test version was made as an MS project. The current version is described in Wieslander (1980d). A list of the commands in SYNPAC is given in Appendix E.

POLPAC

POLPAC is a polynomial oriented design package for multi-output single-input systems. It includes algorithms for pole placement, minimum variance control, and LQG control. The package allows classical design using root loci and Bode plots. Root loci may be drawn with respect to arbitrary parameters. A list of the commands in POLPAC is given in Appendix F.

Portability

The programs were initially written in FORTRAN for a minicomputer PDP-15. A considerable effort was also devoted to development of subroutine libraries and programming standards. (See Elmquist *et al.* 1976, Wieslander 1977, and Cowell 1977.) The advantage of using a large main frame computer for program development was soon apparent. The program development was therefore moved to a Univac 1108 at the Lund University Computing Center. More powerful program development tools like PFORT could then be used. (See Ryder, 1975.) Since there was a considerable interest from external groups to use the programs, a substantial effort went into making the software portable. This included development of FORTRAN routines for file, character and string handling. A plotting library in FORTRAN was also developed. These routines are interfaced with a well defined small set of installation dependent routines. A result of the efforts is that the packages are indeed portable. Packages are currently running on the following computers: PDP-15, PDP-11, DEC-10, VAX-11/780, NOVA-3, Nord-100, ECLIPSE, IBM-1800, IBM-360, CDC-1700, CDC-6400, HP-3000, Honeywell, SEL-32, Univac 1108, PRIME-750.

Large Systems

When working with the projects it was found that there were certain problems where interactive computing is not feasible. These problems typically involve large systems where it can easily happen that the computing time required is so large that it does not make sense to wait for the results at the terminal. We experienced this in connection with identification and simulation of large systems.

LISPID

For identification of large systems we found that it was better to use a batch program which allows an interactive start up and an interactive inspection of the results. LISPID is an example of such a program. This program allows estimation of parameters in linear stochastic systems with arbitrary parameterization and in special types of nonlinear systems. (See Kallstrom *et al.*, 1976.)

DYMOLA

Another problem was also encountered in connection with modeling of large systems. It is straightforward to write down and

Appendix D–SIMNON Commands

1. Utilities

EDIT	— Edit system description.
GET	— Get parameters and initial values.
LIST	— List files.
PRINT	— Print files.
SAVE	— Save parameter values and initial values in a file.
STOP	— Stop.

2. Graphic output

AREA	— Select window on screen.
ASHOW	— Plot stored variables with automatic scaling.
AXES	— Draw axes.
HCOPY	— Make hard copy.
SHOW	— Plot stored variables.
SPLIT	— Split screen into windows.
TEXT	— Transfer text string to graph.

3. Simulation commands

ALGOR	— Select integration algorithm.
DISP	— Display parameters.
ERROR	— Choose error bound for integration routine.
INIT	— Change initial values of state variables.
PAR	— Change parameters.
PLOT	— Choose variables to be plotted.
SIMU	— Simulate a system.
STORE	— Choose variables to be stored.
SYST	— Activate systems.

check the balance equations. It is, however, a major effort to reduce the equations obtained to forms that are suitable for simulation and control design. The language DYMOLA, which admits a simple description of a large hierarchical system, was therefore developed. (See Elmqvist, 1978, 1979a, and 1979b.) Experimental software, which operates on the basic system description and generates simulation programs, for example in SIMNON, and linearized system equations have also been developed. We believe that this is an important step towards effective methods for dealing with large systems.

Experiences of Using The Packages

The packages have been used at our department, at other universities, and in industry. The early use of the packages provided very good feedback to their development. There has been a continuous dialog between users and implementors at all stages of the development. Very valuable input was provided by visitors to the department. They often had different ideas on how to use the programs.

All staff members of the department and a large number of the students, who have done MS and PhD dissertations, have used the packages. The programs have been used in some of our advanced courses and in courses for industrial audiences. They are now being introduced also in elementary courses. The bottleneck for this has been the availability of a sufficient number of graphic terminals. By using the packages, it has been possible to focus on concepts and ideas in the lectures and to work with realistic examples with considerable detail in exercises and projects.

The simulation language SIMNON is used as a standard language for documenting models. The availability of a library of realistic models of different complexity is of course very beneficial in teaching. SIMNON has been used in an interesting way in a forthcoming book on computer control (Åstrom and Wittenmark, 1984), which makes extensive use of simulation. All simulation results are implemented as Macros in SIMNON which are accessable from the student terminals. This means that the students may conveniently check the results and also look into effects of variations of data. SIMNON has also been used in many applied projects at the institute. A typical example is a study of modeling and simulation of a wind turbine (Bergman *et al.*, 1981).

We have found IDPAC to be a very good tool to teach system identification. It is possible for the students to gain much experience by working with real data. IDPAC has also been used in many industrial projects. Typical examples are given in Åstrom and Kallstrom (1976) and Kallstrom and Å (1981). Troubleshooting in the paper industry has been another interesting application area. (See Lundqvist and Nordstrom, 1980, and Johansson *et al.*, 1980.)

Similarly, we have found that SYNPAC is an excellent tool for teaching LQG design. The students can work with realistic problems with reasonable effort. SYNPAC has also been used to design control laws for digital flight control systems. These control laws have also been flight tested. (See Åstrom and Elgerona, 1976, and Folkesson *et al.*, 1982.)

The programs are used by a number of industries in Sweden, and to a limited extent outside Sweden. To spread knowledge about the packages we have given a number of courses on the use of interactive computing. It has been our experience that the average engineer can use the tools quite well after a one week course. The macro facility is very useful because it makes it possible to tailor a few special commands for the standard needs of each user.

Future Work

Computer aided design of control systems is still in its early stages. There are a number of packages like ours. An overview of some packages are found in Atherton (1981), Edgar (1981), Edmunds (1979), Frederick (1982), Hashimoto and Takamatsu (1981), Lemmens and van den Boom (1979), Munro (1979), Rosenbrock (1974), Tysso (1981) and Wieslander (1979b). More references are also found in these papers. Special workshops and symposia devoted to CAD for control systems have been organized by IFAC, GE-RPI, and IEEE CSS. (See Mansour, 1979, Leininger, 1982, Spang and Gerhart, 1981, Herget and Polak, 1982.) Computer aided tools are also popular in many other fields such as mechanical design and VLSI design. The seminal work on computer graphics by Newman and Sproull (1979) and the Foley and van Dam text (1982) contain much material and many references.

The field is in a state of rapid development due to an increased understanding of the technology and the drastic development of computer and graphics hardware. It is safe to predict that future computer aided design tools will be much more powerful than the packages described in

this paper. Some speculations on future development are given in this section.

Computer hardware

The virtual memory requirement for each package is less than 1 Mbyte. A package can run on a computer having a primary storage of 128 kbytes. They require fast floating point operations for reasonable efficiency. The packages could be implemented on a computer like the IBM personal computer with an Intel 8087 floating point processor. A good FORTRAN compiler is required to do this with a reasonable effort.

The personal computers which are projected to appear within a few years have specifications like: a primary memory of 2 Mbytes, a secondary memory of 100 Mbytes, a computing speed of one megaflop/s, and a price less than 20kS. (See Dertouzos and Moses, 1980.) These computers are also expected to have a high resolution bit mapped color graphics display. With computers like this it is possible to have single user workstations with packages which are much more sophisticated than all our current packages. The existence of computers like Apollo, Lisa, PERQ and Sun make the predictions quite credible.

There has been a drastic development of the computer output devices. A teletype is capable of writing at a speed of 10 ch/s (110 Baud). A regular terminal connected to a 19.2 kBaud channel can write a screen of 80 × 24 ch in a second. A good vector graphics terminal can refresh up to 100,000 long vectors or a million short vectors per second. A high resolution bit mapped display may refresh 512 × 512 pixel frames at rates of 60 frames/s (15 Mbit/s).

Unfortunately, the input devices have not developed at the same rate. We still have ordinary keyboards. (See Montgomery, 1982.) A very good typist may type at a rate of 8 ch/s. A normal engineer types considerably slower. Pointing devices like roll balls, mouses and touch panels have been invented. These devices may, perhaps, be used to increase the input rate indirectly by combining the rapid output rate with feedback via the picking device (dynamic menus). Speech input is another possibility. There are, however, no indications of a more drastic increase in the input rate.

The renaissance of graphics

Graphics have played a major role in engineering. The first books used in engineering education were books of drawings of machines by Leonardo da Vinci. Graphical representations have been used extensively ever since. Graphics in the forms of Bode diagrams, Nichols charts, root loci, block diagrams and signal flow diagrams are important tools in classical control theory. Modern control theory has, however, not been much influenced by graphics. This can partly be explained by lack of proper tools for graphics. The situation may change drastically in the future because good graphics hardware will be available at a reasonable cost.

The man-machine interface

A high bandwidth information transmission is required for an efficient man-machine communication. This implies a high rate of transmission of symbols and a high information content in each symbol. The user interfaces in our packages were designed for teletypes combined with graphic terminals having storage screens and data rates of 4800 Baud. These were the only tools available at reasonable cost when our design was frozen. A storage scope is very limited. Curves may be shown but they can not be erased individually. Bit mapped graphics are faster and much more flexible. Individual picture elements may be changed instantaneously. This makes it possible to zoom, scroll and pan a picture. Color and animation add extra dimensions. Imaginative use of color graphics is still in its infancy in CAD packages for control systems. Interesting ideas have been proposed by Polak (1982) in connection with applications of optimization techniques. Animation has not been used much. It is clear that we have a lot to learn from designers of video games. (See Perry *et al.*, 1982.)

The information content of each symbol is related to the expression power of the

Appendix E-SYNPAC Commands

1. Utilities

Same as in IDPAC.

2. Graphic output

Same as in MODPAC.

3. Time series operations

CONC	— Concatenate two time series.
CORNO	— Generate a correlated noise time series.
CUT	— Extract a part of a time series.
INSI	— Generate a time series.
PICK	— Pick equidistant time points.
SCLOP	— Do scalar operations on a time series.
STAT	— Compute statistical characteristics.
VECOP	— Do vector operations on a time series.

4. Matrix operations

Same as in MODPAC.

5. System conversion and analysis

CONT	— Convert to continuous time form.
POLES	— Compute the poles of a system.
SAMP	— Convert to discrete time form.
SIMU	— Simulate the time response of a system.
SPSS	— Compute the frequency response of a system.
SYSOP	— Generate a system from its subsystems.
SYST	— Generate and edit system descriptions.
TRANS	— Convert a criterion from continuous time to discrete time form.

6. Design

FEEDF	— Design feedforward control.
KALFI	— Compute a Kalman filter gain.
LUEN	— Compute a Luenberg observer.
OPTFB	— Compute a linear quadratic state feedback.
PENLT	— Reduce a penalty function to standard form.
RPLAC	— Pole replacement for single input systems.
RECON	— State reconstruction for single input systems.
REDFB	— Compute an output feedback.

commands. Our experience indicates clearly the need for having a CAD language with considerable expression power. It would be nice to describe all operations using notations similar to those used in system theory. An expression parser is needed. Macros are very useful to increase the efficiency of the man-machine dialog. More flexible control structures and more powerful commands than those used in INTRAC would be desirable. One possible extension is the system DELIGHT which is based on the language RATTLE, developed by Nye *et al.* (1981). Other possibilities are to replace INTRAC by languages with an interactive implementation like APL, Lisp or Logo, or an interpretive threaded language like FORTH. (See Winston and Horn, 1981, Abelson, 1982, and Kogge, 1982.)

Short form commands and default values are two simple techniques for increasing the efficiency of the man-machine dialog. More sophisticated techniques like conceptual dependency are found in semantically oriented programs for processing natural language. (See Schank, 1975, and Schank and Abelson, 1977.) It would be interesting to explore how these ideas can be incorporated in CAD systems. Another possibility is to have an operator communication which is more oriented towards graphics. Interesting ideas in this direction are demonstrated in Elmqvist (1982).

Numerical algorithms and design tools

The numerical algorithms for the design primitives are key elements in the software. There have been major advances in numerical software for linear algebra over the past years. A substantial effort has gone into subroutine packages such as EISPACK, Garbow *et al.* (1977), and Smith *et al.* (1976), and LINPACK, Dongarra *et al.* (1979), which are now available in the public domain. A similar effort has not yet been devoted to the numerical calculations required for analysis and design of control systems, although libraries are maintained at many departments. The numerical problems that arise in automatic control are, however, starting to receive attention from numerical analysts. (See van Doren, 1981, Hammarling, 1982, and Laub, 1980.) This is crucial for future development.

Most data processing in current packages is inspired from numerical analysis. The powers of non-numeric data processing have not been exploited. It would be highly desirable to have facilities for symbolic manipulation. This can, for example, be used for model simplification, generation of code for computing equilibrium points, generation of simulation code, linearization, etc. If symbolic manipulations are included, it is also possible to generate code for realization of the control laws. Symbolic calculations were not used in our packages because of the limited computing facilities available. It is, however, feasible in future packages.

When transferring our packages, we have noticed that their power increases considerably if an experienced user is around. The possibilities of providing the packages with a rule based expert system or an advisory system (Barr and Feigenbaum, 1982) is, therefore, very appealing. It is an interesting research problem to find out if expert knowledge in identification, analysis, and design of control systems can be incorporated in the packages.

Implementation languages

The source code for the smallest package described in this paper is about 30,000 lines of source code. A future package may be an order of magnitude larger. A good programming environment and efficient software tools are necessary to develop and maintain such systems. FORTRAN was used in our packages to make them portable. It is, however, unlikely that future systems will be written only in FORTRAN.

FORTRAN libraries like EISPACK, LINPACK, hopefully also a control package, and graphics packages will probably be used. Although INTRAC was written in FORTRAN it is not convenient to do so. Pascal would be much more convenient, particularly if we want to include formula manipulation and the other features that we may expect in a future system. It is thus likely that future systems will be implemented using several programming languages. One indication of this is the design package ISER-CSD which is written in FORTRAN and Pascal. (See Suleyman, 1980.) This system is, however, restricted to one computer system.

Another possibility is to base the interaction on an existing language with an interactive implementation like APL, Lisp or Logo. Systems based on LISP will be extendable automatically, symbolic manip-

Appendix F–POLPAC Commands

1. Utilities

Same as in IDPAC.

2. Graphics

BODE	— Plot Bode diagrams.
HCOPY	— Make hard copy.
LOCPLOT	— Plot root locus diagrams.
NIC	— Plot Nichols diagrams.
NYQ	— Plot Nyquist diagrams.
PLEV	— Plot eigenvalues and edit.
PLOT	— Plot curves with linear scales.

3. System and polynomial operations.

INSI	— Generate a data file.
POLOP	— Evaluate algebraic polynomial expressions.
POLSYS	— Create a system file or a polynomial file.
POLY	— Generate or edit a polynomial.
POLZ	— Compute and plot the zeros of a polynomial.
SIMU	— Simulate a system.
SYSOP	— Build a system from subsystems.

4. Analysis

PROP	— Compute bandwidth, rise time, error coefficients.
ROTLOC	— Compute the root locus.
ROUTH	— Compute and display Routh's tableau.
TRFFR	— Compute frequency response of a transfer function.
TRFSIM	— Simulate.

5. Synthesis

DEADBE	— Compute dead-beat strategy.
MIVRE	— Compute minimum variance control law.
POLPLA	— Make a pole placement design.

ulations are also easy to implement. There are good programming environments for Lisp which have been used to implement very large systems. Natural language interfaces and expert systems are also often written in Lisp.

The programming language Ada (Dod, 1980), which will be available in a few years, is another interesting alternative. The basic subroutine libraries can conveniently be implemented as packages in Ada. A wide range of libraries can be expected to be available for Ada. Since Ada supports the concept of tasks it will also be possible to apply ideas from concurrent programming. A good programming environment which will be a substantial help in software development is also planned for Ada. The deciding factor will probably depend on how well Ada will be accepted.

Some computations, such as simulation and identification, are quite demanding computationally. The problem solving would be more efficient and convenient if the user could perform several tasks like plotting, editing and report writing in parallel. This mode of operation is particularly useful for a system with windowing. (See Goldberg *et al.*, 1983.)

Implementation tools

Our packages were developed from scratch. Future packages may be expected to use ready made modules to a much larger extent.

Descriptions of control systems problems require flexible data structures. Many problems may be characterized in terms of arrays only. Arrays will go a long way to describe linear systems in state space form and to describe signals. Many problems can be solved using a matrix language like MATLAB (Moler, 1980), and one of its extensions, Matrix$_x$ (Walker *et al.*, 1982). It is, however, clear that it is very useful to also have polynomials, rational functions and good general structures for linear and nonlinear systems. In some cases it is also valuable to describe systems as hierarchical interconnections of subsystems.

For simple systems with only one data type, like matrices, all data may be stored in a stack or in a simple array. In our packages, which used more sophisticated data structures, the data was stored in files. Our experiences indicate that it would be very useful to have a more flexible system. It is probably a very good idea to build a system around some general database system. The need for multiple descriptions of a system is one special problem which is conveniently solved using databases. A typical example is when a system is represented both as a transfer function and as a state equation. Small systems are not much of a problem because it is easy to transform from one form to another. Such computations may, however, be extensive for large systems. To obtain a reasonable efficiency it is then necessary to store the different descriptions. It may also be desirable to have models of different complexity for the same physical object as well as linearized models for different operating conditions. Since it is very difficult to visualize all possible combinations *a priori*, it is useful to have a data base system which admits modifications of the structure of the data.

In our packages we had to develop our own graphics interface. A few simple routines which were compatible with Tektronix 4010 systems were used. The situation will be much better when standards like Graphical Kernel System (GKS) and raster graphics extensions of Siggraph Core materialize. See Anon. (1982) and Foley and van Dam (1982).

Conclusions

Interactive computing is a powerful tool for problem solving. An engineer can come to the work station with a problem and he can leave with a complete solution after a few hours. The results are well documented in terms of listings, text and graphs. The problem solver can obtain the solution by himself without relying on programmers as intermediaries. Our projects have shown that the productivity in analyzing and designing control systems can be increased substantially by using these tools. We believe that interactive computer aided design tools can make modern control theory cost effective.

Computer aided design of control systems is still in its infancy. A small number of systems have been implemented in a few places. There are many possible future developments which are mainly driven by the computer development. Packages of the type we have been experimenting with can easily be fitted into the personal computers or work stations that will be available in a few years. The bit mapped high resolution color displays that will be available on these computers offer new possibilities for an efficient man-machine dialog. With the drastic increase in computer capacity that is forthcoming, it is also possible to make much more ambitious projects. Applications of computer aided design also appear in many other branches of engineering. Cross fertilization between the fields will most likely lead to a rapid development.

Acknowledgements

The work reported in this paper has been supported by the Swedish Board of Technical Development for many years. This support is gratefully acknowledged. The projects, which the paper draws upon, have been true team efforts. Many members of the department have contributed to discussions of command structures, implementation, testing and evaluation. Particular thanks are due to Johan Wieslander and Hilding Elmqvist, who generated many of the important ideas, and to Tommy Essebo and Thomas Schonthal, who did a major part of the programming of the packages.

References

Abelson, H. (1982): *LOGO for the Apple II.* Byte/McGraw-Hill, Peterborough, New Hampshire.

Anonymous (1982): Graphical Kernel System (GKS)—Functional Description. Draft International Standard ISO/DIS 7942 Version 7.02, August 9, 1982. Available through American National Standards Institute Inc. New York, N.Y.

Armstrong, E. S. (1980): *ORACLS—A design system for linear multivariable control.* Marcel Dekker, New York.

Åström, K. J. (1963): On the choice of sampling rates in optimal linear systems. Report RJ-243. IBM Research Laboratory, San Jose, CA.

Åström, K. J., Bohlin, T. and Wensmark, S. (1965): Automatic construction of linear stochastic dynamic models for stationary industrial processes with random disturbances using operating records. Report TP 18.150, IBM Nordic Laboratory, Sweden.

Åström, K. J. and Kallstrom, C. G. (1976): Identification of ship steering dynamics. *Automatica* 12, 9-22.

Åström, K. J. (1979): Reflections on theory and practice of automatic control. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden. Report CODEN: LUTFD2/(TFRT-7178)/1-26/(1979).

Åström, K. J. (1980): Maximum likelihood and prediction error methods. *Automatica* 16, 551-574.

Åström, K. J. and Elgcrona, P. O. (1976): Use of LQG theory and SYNPAC in design of flight control systems. Reports SAAB, Linkoping, Sweden.

Åström, K. J. and Wieslander, J. (1981): Computer aided design of control systems—Final Report STU Projects 73-3553, 75-2776 and 77-3548. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD2/(TFRT-3160)/1-23/(1981).

Åström, K. J. and Wittenmark, B. (1984): *Computer Control Theory.* Prentice Hall, Englewood Cliffs, N. J. (to appear).

Atherton, D. P. (1981): The role of CAD in education and research. IFAC Congress VIII, Kyoto, Japan.

Barr, A. and Feigenbaum, E. A. (1982): *The Handbook of Artificial Intelligence, Vol. II.* W. Kaufmann Inc. Los Altos, Calif.

Bergman, S., Mattson, S. E. and Ostberg A. B. (1981): A modular simulation model for a wind turbine system. AIAA-81-2558. Second AIAA Terrestrial Energy Systems Conference, Colorado Springs. To appear in AIAA Journal.

Cowell, W. (Ed) (1977): Portability of numerical software, Lect. Notes in Comp. Sci., Vol. 57, Springer-Verlag, New York.

Dertouzos, M. L. and Moses, J. (1980): *The Computer Age: A Twenty Year View*. MIT Press, Cambridge, Mass.

DOD (1980): Reference Manual for the Ada Programming Language. Defense Advanced Research Projects Agency. United States Department of Defense.

Dongarra, J. J., Moler, C. B., Bunch, J. R. and Stewart, G. W. (1979): LINPACK—Users' Guide. SIAM, Philadelphia.

Edgar, T. F. (1981): New results and the status of computer-aided process control system design in North America. Engineering Foundation Conference on Chemical Process Control—II, Sea Island, Georgia.

Edmunds, J. M. (1979): Cambridge linear analysis and design programs. IFAC Symposium on Computer Aided Design of Control Systems, Zurich, 253-258.

Elmqvist, H. (1975): SIMNON—An Interactive Simulation Program for Nonlinear Systems. Report 7502. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Elmqvist, H., Tysso, A. and Weislander, J. (1976): Scandanavian control library. Programming. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD2/(TFRT-3139)/(1976).

Elmqvist, H. (1977): SIMNON—An Interactive Simulation Program for Nonlinear Systems. Simulation '77, Montreux, Switzerland, June 1977.

Elmqvist, H. (1978): A Structured Model Language for Large Continuous Systems. Ph.D. Thesis. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD2/(TFRT-1015)/1-226/(1978).

Elmqvist, H. (1979a): DYMOLA—A Structured Model Language for Large Continuous Systems. Summer Computer Simulation Conference, Toronto, Canada, July 1979.

Elmqvist, H. (1979b): Manipulation of Continuous Models Based on Equations to Assignment Statements. Simulation of Systems '79. Sorrento, Italy, September 1979.

Elmqvist, H. (1982): A graphical approach to documentation and implementation of control systems. Proc 3rd IFAC/IFIP Symposium on Software for Computer Control, SOCOCO 82. Madrid, Spain.

Folkesson K., Elgcrona, P. O. and Haglund, R. (1980): Design and experience with a low-cost digital fly-by-wire system in the SAAB JA37 Viggen A/C. Proc 13th International Council of the Aeronautical Sciences, Seattle, WA.

Foley, J. D. and van Dam, A. (1982): *Fundamentals of Interactive Computer Graphics*. Addison Wesley, Reading, Mass.

Frederick, D. K. (1982): Computer packages for the simulation and design of control systems. Lecture notes. Arab school on science and technology.

Furuta, K. and Kajiwara, H. (1979): CAD system for control system design. J. of the Society of Instrument and Control Engineers, Japan, 18 (9). (In Japanese).

Garbow, B. S., *et al.* (1977): Matrix eigensystem routines—EISPACK Guide Extension. Lect. Notes in Comp. Sci., Vol. 51, Springer-Verlag, New York.

Goldberg, A. J., Robson, D. and Ingalls, D. H. H. (1983): *SMALLTALK-80: The Language and its Implementation*. Addison-Wesley, Reading, MA.

Gustavsson, I. (1979): Processidentifiering-overheadbilder (Process identification—transparencies). Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, CODEN: LUTFD1/(TFRT-7166)/1-248/(1979).

Gustavsson, I. and Nilsson, A. B. (1979): Ovningar for IDPAC (Exercises for IDPAC). Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD1/(TFRT-7169)/1-55/(1979).

Hammarling, S. (1982): Some notes on the use of orthogonal similarity transformations in control. NPL Report DITC.

Hashimoto, I. and Takamatsu, T. (1981): New results and the status of computer aided process control systems design in Japan. Engineering Foundation Conference on Chemical Process Control-II, Sea Island, Georgia.

Herget, C. J. and Laub, A. J. (Ed.) (1982): Proc IEEE CSS Workshop on Computer Aided Control System Design. Berkeley, Calif. *IEEE CSM* 2:4. Special Issue on Computer-Aided Design of Control Systems.

Johansson, B. L., Karlsson, H. and Ljung, E. (1980): Experiences with computer control based on optical sensors for pulp quality of a two state TMP plant. Preprints Process Control Conf. Canadian Pulp and paper Ass. Halifax, Canada.

Kogge, P. M. (1982): An Architectural trail to threaded-code systems. *Computer* 15:3 22-32.

Källström, C. G., Essebo, T. and Åstrom, K. J. (1976): A computer program for maximum likelihood identification of linear multivariable stochastic systems. Proc 4th IFAC Symposium on Identification and System Parameter Estimation, Tblisi, USSR.

Källström, C. G. and Åstrom, K. J. (1981): Experiences of system identification applied to ship steering. *Automatica* 17, 187-198.

Laub, A. J. (1980): Survey of computational methods in control theory. In Erisman, A. M. *et al.* (Eds.), Electric power problems. The mathematical challange, SIAM, Philadelphia, pp 231-260.

Leininger, G. (Ed) (1982): Computer aided design of multivariable technological systems. Preprints second IFAC Symposium on Computer Aided Design of Multivariable Technological Systems. West Lafayette, Indiana, USA.

Lemmens, W. J. M. and Van den Boom, A. J. W. (1979): Interactive computer programs for education and research: a survey. *Automatica*, 15, 113-121.

Lundqvist, S. O. and Nordstrom, H. (1980): The development of a control system for a pulp washing plant through the use of dynamic simulation. Preprints IFAC Conf. on Instrumentation and automation in the paper, rubber, plastics and polymerisation industries. Gent, Belgium.

Mansour, M. (Ed.) (1979): Preprints first IFAC Symposium on CAD of Control systems. Zurich. Pergamon.

Moler, C. (1980): MATLAB Users' Guide. Report, Department of Computer Science, University of New Mexico.

Munro, N. (1979): The UMIST control system design and synthesis suites. IFAC Symposium on Computer Aided Design of Control Systems, Zurich, 343-348.

Newman, W. M. and Sproull, R. F. (1979): *Principles of Interactive Computer Graphics*. McGraw-Hill, New York.

Nye, W., Polak, E., Sangiovanni-Vincentilli, A. and Tits, A. (1981): An optimization-based computer-aided-design system. Proc ISCAS, April 24-27.

Perry, T., Truxal, C. and Wallich, P. (1982): Video games: the electronic big bang. *IEEE Spectrum* 19:12 20-33.

Polak, E. (1981): Optimization-based computer-aided-design of control systems. Proc JACC. University of Virginia.

Rosenbrock, H. H. (1974): *Computer-Aided Control System Design*. Academic Press, New York.

Ryder, B. G. (1975): The PFORT verifier: User's Guide. CS Tech. Rept. 12, Bell Labs.

Schank, R. C. (1975): *Conceptual Information Processing*. North Holland. Amsterdam.

Schank, R. C. and Abelson, R. P. (1977): Scripts, plans, goals and understanding. Lawrence Erlbaum Associates, Hillsdale, NJ.

Smith, B. T., *et al.* (1976): Matrix eigensystem routines—EISPACK Guide. 2nd ed., Lect. Notes in Comp. Sci., Vol.. 6, Springer-Verlag, New York.

Spang, H. A. III and Gerhart, L. (Eds.) (1981): Preprints GE-RPI, Workshop on control design, Schenectady, N.Y.

Suleyman, C. (1981): Interactive system for education and research in control system design. IEEE International Conference on Cybernetics and Society, Atlanta, Georgia.

Tyssö, A. (1979): CYPROS: Cybernetic program packages. IFAC Symposium on Computer Aided Design of Control Systems, Zurich, 383-389.

Tyssö, A. (1981): New results and the status of computer aided process control systems design in Europe. Engineering Foundation Conference on Chemical Process Control-II, Sea Island, Georgia.

Van Doren, P. (1981): A generalized eigenvalue approach for solving Riccati equations. SIAM J Sci. Stat. Comput. 2, 121 135.

Walker, R., Gregory, C. and Shah, S. (1982): MATRIX$_X$ A data analysis, system identification, control design and simulation package. *IEEE CSM* 2:4 30-37.

Wegner, P. (1968): *Programming Languages, Information Structures, and Machine Organization*. McGraw-Hill, New York.

Wieslander, J. (1973): Computer aided control system design. IEE, Publ. no 96.

Wieslander, J. (1977): Scandinavian control library. A subroutine library in the field of automatic control. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD2/(TFRT-

3146)/1-38/(1977).

Wieslander, J. (1979a): Interaction in computer aided analysis and design of control systems. PhD thesis, Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD2/(TFRT-1019)/1-222/(1979).

Weislander, J. (1979b): Design principles for computer aided design software. Preprints, IFAC Symposium on CAD of Control Systems, Zurich, 493.

Wieslander, J. (1980a): Interactive programs—General guide. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD2/(TFRT-3156)/1-30/(1980).

Wieslander, J. (1980b): IDPAC commands—User's guide. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD2/(TFRT-3157)/1-108/(1980).

Wieslander, J. (1980c): MODPAC commands—User's guide. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD2/(TFRT-3158)/1-81/(1980).

Wieslander, J. (1980d): SYNPAC commands—User's guide. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD2/(TFRT-3159)/1-130/(1980).

Wieslander, J. and Elmqvist, H. (1978): INTRAC, A communication module for interactive programs. Language manual. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden, Report CODEN: LUTFD2/(TFRT-3149)/1-60/(1978).

Wilkinson, J. H. and Reinsch, C. (1971): *Linear Algebra*. Springer-Verlag, Berlin.

Winston, P. H. and Horn, B. K. P. (1981): LISP. Addison-Wesley, Reading, Mass.

Karl Johan Astrom was born in Ostersund, Sweden on August 5, 1934. He was educated at the Royal Institute of Technology (KTH) in Stockholm, Sweden. He has held various teaching positions at KTH, and he has worked for the Research Institute of Swedish Defense, and for IBM. In 1965 he was appointed to the chair of Automatic Control at Lund Institute of Technology (LTH). His main research interests are stochastic control theory, identification, adaptive control, and computer aided design of control systems. Astrom has also worked with industrial applications of automatic control in the fields of inertial guidance, paper mills, flight control, and ship steering. Apart from his professional work he and his family enjoy skiing and sailing.

STFI

Forskning och utveckling för massa och pappersindustrin

- ca 300 anställda

Finansiering

o Avtal: industrin-STU

o Uppdrag, speciella projekt

STFI

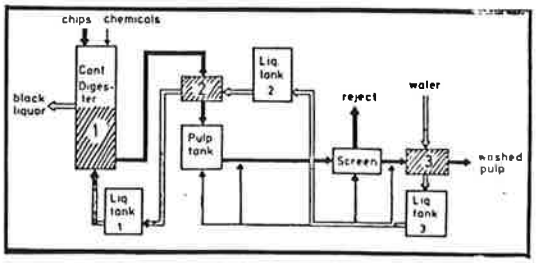
Verksamhet

- Processteknik, massa
- Processteknik, papper
- Grundläggande kemi
- Mät- och styrteknik
  - o massatillverkning
  - o papperstillverkning
  - o kemikalieåtervinning

SIMNON - typiska tillämpningar, STFI

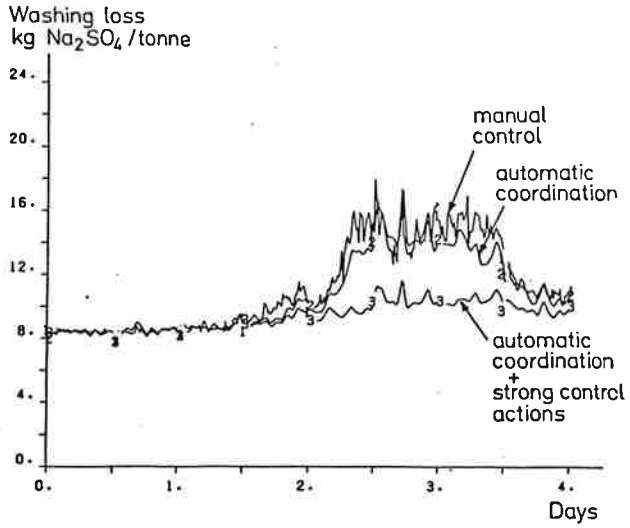
- o ökad processförståelse
- o utveckling och trimning av regulatorer
- o rekonstruktion
- o beräkningar, grafik
- o dataredigering

THE REFERENCE PLANT



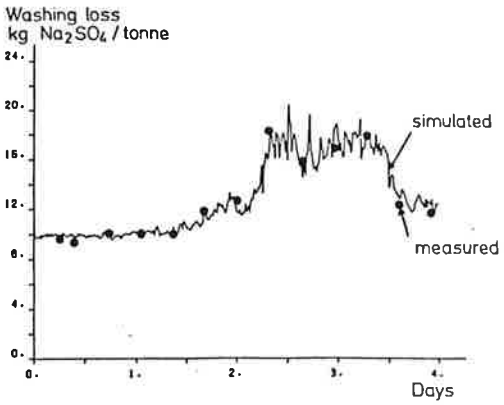
Vid STFI har SIMNON använts för att studera alternativa körstrategier för en fiberlinje i en sulfatmassafabrik.

SIMULATED EFFECTS OF CONTROL

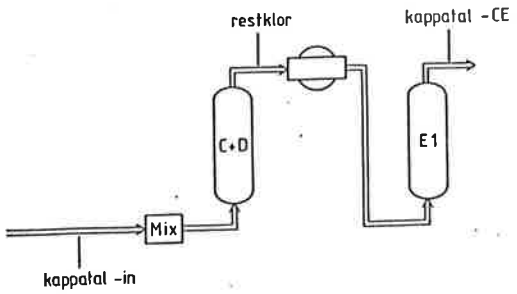


Vid manuell styrning ger en störning i kokaren en kvarstående nedsmutsning av tvättsystemet. Automatisk styrning förbättrar situationen obetydligt. Med hjälp av SIMNON-modellen visades att en förbikoppling av tvättvätska nästan helt kunde eliminera effekten av störningen.

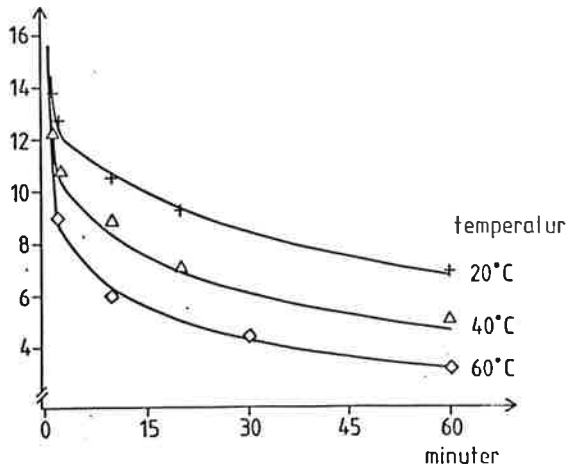
MODEL ACCURACY



SIMNON-modellen kunde med god noggrannhet följa ändringar i tvättförlusten.



Vid STFI har SIMNON använts för att studera och utveckla en regulator för klorblekningsprocessen. Kappatalet anger hur mycket lignin det finns kvar i massan och är därmed ett mått på hur vit massan är.



Kurvorna visar hur kappatalet minskar vid en "ideal" klorblekningsreaktion. Kemikaliehalterna i vätskefasen är konstanta. Motsvarande data för ett labförsök med klorblekning har markerats i figuren.

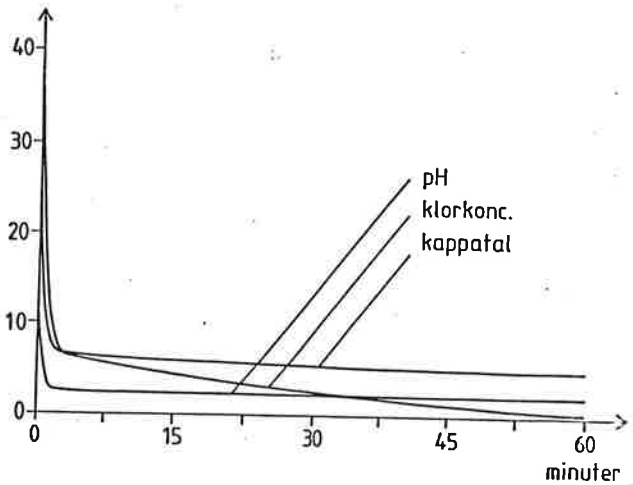
STFI

IDPAC ANVÄNDNING

- **PROCESSAMBAND**
- **DATAREDIGERING**
- **DATAPRESENTATION**
- **IDENTIFIERING**

Förbättrat processanalysverktyg

- **Ickevärdeshantering**
- **Bättre selektering**
- **Multipelregression**
- **Bättre redigeringsmöjligheter**
- **Asynkrona data, labdata**



I den verkliga processen minskar kemikaliehalterna vartefter klor förbrukas. Dynamiken blir då snabb i början och långsam på slutet, vilket gav numeriska problem i SIMNON. För att klara detta användes en modell utan flödesdynamik. Utgående kemikaliehalter beräknades för en given uppehållstid, som gavs av flöde och tankvolym.

Krav på ett modernt simuleringsverktyg

- o **delsystem av procedurtyp**
- o **algebraiska ekvationer**
- o **implicita ekvationer**
- o **bättre hantering av snitt mellan delsystem**
- o **meny- och kommandoorienterat**

IFIP/IFAC Symposium ASSOPO '85
Trondheim, Norway, 1985

THE USE OF HYBRID MODEL TESTS AND COMPUTER SIMULATIONS FOR OFFSHORE INSTALLATIONS

Claes G. Källström
SSPA Maritime Consulting AB
Göteborg, Sweden

&

Furkan A. Ramzan
Brown & Root (UK) Ltd.
London, UK

The effective approach to combining scale model tests and computer simulations for offshore installations is described. Suitable scale model tests and the development of mathematical models by applying system identification to the recorded data are discussed. The use of the mathematical models for analysis, statistical predictions, procedure development and operator training in real time is illustrated. The installation of the Conoco Hutton tension leg platform and the semi-submersible GVA 4000 are chosen as test examples.

1. INTRODUCTION

Offshore installations have become more and more complex, thus requiring careful planning and analysis of the different operations. Especially line tensions and motions of floating structures due to disturbances from waves, wind and currents are of crucial importance.

Simple computer calculations have proved valuable at the preliminary design stage of an installation proceduce, but for a detailed analysis scale model tests are required in order to obtain accurate data. However, as the number of parameters to be varied usually becomes very large it is expensive to cover all possibilities with scale model tests. Quite often changes are also necessary at a late stage of the project, thus requiring complementary model tests.

A much more flexible and efficient way to progress is to combine basic scale model tests with computer simulations. This hybrid model tests and computer simulations approach has proved to give accurate results at a low cost. It is, however, important that the simulation models are developed based on the recorded data from the scale model tests. An efficient way to perform this is by system identification techniques.

1.1 System identification

There are in principle two ways to determine a mathematical model. The first way is from physical laws and the second one by experimentation. Mathematical model-building based on data from experiments is called system identification. Many different identification techniques, for example least squares, output error, maximum likelihood and general prediction error methods, have been suggested [1].

There are several successful applications of system identification to marine systems, especially to the ship manoeuvring dynamics [2]-[8]. Determination of the dynamics of wave-induced motions of semi-submersible platforms by the use of system identification applied to data from scale model tests is described in [9]-[12].

1.2 The Conoco Hutton tension leg platform

The installation of the Conoco Hutton tension leg platform (TLP) is chosen as one of the test cases.

In the summer of 1984 the world's first production tension leg platform was successfully installed in the Hutton Field in the North Sea for the field operators Conoco (UK) Ltd.

The TLP was towed out to the Hutton Field in a completed state. The principle remaining activities before first oil production were installation of the sixteen tethers and then tying back the pre-drilled wells to be platform and connecting the sales riser.

Heerema Offshore Contractors were responsible for holding the TLP on location above the foundation templates while Conoco installed the steel tethers. This work consisted of lowering the tethers in sections until they entered the guide cones and then subsequently latching them into the templates. To enable this to be done it was necessary to hold the TLP in position to within certain tolerances for the different phases of the operation. A photograph from the installation procedure is shown in Fig 1. An overall view of the TLP with key dimensions is shown in Fig 2. The semi-submersible assisted mooring system used at the installation is shown in Fig 3. The towing tugs remained connected throughout the operation.

1.3 The semi-submersible rig GVA 4000

The four-column semi-submersible rig GVA 4000, which is chosen as another test example, was designed and constructed by Götaverken Arendal AB (GVA), Göteborg, Sweden. A co-ordinate system xyz fixed to the rig is defined in Fig 4. Origin is at the centre of the rig and 0.45 m below the calm water level at operational draught. The particulars of the rig are summarized in Table 1.

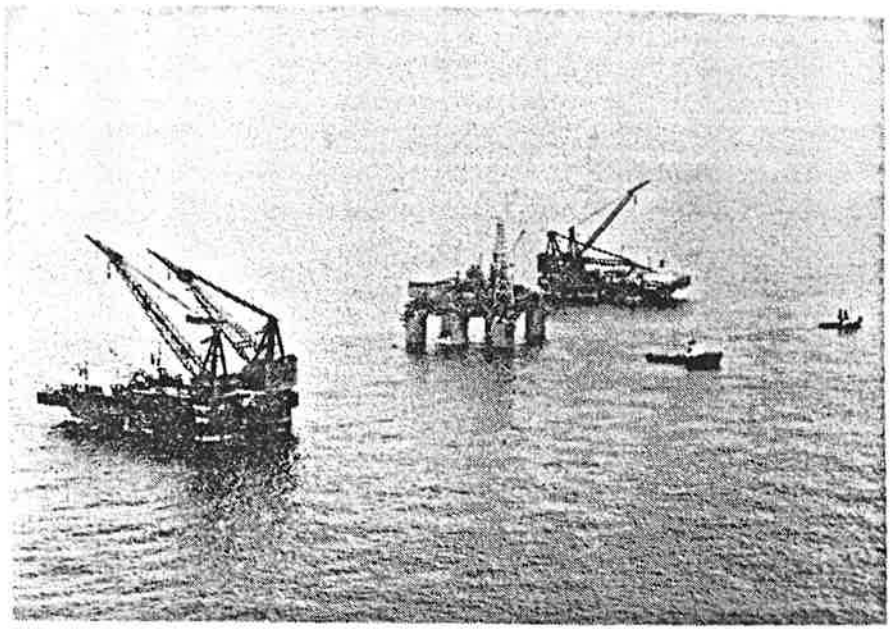
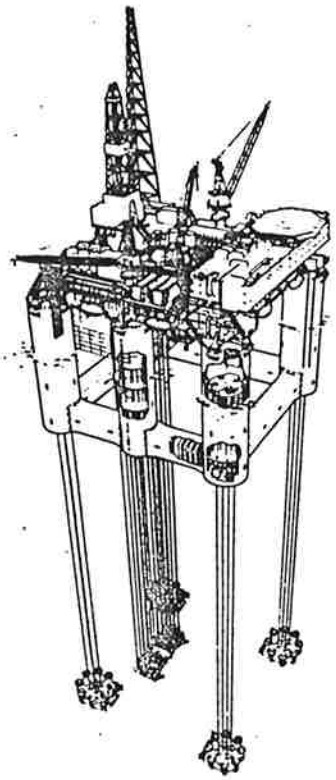


Fig 1 Installation of the Conoco TLP in the Hutton Field using Heerema's semi-submersible crane vessels 'Balder' and 'Hermod'



GEOMETRY (All dimension to moulded lines)		
LENGTH	- Between column centres	76.00 M
	- Overall	86.70 M
BREADTH	- Between column centres	74.00 M
	- Overall	81.70 M
HEIGHT	- Keel to main deck	67.70 M
	- Main deck to weather deck	11.25 M
DRAUGHT	- Operating	32.00 M at L.A.T
FREEBOARD	- To underside of main deck	24.50 M at L.A.T
WATER PLANE	- Area	7224.00 M^2
COLUMNS	- 6 Corners	17.70 M Dia.
	- 1 Centre	14.50 M Dia.
PONTOONS	Height	10.80 M
	Width	8.00 M
	Corner radius	1.50 M
DISPLACEMENT	- Approx.	61850 Tonnes
TOTAL WEIGHT	- Including mean tension (Approx)	46800 Tonnes

Fig 2 Overall view of the Conoco TLP with key dimensions

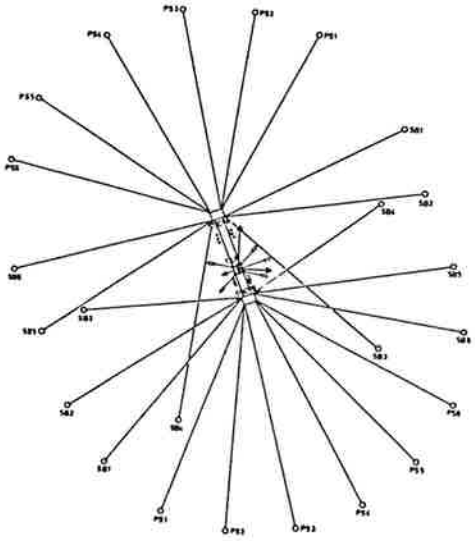


Fig 3 The Semi-Submersible Assisted Mooring System (SAMS)

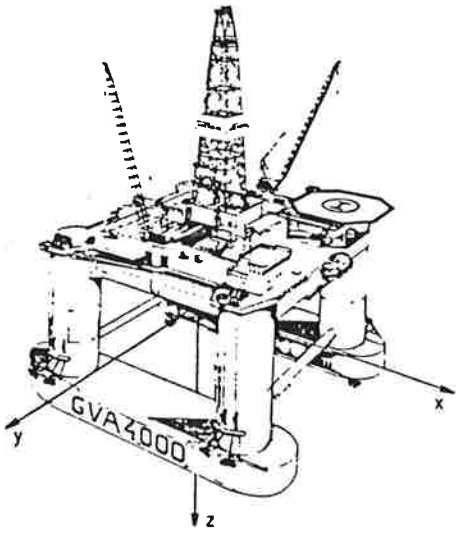


Fig 4 Definition of co-ordinate system fixed to the rig. Translations along the co-ordinate axes are called surge, sway and heave and rotations about these axes are called roll, pitch and yaw, respectively

Table 1
Particulars of GVA 4000

Length over all	(L)	85.6 m
Beam over all	(B)	76.8 m
Operational draught	(T)	20.5 m
Operational displacement	(∇)	25 310 m^3
Metacentric height	(GM)	2.4 m
Radius of gyration around z-axis	(k_{zz})	31.2 m

1.4 Organization of paper

Scale model tests appropriate for system identification are discussed in Section 2. Mathematical model-building for computer simulations based on a combination of physical knowledge and system identification applied to test data is described in Section 3. An example of the use of the mathematical model of GVA 4000 is given in Section 4, where a chain breakage in the mooring system is simulated. Statistical predictions of maximum motions and line tensions, based on simulations, are discussed in Section 5 with the TLP as an example. Procedure development and training for the TLP installation, using real time simulations, are described in Section 6. Finally, some conclusions are given in Section 7.

2. SCALE MODEL TESTS

It is common to include all environmental disturbances from wind, waves and currents at the same time in a test series for an offshore installation. However, if the recorded data are to be used for mathematical model-building it can be very difficult to distinguish between the different effects on the motions and line tensions. Thus it is recommended to perform different types of basic model tests, where the different motions and environmental forces are investigated separately, as described further on in this section. However, for calibration of the total mathematical model complete scale model tests with all environmental disturbances can be very useful.

2.1 Decay tests

Decay tests with moored structures are useful for determination of added mass and damping coefficients for the low-frequency motions. The structure is given an initial displacement in surge, sway or heave or an initial roll, pitch or yaw angle and the motion is recorded. An example is shown in Fig 5, where the TLP was given an initial surge displacement of -17.3 m. Notice that in this paper the full scale units are always presented.

2.2 Wave tests

The behaviour in waves is the most difficult part of the dynamics of a floating structure. Scale model tests in irregular seas according

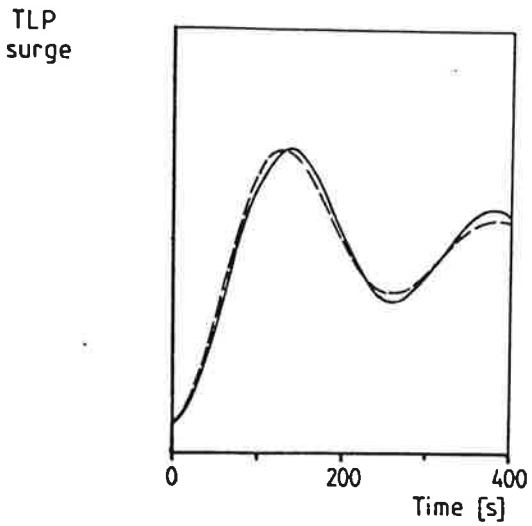


Fig 5 The TLP surge motion with the semi-submersible system after an initial displacement of -17.3 m. The continuous line is scale model test results and the dashed line is from the simulation model

to a given wave spectrum or a pink noise spectrum are very useful. By regarding the wave profile as input and the different motions as outputs system identification can be applied, resulting in a mathematical model of the first-order wave-induced motions. Second-order wave drift forces can also be determined from such tests. Tests in regular waves can also be used for mathematical model building, but such experiments are not as valuable as irregular wave tests.

The model testing of the mooring system for the TLP in waves is illustrated in Fig 6. The tests were carried out at SSPA in the Maritime Dynamics Laboratory (MDL). A description of the MDL is given in [13]. A survey of the model tests with the TLP is given in [14].

Extensive wave tests with GVA 4000 have also been carried out at SSPA [15]. The mooring system was symmetrical with respect to both the x-axis and the y-axis, as shown in Fig 7. Each chain was 900 m long and had a non-linear characteristic with a pretension of 1 275 kN (see Fig 8). Wave height, surge, sway, yaw motions and mooring forces were recorded at a sampling period T_s corresponding to 0.64 s in full scale. Some of the measured signals are shown in Fig 9. Notice that the motions are characterized by one high-frequency wave-induced part and one low-frequency part determined by the mass of the rig and the stiffness of the mooring system. The significant wave height H_s was 5.7 m and the mean wave period T_z was 8.5 s (bow quartering seas).

2.3 Hydrodynamic and current tests

Mathematical models of the low-frequency dynamics of ships are commonly determined by captive scale model tests, where hydrodynamic forces and moments are measured for different types of motions. Such tests are useful also for offshore structures, although they are rarely carried out.

However, scale model tests for determination of the current forces and moments are common. The results are easily incorporated in simulation models [9], [10]. It should, however, be pointed out that the Reynolds number effects can be of crucial importance when performing such tests [16].

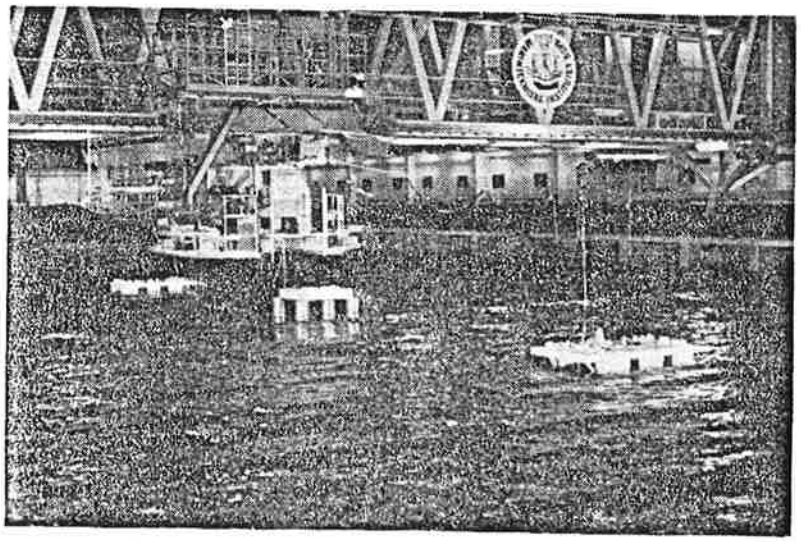


Fig 6 Scale model testing of the semi-submersible system (SAMS) at SSPA in the MDL

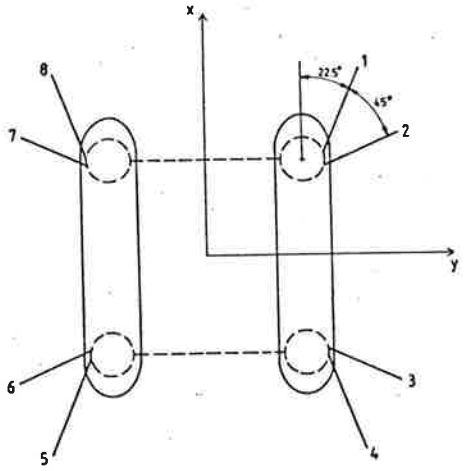


Fig 7 The eight-chain mooring system for GVA 4000

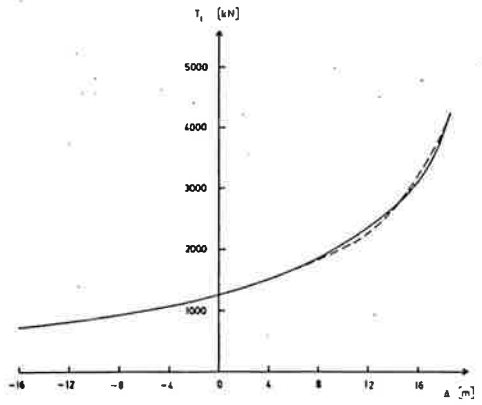


Fig 8 Line tension at the rig versus displacement. The continuous line is the characteristics used in the scale model tests and the dashed line the approximation used in the simulations

2.4 Wind tests

Wind forces and moments measured in wind tunnel, or cavitation tunnel, tests can also be directly incorporated in simulation models [9], [10]. It is also important at this kind of tests to consider the Reynolds number effects.

2.5 Special tests

Dependent on the type of installation to be analysed special scale model tests are sometimes required. The tether tip motions for the TLP installation are an example. By measuring the corner motions of the TLP and the corresponding tether tip motions it was possible to determine a mathematical model by system identification [11].

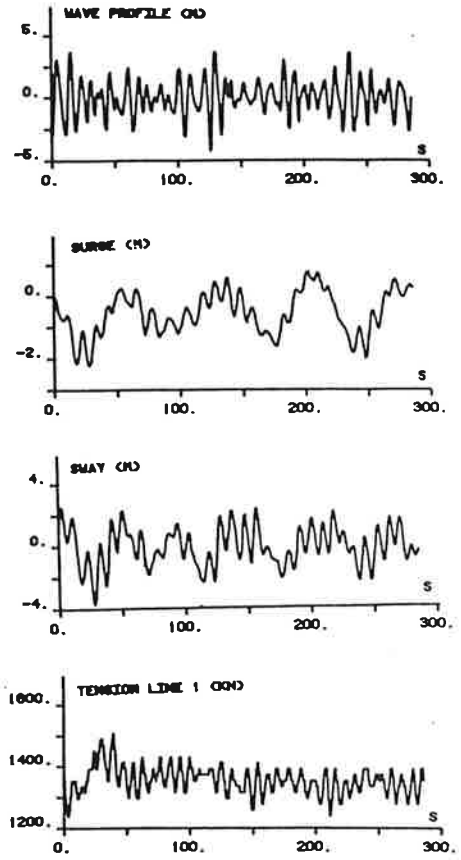


Fig 9 Measurements (in full scale units) from scale model tests with GVA 4000

For floating structures equipeed with thruster units another type of tests is important. In order to obtain relevant information about the effective force obtained from a thruster unit for varying thrust angles the thruster/hull and the thruster/thruster interaction effects should be determined. This can be done by special scale model tests [9], [10], [17], [18].

3. MATHEMATICAL MODELLING

Provided that the mooring system is not too stiff, it is a reasonable assumption that the first-order wave-induced motions can be separated from the low-frequency motions. Mathematical models of the wave-induced motions can preferably be obtained by system identification. Shaping filters for generation of irregular waves in simulations can be determined in the same way.

3.1 Waves

For the generation of irregular waves in computer simulations a shaping filter can be determined from wave measurements. The parameters of the autoregressive moving average (ARMA) model

$$\zeta(t) + a_1\zeta(t-1) + \ldots + a_n\zeta(t-n) = \lambda[e(t) + c_1e(t-1) + \ldots + c_ne(t-n)] \quad (1)$$

are estimated with the maximum likelihood method using the program IDPAC [19]. The wave elevation is denoted ζ and e is white noise in Eq (1).

The wave measurements of Fig 9 were used as an example. The order n of (1) was varied and it was found that n = 6 was appropriate. This resulted in the following parameter values:

$$\begin{aligned} a_1 &= -3.51 & c_1 &= -1.76 \\ a_2 &= 4.61 & c_2 &= 0.24 \\ a_3 &= -2.17 & c_3 &= 0.70 \\ a_4 &= -0.68 & c_4 &= -0.03 \\ a_5 &= 1.07 & c_5 &= -0.10 \\ a_6 &= -0.32 & c_6 &= -0.04 \end{aligned} \quad (2)$$

$$\lambda = 0.22$$

3.2 First-order wave-induced motions

For the determination of the wave-induced dynamics it is necessary to filter the total motions with an appropriate high-pass filter (cf Fig 9). Fig 10 shows the results obtained with a filter of the order four and a cut-off frequency of 0.04 Hz using the GVA 4000 data. The filtered data were used to determine models

$$y(t) + a_1y(t-1) + \ldots + a_ny(t-n) = b_1\zeta(t-1) + \ldots + b_n\zeta(t-n) \quad (3)$$

by system identification with IDPAC. The least squares method was applied and the model order n was varied between one and eight. The results of the surge and sway motions for n = 4 and n = 8 are shown in Fig 11. It was decided to use n = 8 based on a statistical test [12].

The same approach as described above was used for the TLP installation. Models of the structure (3) were determined for the surge, sway and yaw motions of the TLP, SSCVs and tugboats. The order n was chosen to be four, resulting in a deviation between simulations and scale model tests usually less than 5% with respect to standard deviations, see [11].

The described identification procedure has the attractive feature that the accuracy of the resulting mathematical model can be controlled by the order n. However, the order should not be chosen too large, so as to avoid unnecessarily long execution times.

3.3 Low-frequency motions

The total motions of floating structures are obtained by superposition of high-frequency wave-induced motions described in Section 3.2 and low-frequency motions. A suitable model for the low-frequency surge, sway and yaw motions is described in [12].

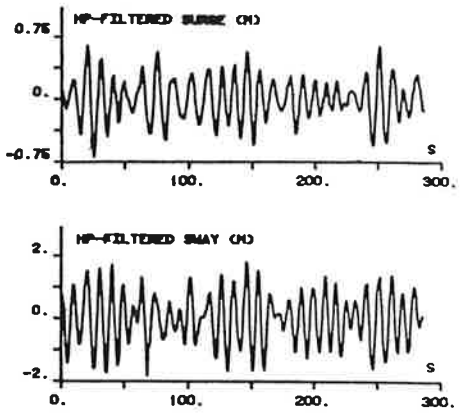


Fig 10 The results of high-pass filtering (order 4, cut-off frequency 0.04 Hz) of surge and sway motions for the GVA 4000 tests

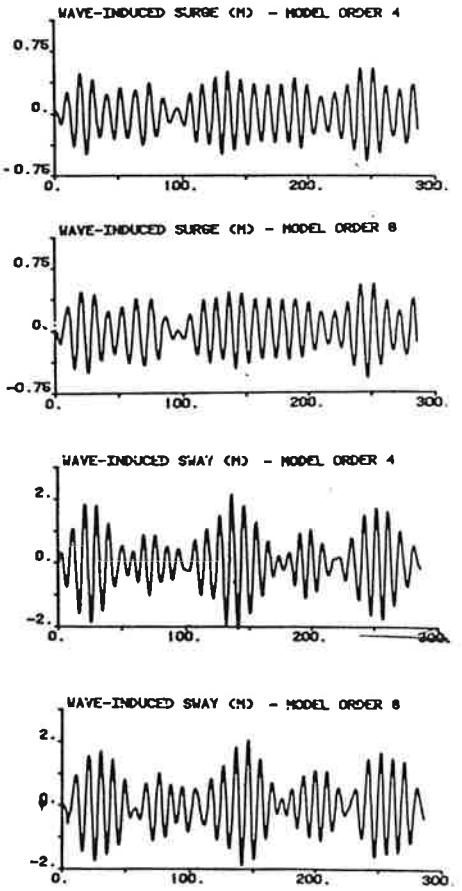


Fig 11 Results of identifications of 4th and 8th order models to the GVA 4000 data (cf Fig 10)

The forces from the wave drift, wind and currents are added to the low-frequency model. The wind turbulence can also be modelled by feeding white noise through an appropriate filter [9], [10], [11].

3.4 Mooring forces

Each mooring line of a system is individually modelled by a non-linear expression

$$T_1 = f(d-d_0) + s_0 \qquad (4)$$

where T_1 is the line force and d is the distance between the two end points of the line. The pretension $T_1 = s_0$ is obtained when the distance d is equal to the nominal distance d_0. Different non-linear functional expressions for f in (4) can be used.

The tension T_1 [kN] of each mooring chain for GVA 4000 (see Fig 7) is given by:

$$\begin{aligned} T_1 &= 0 && \Delta < -43 \\ T_1 &= 26.7\Delta + 1148 && -43 \leq \Delta < -10 \\ T_1 &= 1.73\Delta^2 + 56.7\Delta + 1275 && -10 \leq \Delta < 10 \\ T_1 &= 21.8\Delta^2 - 360\Delta + 3440 && 10 \leq \Delta \end{aligned} \qquad (5)$$

where Δ [m] is the displacement, including both high and low frequency motions, from the nominal position. Note that the pretension 1 275 kN is obtained for $\Delta = 0$, see Fig 8.

The total mooring forces are added to the low-frequency models in the same way as the disturbance forces. If a mooring system is stiff it is also necessary to consider the line forces in the first-order wave-induced dynamics. An example of this is the sway direction of the semi-submersible assisted mooring system for the TLP [11].

3.5 Validation of mathematical models

The purpose of model validation is to find out if a mathematical model is reasonable and if simulation results are consistent with scale model tests. In order to produce reliable results from the computer simulations the model validation must be carried out carefully.

Simple decay tests should be simulated and compared with scale model experiments, see Fig 5. Irregular wave tests should also be simulated and both the first-order motions and the total motions as well as the line forces should be compared with scale model experiments, as described in [12].

4. SIMULATION OF A CHAIN BREAKAGE

When a mathematical model has been developed and calibrated it can be used for analysis of motions and line forces, for example. The behaviour under different weather conditions and for alternative mooring systems is easy to evaluate. Casualties, such as chain breaks, can also be investigated in a safe way.

An example is shown in Fig 12, where a chain breakage of line No 1 for GVA 4000 is simulated after 70 s, cf Fig 7. Irregular waves corresponding to a significant wave height H_s of 8 m (bow quartering) were assumed. The current was 1 m/s towards south, but the wind influence was neglected in the simulation. Normally a mooring chain will not break at such a low tension as 2 000 kN, unless the fatigue strength has been decreased due to severe mo-

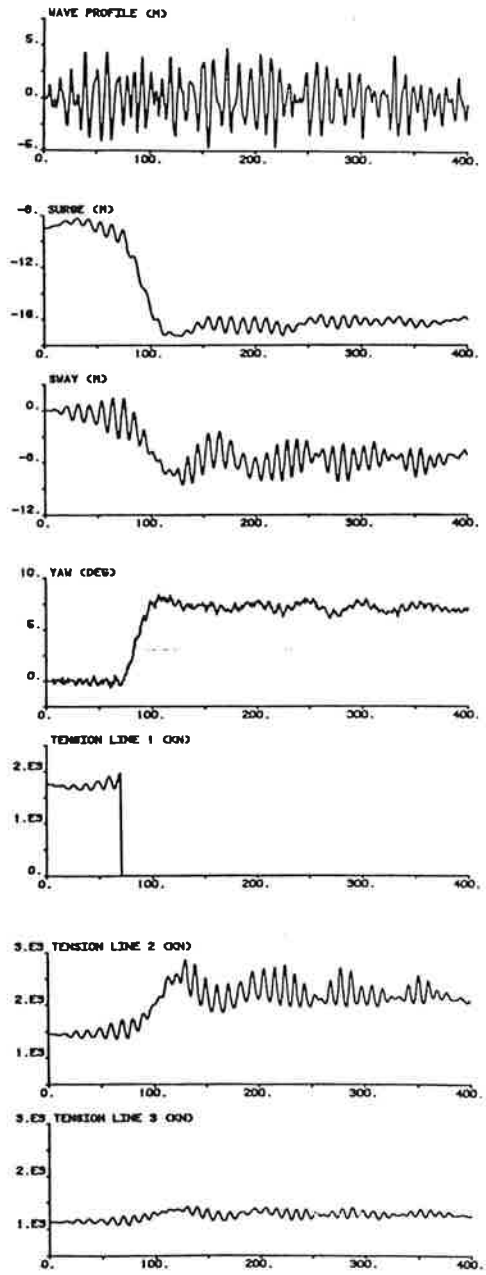


Fig 12 Simulation of a break of chain No 1 for GVA 4000 after 70 s

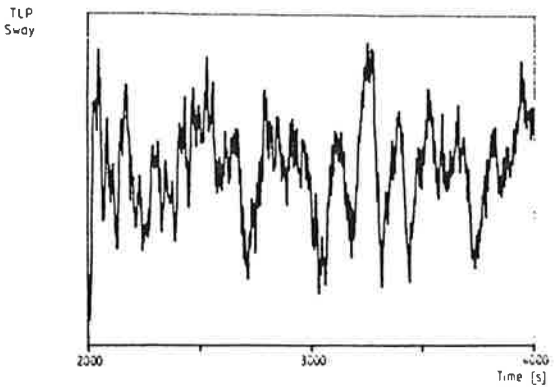


Fig 13 Simulated TLP sway motion with the semi-submersible system in irregular waves, wind and current

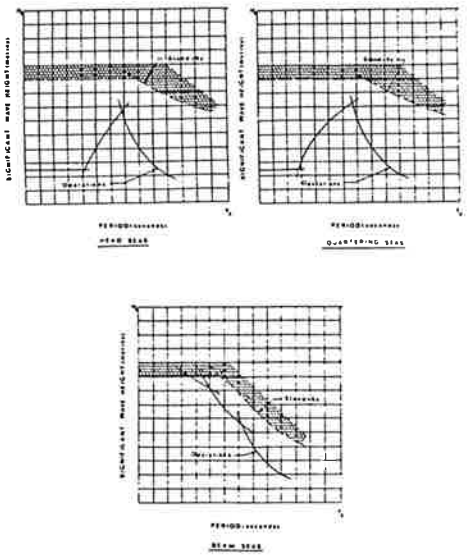


Fig 14 SAMS environmental limits derived from simulations/predictions (adopted from [14])

tions during a long period. The simulation was carried out efficiently with the interactive simulation program SIMNON [20].

5. STATISTICAL PREDICTIONS

The mathematical model of the semi-submersible system for the TLP installation was also implemented in the simulation program SIMNON. Accelerated time simulations were performed and the resulting data formed the basis for statistical predictions of maximum values of line tensions and motions. An example of a simulation with irregular waves, wind and current is shown in Fig 13.

The prediction algorithm was based on the well-established Longuet-Higgins method [21] of statistical maxima and allows the maximum values for line tensions and motions to be computed. The predictions enabled the design contractor to evaluate weather windows so the installation of the TLP could be efficiently planned.

The maximum value of a stochastic process ζ during the time T with risk, or probability, of exceedance α (α = 0.2 in the TLP project) is given by

$$M_{T,\alpha} = \bar{\zeta} + \sqrt{2 m_0 (\ln N + \beta)}$$

$$\beta = \ln \frac{1}{\ln(\frac{1}{1-\alpha})} \qquad (6)$$

$$N = T \cdot \sqrt{\frac{m_2}{m_0}}$$

where the mean value $\bar{\zeta}$ and the moments m_0 (= the variance) and m_2 are computed from the simulations. Prediction intervals T of 15, 30, 600 and 2 640 minutes, which were related directly to the tether installation activities, were used. A sample of operating environmental limits, derived from simulations/predictions, is shown in Fig 14.

6. REAL TIME SIMULATIONS

Mathematical models can also be used in real time simulations for procedure developments, operator training and testing of equipment to be installed onboard.

The mathematical model for the TLP installation was implemented in a real time version of the program SIMNON at the SSPA real time simulator. The simulation model was run in a real time mode for two different purposes. The first was development of detailed procedures for the manoeuvring of the TLP and holding it on position. The second was for training of the key installation personnel, who would be operating the mooring system during the TLP tether running. A photograph of the procedure development at SSPA is shown in Fig 15.

During the actual TLP installation in the summer of 1984 the recorded wave and wind conditions were very low, thus helping the smooth running of the mooring system operation and tether installation. There was general agreement between the installation personnel that the computer simulator training was of great value, particularly because of the novel use of the SSCVs and therefore the unfamiliarity of the operators with the mooring system characteristics.

7. CONCLUSIONS

The installation of the world's first commercial TLP was successfully performed using two semi-submersible crane vessels in an interactive mooring spread. A combination of model testing and computer simulations proved to be an efficient design tool for the mooring system and its components. The real time training simulator proved invaluable in enabling installation personnel to gain "hands-on" experience of the system. In conclusion, the use of model tests

Fig 15 Set-up of the SSPA real time simulator for procedure development with the semi-submersible system for the TLP installation

and computer simulations for design, training and evaluation of installation logic was effective for the Conoco Hutton TLP installation.

It is believed that the hybrid model tests and computer simulations approach is very effective for many types of offshore installations.

8. ACKNOWLEDGEMENTS

The authors wish to thank their respective companies for permission to publish this paper. The opinions and ideas expressed in the paper are, however, entirely those of the authors and do not necessarily represent the views of the companies.

The authors are also grateful to the management of Heerema and of the participating companies in the Hutton Field Development for permission to publish this paper.

Participating companies were:

Conoco (U.K.) Limited
Britoil PLC
Gulf Oil Corporation
Amerada Hess (U.K.) Limited
Amoco (U.K.) Exploration Co
Enterprise Oil PLC
Mobil North Sea Limited
Texas Eastern North Sea Inc

The scale model tests with GVA 4000 were commissioned by Götaverken Arendal AB, which are also gratefully acknowledged.

9. REFERENCES

[1] Åström, K.J. and Eykhoff, P., System Identification - A Survey, Automatica 7 (1971) 123-162

[2] Åström, K.J. and Källström, C.G., Identification of Ship Steering Dynamics, Automatica 12 (1976) 9-22

[3] Åström, K.J., Källström, C.G., Norrbin, N.H. and Byström, L., The Identification of Linear Ship Steering Dynamics using Maximum Likelihood Parameter Estimation, SSPA Publ. No 75, Göteborg, Sweden (1975)

[4] Källström, C.G., Identification and Adaptive Control Applied to Ship Steering, Thesis, Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden. CODEN:LUTFD2/(TFRT-1018)/1-192/(1979). Also SSPA Publ No 93 (1982)

[5] Källström, C.G. and Åström, K.J., Experience of System Identification Applied to Ship Steering, Automatica 17 (1981) 187-198

[6] Tiano, A. and Volta, E., Application of the Identification Technique to the Ship System, Proc. 7th IFAC Triennial World Congress, Helsinki, Finland (1978)

[7] Blanke, M., On Identification of Nonlinear Speed Equation from Full Scale Trials, Proc. 5th Ship Control Systems Symp., Vol 6, Annapolis, Maryland, USA (1978)

[8] Flobakk, T., Parameter Estimation of Ship Manoeuvring Equations, Modeling, Identification and Control 4 (1983):3, 175-194

[9] Källström, C.G., Mooring and Dynamic Positioning of a Semi-Submersible. A Computer Simulation Study, 2nd Intern. Symp. on Ocean Engineering and Ship Handling, Göteborg, Sweden (1983) 417-442

[10] Källström, C.G., A Dynamic Positioning System for Semi-Submersibles, RINA Symp. on 'Semi-Submersibles: The New Generation', London, U.K. (1983)

[11] Källström, C.G., Guy, S., Chivvis, J. and Ramzan, F.A., Computer Simulations of the Installation of the Conoco Hutton Tension Leg Platform, RINA Symp. on 'Offshore Transportation and Installation', London, U.K. (1985)

[12] Källström, C.G., System Identification Applied to Data from Scale Model Tests with a Moored Semi-Submersible, 7th IFAC/IFORS Symp. on 'Identification and System Parameter Estimation', York, U.K. (1985)

[13] van Berlekom, W.B. and Lindgren, H., The SSPA Maritime Dynamics Laboratory. Background, Experience and Future Development, Paper No 1, 1st Intern. Symp. on Ocean Engineering and Ship Handling, SSPA, Göteborg, Sweden (1980)

[14] Kypke, D., Guy, S., Byström, L. and Chivvis, J., Positioning during Installation of the Hutton TLP Vessel - A Survey of Model Tests, Offshore Göteborg 85, Göteborg, Sweden (1985)

[15] Lundgren, J. and Berg. A., Wave-Included Motions on a Four-Column Semi-Submersible Obtained from Model Tests, Offshore Technology Conference, Houston, Texas, USA. Paper OTC 4230

[16] Jacobsson, P. and Dyne, G., Reynolds Number Effects in Model Tests with a Four-Column Semi-Submersible, 2nd Intern. Symp. on Ocean Engineering and Ship Handling, Göteborg, Sweden (1983) 343-362

[17] Moberg, S. and Hellström, S.A., Dynamic Positioning of a Four-Column Semi-Submersible, 2nd Intern. Symp. on Ocean Engineering and Ship Handling, Göteborg, Sweden (1983) 443-480

[18] Allenström, B. and Källström, C.G., Power Minimization in Dynamic Positioning Systems, Offshore Göteborg 85, Göteborg, Sweden (1985)

[19] Wieslander, J., Idpac Commands - User's Guide, Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden. CODEN:LUTFD2/(TFRT-3157)/1-108/

[20] Elmqvist, H., SIMNON - An Interactive Simulation Program for Nonlinear Systems, User's Manual, Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden

[21] Longuet-Higgins, M.S., On the Statistical Distribution of the Heights of Sea Waves, Jrnl Mar. Res., Vol. 11, No 3 (1952) 245-266

SSPA
MARITIME CONSULTING
SSPA Maritime Consulting AB

SSPA Maritime Consulting AB

SSPA är ett av svenska staten helägt aktiebolag. Företaget är i sin helhet lokaliserat till ett område i nära anslutning till Chalmers Tekniska Högskola, Göteborg.
SSPA omsätter 48 milj kr och vinsten är ca 7 milj kr. Antalet anställda är ca 120 st.

SSPA:s affärsområden

SSPA är ett marknadsorienterat företag, som identifierar fyra huvud-affärsområden:

SHIPPING, dvs transportsystem till havs inkluderande t ex handelsfartyg, hamnar och kanaler.

NAVAL, dvs marinmilitära system inkluderande t ex örlogsfartyg, ubåtar och torpeder.

OFFSHORE, dvs system för olje- och gasutvinning till havs inkluderande t ex semi-submersibles, jackets och pråmar.

EQUIPMENT, dvs. försäljning av mikrodatorbaserade reglersystem, nyckelfärdiga träningssimulatorer och utrustning för modellförsök samt datorprogram, med speciell inriktning mot det marintekniska området.

Av SSPA:s intäkter kommer ca 50% från utlandet. Marknaden finns dels i Europa och dels i ett antal utom-europeiska länder.

SSPA:s verksamhet

Basen för verksamheten är SSPA:s anläggningar för modellprovning av fartyg, offshore-konstruktioner, torpeder, ubåtar, etc. Följande anläggningar finns:

* Kavitationstunnel
* Släpränna
* Våg- och manöverbassäng

Förutom modellprovningar erbjuder SSPA också konsulttjänster och datorberäkningar inom det marintekniska området. Som exempel kan nämnas design av skrovformer och propellrar, strömningstekniska beräkningar, datorsimuleringar och design av reglersystem.

☞

Den datormjukvara, som SSPA har utvecklat, används dels internt men den säljs också till SSPA:s kunder i form av kundanpassade programsystem. Datorsimuleringar av rörelserna hos marina konstruktioner är en viktig del av SSPA:s verksamhet. Som en följd av denna verksamhet har två olika träningssimulatorer utvecklats och tillverkats vid SSPA:

* Manöversimulator för fartygshantering

* Simulator för ballasthantering av semi-submersibles

SSPA åtar sig att tillverka och leverera nyckelfärdiga träningssimulatorer enl kundens specifikationer. SSPA erbjuder också sina tjänster vid design av nya modellprovningsanläggningar i olika delar av världen.

En relativt ny verksamhet vid SSPA är design och tillverkning av mikrodatorbaserade reglersystem. Denna verksamhet, som bygger på mikrodatorn Motorola 68000, syftar till försäljning av kompletta, specialiserade system i korta serier.

SSPA bedriver också egen FOU verksamhet omfattande ca 10% av omsättningen.

SSPA:s organisation

I SSPA:s styrelse ingår representanter för bl.a svenska staten, varv, rederier och FMV. Styrelsens ordförande är Alf Åkerman. SSPA:s verkställande direktör är Hans Peder Loid.

SSPA Maritime Consulting AB

Box 24001

400 22 GÖTEBORG

Tel: 031 - 63 95 00

Kontaktpersoner: Hans Peder Loid eller Claes Källström.

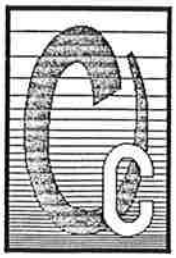
Optimus Control

Konsultföretag för
datorteknik i processtillämpningar

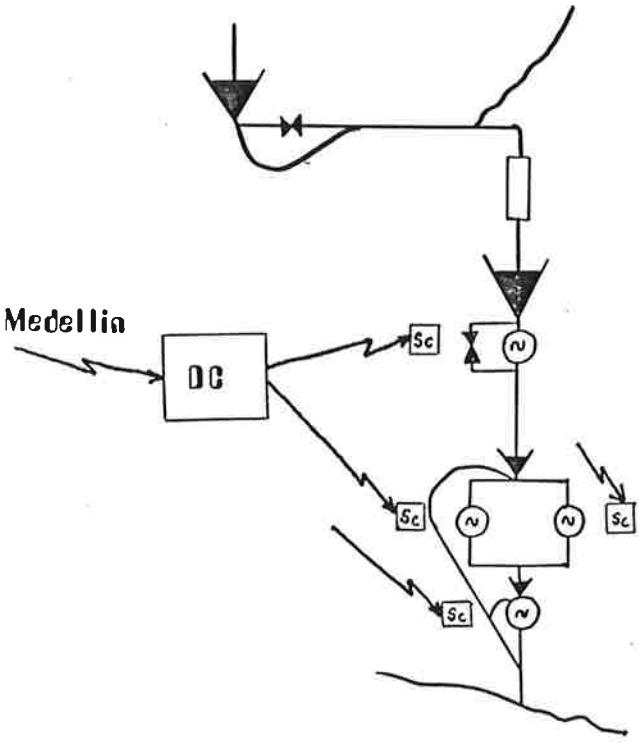
* Vattenkraftprojekt Colombia

* Reglerfrågor

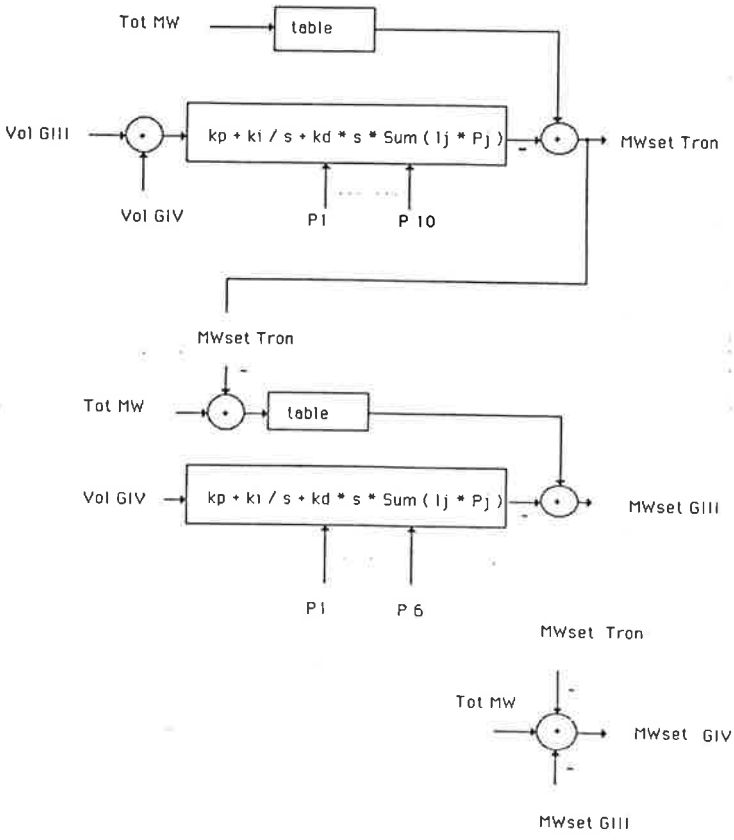
* Simuleringsstudier



Guadelupe

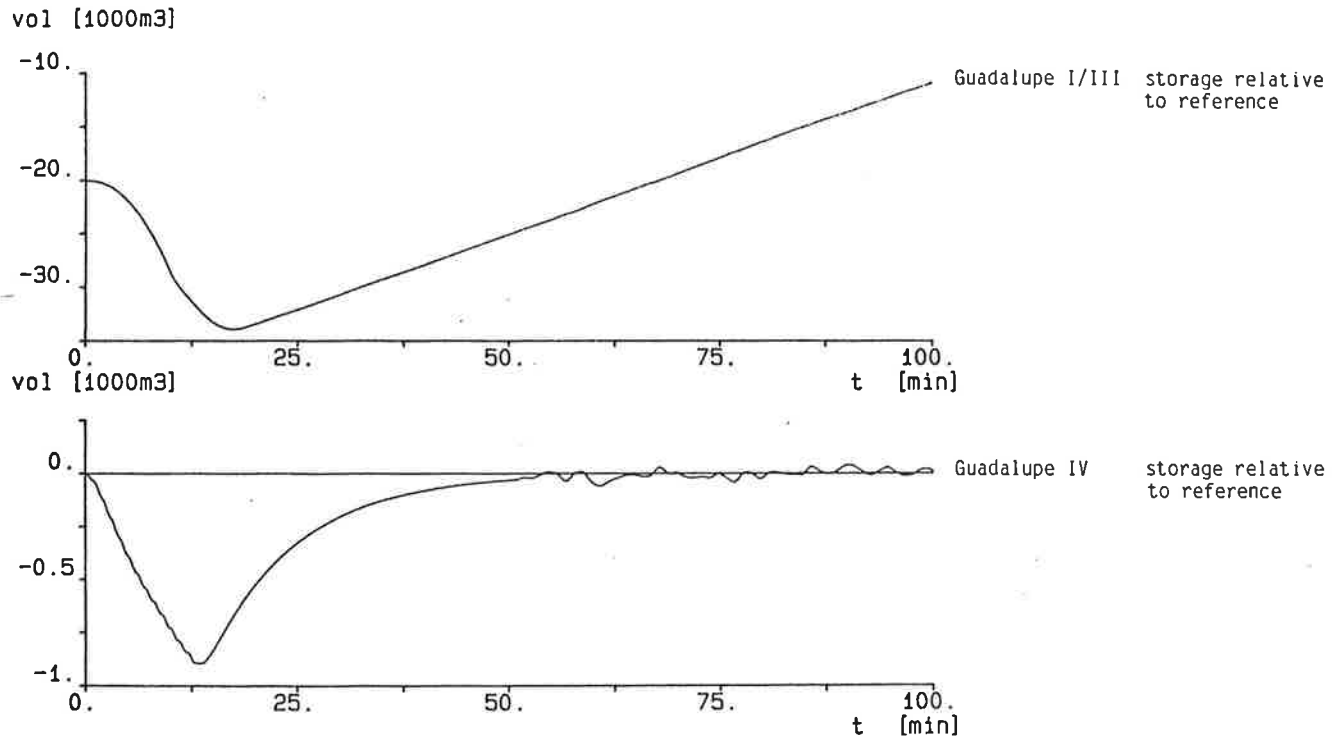


Cascade Regulator of Guadalupe



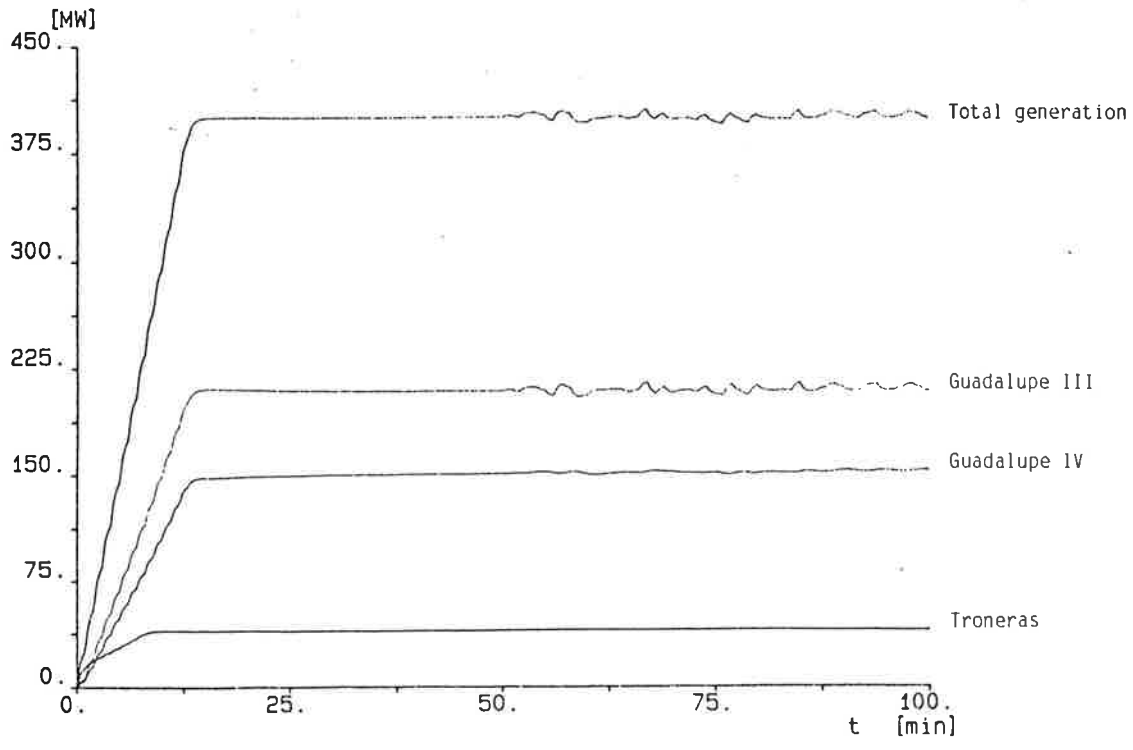
Submode A Operational Conditions
Refer to section 4.3.3

FIG: 3h
Submode A: noise in GIII generation



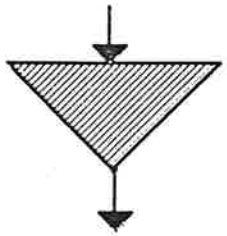
Submode A Operational Conditions
Refer to section 4.3.3

FIG: 3f
Submode A : noise in GIII generation



Models

Reservoir



$1 / S$

Volume = 1/s *netflows

Level = a0 + a1*vol + a2*vol**2 + a3*vol**3

Plant

$1 / (1 + sT)$

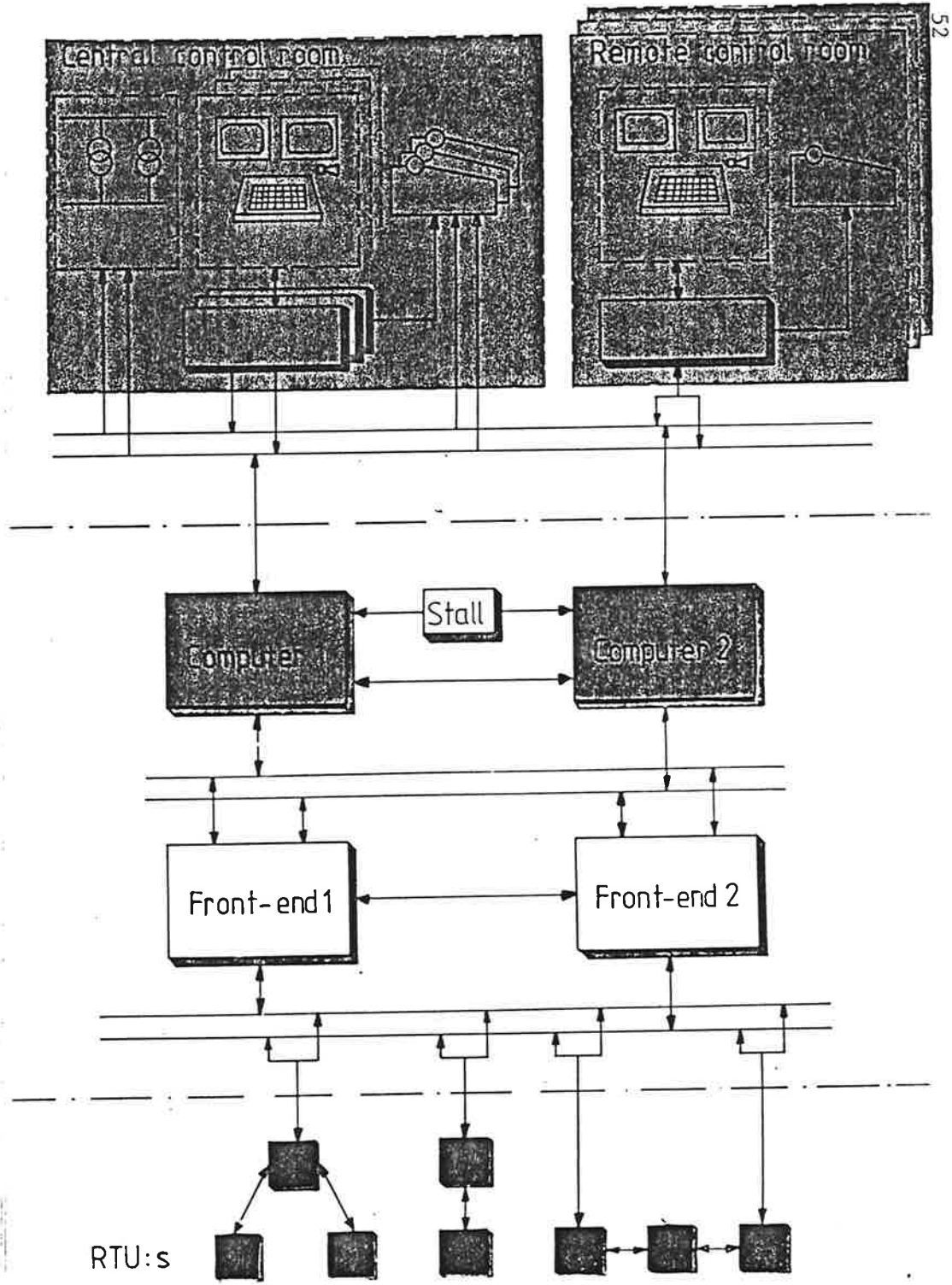
MW = 1/ (1 +Ts) MWset

Turbine flow = b0 + b1*MW

River flows

e^{-sTd}

Flow2 = Flow1 * e ** (-sTd)



COMPUTER AIDED
CONTROL ENGINEERING
(CACE)

Research Project, 1985 – 1989

Aimed at developing a software environment
for Computer Aided Control Engineering

at
The Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

Supported by
The National Swedish Board
for Technical Development (STU)

Budget: 7.5 Million SEK

STEERING COMMITTEE

Sven Gunnar Edlund, STFI
Swedish Pulp and Paper Research Institute

Arne Otteblad, STU
National Swedish Board for Technical Development

Karl Eklund, Datema AB – ASEA Processdata AB

Claes Källström, SSPA Maritime Consulting AB

Erik Sandewall, LiTH
Department of Computer and Information Science

Gustaf Söderlind, KTH, ITM, LTH
Department of Computer Science

Karl Johan Åström, LTH
Department of Automatic Control

THE CACE PROJECT - An Overview

Sven Erik Mattsson

Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

Contents:

1. The CACE project
2. How could future CACE systems look like?
3. International efforts

TODAY'S CACE TOOLS

Today's CACE tools have proved to be useful.

However, they could be improved:

- Designed for computers with moderate computing capacity and primitive I/O-devices.
- Primitive ways of representing systems
  - For instructing the computer
  - Not frameworks for organizing ideas
  - May be missing e.g. just matrices
  - Hard to reuse models
  - Hard to describe structure and decomposition
  - No use of graphics
  - No concepts for qualitative descriptions
- Only numerical calculations, not symbolic
- Quantitative, not qualitative methods
- No means to handle incomplete information
- Poor help systems, not goal directed

KEY ELEMENTS OF THE CACE PROJECT

Inject new elements which are not available in current software for CACE.

Explore impact of new hardware technology:
- Workstations
- High performance, fast, color graphics

Explore new software technology:
- Numerics routines
- Symbolic formula manipulation
- Graphics based man-machine interaction
- Expert system techniques

Establish national and international contacts.

PILOT PROJECTS

Investigate the feasibility of various ideas and approaches:

1. Graphics based man-machine interaction
2. Expert system interfaces
3. Formula manipulation and numerics
4. High-level problem solving languages
5. Implementation languages
6. Expert control
7. Numerical solution of DAE-systems

WORKSTATIONS

Advantages:
- Single user system
- Significant computing power
- High interactivity between computer and user
- Graphics.

A reasonable configuration:
- 1 Mips machine
- 4 Mbyte memory
- 1 Mpixel display
- Multitask OS, multiwindow
- 1 Mbyte/s network transfer rate

Price: < 200 000 SEK.

Problem areas:
- Operating systems
- Protocols for integrating systems
- Graphics and windowing standards
- Companies are coming and going.

How Could Future CACE-systems Look Like?

Not closed packages

Must be extensible

Tools, toolbox, toolmachine
- modularized with well-defined interfaces
- integrated; uniform interface for a user

Modularization:
1. User interface
2. Data
3. Processing tools

THE USER INTERFACE

1. Various types of I/O devices:
   - Keyboard
   - Light pen, mouse
   - Voice
   - Alphanumeric display
   - Graphics display (mono, color)
2. Different users have different needs
   - Command dialogue
   - Menus (Pop up, pull down etc.)
   - Graphics
   - Macintosh, Smalltalk 80, Lisp environments
   - Help systems
   - Expert system interfaces

$\Rightarrow$

Separate the user interface from processing tools.

Could view the CACE-system as performing a Read-Evaluate-Print loop

The user interface should handle all communication with the user:

1. Collect the user input
   - Prompt when needed
   - Expert system interfaces
2. Convert to a textual representation
   - Logging, command procedure
3. Invoke a processing tool
4. Present results and errors messages to the user or the command procedure!

User interface: Language and environment
The objects and operations should have same semantics
but different syntax: Text, menus, graphics.

DATA

Should be common to make the tools integrated.

All information must be available for

1. The User
2. Expert system interfaces
3. Command procedures

Models etc on symbolic form from which linearized models code for simulation etc could be generated.

PROCESSING TOOLS

Must not communicate with the user.

Should not have internal states.

Could provide test functions, so the user interface could check arguments before invoking a tool.

IMPLEMENTATION LANGUAGES

TASKS

- I/O handling
- Command decoding
- Graphics
- Numerics
- Symbolic formula manipulation
- Database management
- Expert system

ASPECTS

- Exploratory vs Production programming
- Portability
- Readability
- Security
- Efficiency

LANGUAGES

- Fortran
- C
- Pascal, Ada
- Lisp, Prolog
- Smalltalk

IMPLEMENTATION LANGUAGES

CACE-systems will (at least in the near future) be implemented in several languages:
- Must reuse available software
- Use the language which is best suited

Most workstations support
- Fortran
- C
- Pascal
- Common Lisp

and allows you to mix.

Our approach:
- Use Common Lisp
  - interactive, flexible, fast prototyping
  - can use Lisp as a "command procedure language"
- Incorporate software written in other languages.

INTERNATIONAL EFFORTS

IEEE

Symposia
- CACSD'83, Boston MA, September 28–30, 1983
- CACSD'85, Santa Barbara CA, March 13–15, 1985
- CACSD'86, Arlington VA, September 24–26, 1986

Technical Committee on CACSD
- Charles Herget, LLNL, USA

Working groups
- Algorithms, AI, Benchmarks, Simulation,
- Design and Applications,
- Workstations

IFAC

Symposia
- Zürich, Switzerland, 1979
- West Lafaytte, USA, 1982
- CADCE'85, Lyngby, Denmark, 1985

Has formed a working group for standardization of CACE software.

CATALOGUING

ELCS — Extended List of Control Software
- One page summaries written by the developers
- Collected by:
  - Magnus Rimwall, ETH, Zürich
  - Charles Herget, LLNL, USA

SYCOT — SYstem and COntrol Tools
- Basic Software Library for CACSD
- Working Group on Software (WGS), Benelux

UNITED KINGDOM
SERC – Science and Research Council

CDTCE – Research Program
Computing and Design Techniques for CE
2.8M GPB, 5 years

ECSTASY
Environment for Control System Theory, Application and Synthesis
UMIST, Munro

Projects:
Cambridge, Maciejowski, data model for CACE
UMIST, Munro, MMI
Salford, Gray and King, ARGOS
Swansea, Barker, CES
Cambridge, MacFarlane, Expert system for design
Kingston Polytechnic, Denham, numerics
several, design tools and applications

UNITED KINGDOM

CEGB – Central Electric Generating Board
Long experience in CACE
Federated Suite; PMSP
Very elaborate simulation; 500–4000 states
Good, fast, robust numerical routines

Well-established contacts.
Agreement between SERC and STU.

Joint SERC/STU workshops:
Graphical front ends, UMIST, July 14–18, 1986
Expert systems, Cambridge, April 6–10, 1987

FRANCE

INRIA
Delebecque and Steer, Blaise
Gomez, AI, symbolic manipulation, nonlinear filters
in collaboration with Maryland, USA

SWITZERLAND

ETH, Rimwall, IMPACT
Interactive Program for Automatic Control
implemented in Ada

USA

General Electric:
Federated Computer-Aided Control System Design
Idpac, Simnon, CLADP, SSDP
AI for design, together with RPI and West Point

Lawrence Livermore National Laboratory, LLNL:
EAGLES – Env for Adv Graph (LLNL) Eng Sys
M language
Public software
We are β-test site
DASSL – DAE-solver, Petzold

Univ. of Tennessee:
AI for design, Birdwell
part of DOE's CASCADE project

Berkeley:
Delight, Polak

University of Maryland:
Systems Research Center
robotics, chemical processes
Symbolic manipulation, nonlinear filters
collaboration with INRIA, France

Santa Barbara:
Laub, numerics, linear algebra, Riccati equations

The CACE Project 1987-03-03
Department of Automatic Control
Lund Institute of Technology, Lund, Sweden

Published Papers, Conference Contributions and Reports

ÅRZÉN, K-E. (1986): "LISP—A One-Week Course," Report CODEN: LUTFD2/TFRT-7310, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ÅRZÉN, K-E. (1986): "Expert systems for process control," in Sriram, D. and R. Adey (Eds.): *Applications of Artificial Intelligence in Engineering Problems*, Proc. of 1st Int. Conf. on Applications of AI in Engineering Practice, Southampton University, U.K., April 1986, Springer-Verlag, pp. 1127–1138, Also available as internal report TFRT-7315.

ÅRZÉN, K-E. (1986): "Use of Expert Systems in Closed Loop Feedback Control," *Proc. ACC*, Seattle, USA, 1986, pp. 140–145, Also available as internal report TFRT-7320.

ÅRZÉN, K-E. (1986): "Kunskapsbaserade Regulatorer (Knowledge Based Controllers)," *SAIS '86*, The Swedish AI Society's Annual Workshop, Linköping, April 24–25, 1986.

ÅRZÉN, K-E. (1986): "Reserapport—AAIEP 1986," Travel Report CODEN: LUTFD2/TFRT-8044, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ÅRZÉN, K-E. and K.J. ÅSTRÖM (1987): "Knowledge-based feedback control," To appear.

ÅSTRÖM, K.J. (1985): "Computer Aided Tools for Control System Design—A perspective," in Jamshidi M. and C.J. Herget (Eds.): *Computer-Aided Control Systems Engineering*, North-Holland.

ÅSTRÖM, K.J. (1986): "Auto-Tuning, Adaptation and Smart Control," in Morari and Mc Avoy (Eds.): *Proc Chemical Process Control—CPCIII*, CACHE, Elsevier, pp. 427–466.

ÅSTRÖM, K.J, J.J. ANTON and K-E. ÅRZÉN (1986): "Expert Control," *Automatica* **22**, No. 3, 277–286.

ÅSTRÖM, K.J. and W. KREUTZER (1986): "System Representations," *Proceedings of the IEEE Control Systems Society Third Symposium on Computer-Aided Control Systems Design (CACSD)*, Arlington, Virginia, September 24–26, 1986, Also available as internal report TFRT-7330.

BRÜCK, D.M. (1986): "Implementation of Graphics for Hibliz," Report CODEN: LUTFD2/TFRT-7328, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. and S.E. MATTSSON (1986): "A Simulator for Dynamical Systems Using Graphics and Equations for Modelling," *Proceedings of the IEEE Control Systems Society Third Symposium on Computer-Aided Control Systems Design (CACSD)*, Arlington, Virginia, September 24–26, 1986.

HOLMBERG, U. (1986): "Some MACSYMA Functions for Analysis of Multivariable Linear Systems," Report CODEN: LUTFD2/TFRT-7333, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

HOLMBERG, U., M. LILJA and B. MÅRTENSSON (1986): "Integrating Different Symbolic and Numeric Tools for Linear Algebra and Linear Systems Analysis," Report CODEN: LUTFD2/TFRT-7338, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, Presented at SIAM Conference on Linear Algebra in Signals, Systems and Control, August 12–14, 1986, Boston, Massachusetts, USA.

HOLMBERG, U., M. LILJA and S.E. MATTSSON (1987): "Combination of Symbolic Manipulation and Numerics," Final Report 1987-03-31, STU project 85-4809, STU program: Computer Aided Control Engineering, CACE, Report CODEN: LUTFD2/TFRT-3186, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

KREUTZER, W. and K.J. ÅSTRÖM, "System Representations," To appear.

LARSSON, J.E. (1984): "An Expert System Interface for Idpac," Master Thesis, CODEN: LUTFD2/TFRT-5310, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

LARSSON, J.E. and K.J. ÅSTRÖM (1985): "An Expert System Interface for Idpac," Report CODEN: LUTFD2/TFRT-7308, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, Presented at CACSD '85, The 2nd IEEE Control Systems Society Symposium on Computer-Aided Control System Design (CACSD), Santa Barbara, California, March 13–15, 1985.

LARSSON, J.E. and P. PERSSON (1986): "Ett expertsystemsnitt för Idpac (An Expert System Interface for Idpac)," *SAIS '86*, The Swedish AI Society's Annual Workshop, Linköping, April 24–25, 1986.

LARSSON, J.E. and P. PERSSON (1986): "Knowledge Representation by Scripts in an Expert Interface," *Proceedings of the 1986 American Control Conference*, Seattle, June 1986, Also available as internal report TFRT-7332.

LARSSON, J.E. and P. PERSSON (1987): "An Expert System Interface for Idpac," Licentiate Thesis CODEN: LUTFD2/TFRT-3184, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

LARSSON, J.E. and P. PERSSON (1987): "An Intelligent Help System for Idpac," To appear.

LILJA M. (1986): "Some SISO Transfer Function Facilities in CTRL-C," Report CODEN: LUTFD2/TFRT-7325, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. (1985): "New Forms of Man-Machine Interaction," Status Report 1985-09-30, STU project 84-5069, STU program: Computer Aided Control Engineering, CACE, Report CODEN: LUTFD2/TFRT-7293, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. (1986): "On Differential/Algebraic Systems," Report CODEN: LUTFD2/TFRT-7327, Department of Automatic Control, Lund

Institute of Technology, Lund, Sweden.

MATTSSON, S.E. and K.J. ÅSTRÖM (1986): "The CACE Project— Steering Committee Meeting 2," Report CODEN: LUTFD2/TFRT-7321, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. and K.J. ÅSTRÖM (1986): "The CACE Project— Steering Committee Meeting 3," Report CODEN: LUTFD2/TFRT-7322, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. and K.J. ÅSTRÖM (1987): "The CACE Project— Steering Committee Meeting 4," Report CODEN: LUTFD2/TFRT-7343, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E., K.J. ÅSTRÖM, D.M. BRÜCK and T. SCHÖNTHAL (1987): "A Trip to the University College Swansea and the Central Electric Generating Board, Gloucester," Travel Report CODEN: LUTFD2/TFRT-8045, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E., H. ELMQVIST and D.M. BRÜCK (1986): "New Forms of Man-Machine Interaction," Final Report 1986-09-30, STU project 84-5069, STU program: Computer Aided Control Engineering, CACE, Report CODEN: LUTFD2/TFRT-3181, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MÅRTENSSON, B. (1986): "CODEGEN—Automatic Simnon Code Generator for Multivariable Linear Systems," Report CODEN: LUTFD2/TFRT-7323, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MÅRTENSSON, B. (1986): "Automatic TEX Code Generation from Macsyma and CTRL-C," Report CODEN: LUTFD2/TFRT-7334, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MÅRTENSSON, B. (1986): "Hcopy2PS-Hcopy to PostScript filter," Report CODEN: LUTFD2/TFRT-7335, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MÅRTENSSON, B. (1986): "Experiences With Computerized Document Preparation Tools," Report CODEN: LUTFD2/TFRT-7336, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

NILSSON, B. (1986): "LQG-I/O—A CTRL-C library for LQG design," Report CODEN: LUTFD2/TFRT-7329, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

International CACE Software Efforts

Sven Erik Mattsson
Karl Johan Åström

Department of Automatic Control
Lund Institute of Technology
March 1987

1. Introduction

As stated in the general program for the Swedish, STU supported CACE project, an important part of the project is to set up international contacts. The purpose of this report is to identify and describe some foreign CACE software efforts, which are of special interest for the Swedish CACE project. These projects may give us inspiration and possibilities to exchange ideas, experiences and software. These contacts also help us avoiding redoing things and mistakes already done by others. Special stress has been laid on the computer scientific parts like man-machine interaction, numerics, high level language design etc., whereas more traditional automatic control like design of algorithms almost has been neglected.

Many packages for computer aided modelling, analysis and design of control systems have been developed and more are to come. Every university with education and research in automatic control seems to have developed software in this area. The software is often the outcome of MS thesis projects. The quality of the packages varies. As stated above it is not our purpose to give a list of control software. This task has been undertaken by other groups and is today coordinated into the Extended List of Control Software (ELCS). ELCS is a collection of one-page summaries on computer-aided control system design packages and related software libraries. The summaries are written by the software developers. In the US they are collected by Dr. Charles J. Herget, Lawrence Livermore National Laboratory, Livermore, California and elsewhere by Magnus Rimwall, Dept. of Automatic Control, ETH, Zürich, Switzerland. The second issue of ELCS (ELCS, 1986) contains more than 80 software descriptions.

International organizations like IEEE and IFAC play also an important role. They organize conferences, symposia, workshops etc. which make it possible to find and establish new contacts as well as in an easy way maintain old ones. The IEEE Control Systems Society (CSS) has arranged three symposia on computer-aided control system design: CACSD'83 in Boston, September 28–30, 1983, CACSD'85 in Santa Barbara, California, March 13–15, 1985 and CACSD'86 in Arlington, Virginia, September 24–26, 1986. The plan is to arrange a symposium each 18th month. In June 1981 IEEE CSS formed the Technical Committee on Computer-Aided Control Systems Design. It is chaired by Dr. Charles Herget. In December 1984 IEEE CSS also formed six working groups: Algorithms, AI, Benchmarks, Design and Applications, Simulation and Workstations (IEEE CSS, 1985). The working group for benchmark problem has started with preparing problems for evaluating the analytic capabilities of packages (Frederick and Rimer, 1985, 1986; IEEE CSS, 1985; Hawley and Stevens, 1986) and will continue with procedures for evaluating the computer-aided engineering capabilities.

IFAC has organized three symposia on computer aided design in control and engineering systems. The first one was held in Zürich, Switzerland, 1979 and the second one in West Lafayette, 1982. The third one CADCE '85 was held in Lyngby, Denmark, July 31–August 2, 1985. One session there was devoted to standardization of software for CACSD. One outcome of the discussion was to form a working group. The first meeting was held in conjuncture with CACSD'86 and the next will be held during the IFAC World Congress in Munich, FRG, July 26–31, 1987.

2. Projects

In this section we will consider a number of CACE projects in different countries and our relation to them.

United Kingdom

Current work in computer aided design of control systems in the United Kingdom is surveyed by Jacobs (1985).

Over the past decade UK control engineering developments have been concentrated on research into the fundamental theory of multivariable systems and the development of new control design techniques for such systems. This work has resulted in such design techniques as the complex variable design methods (MacFarlane, Cambridge), the inverse Nyquist array design method (Rosenbrock, UMIST), optimization techniques (Mayne, Imperial College). Many of these developments have been embodied in CAD packages.

Unfortunately, this work does not fulfil the desires and demands from aerospace, marine and production industries. The design methods are not suitable in their present form for systems with widely varying parameters or significant nonlinearities. Furthermore, the software has been produced in a fragmented and an unstandardized manner. To remedy these weak points the Science and Engineering Research Council (SERC) promotes a special programme of research in Computing and Design Techniques for Control Engineering, CDTCE (SERC, 1986). It was started in 1984 and the budget over the first 5 year period is 2.8M GBP. Groups participate from amongst others the University of Manchester (UMIST), Cambridge University, Salford University, University College of Swansea, National Physics Laboratory and Imperial College.

Professor Munro, UMIST has got a contract to specify and develop an Environment for Control System Theory, Application and Synthesis (ECSTASY). The specification was frozen in the autumn 1986 and the implementation is based on a software environment developed by PA Consulting Services, Computer Aided Engineering, Cambridge Laboratory.

A basic idea is to produce a tool kit of general purpose modules and to avoid to build large software packages. To be able to do that without a lot of interfaces a common data model must be used. Maciejowski (1984, 1985a,b) at Cambridge has got a grant to develop a data model which could be accepted as a standard by the control community. At UMIST they also have projects dealing with the man-machine interface (Goodfellow and Munro 1985a, b, c; Munro et al, 1986). There are two major projects dealing with man-machine interfaces and interactive graphics (block diagrams): ARGOS (A Real time Graphics Operating System) at the University of Salford (King and Gray, 1985) which a general purpose graphics based man-machine interface, and CES (Control Engineering (work)Station) at the University College of Swansea (Barker et al, 1986, 1987). Work on a Control Engineering Subroutine Library has been centered at Kingston Polytechnic. Professor MacFarlane at Cambridge University runs a project aimed at developing expert systems for design and analysis of control and dynamic systems (Pang et al, 1986). The CDTCE research programme contains also a number of projects for developing design tools (SERC, 1986).

The contacts with the research groups participating in the CDTCE research programme are well-established both formally and informally. A framework of collaboration has been agreed between STU and SERC representatives.

A joint SERC/STU workshop on the subject of graphical front ends for CACE was held at UMIST, July 14–18, 1986. A joint workshop on use of expert system and knowledge based engineering in CACE will be held at Cambridge in April 1987. We are also visiting each other and we are exchanging information and software.

We have also established relations with a group at the Central Electric Generating Board, CEGB. CEGB has a long experience in designing and using programs for CACE. An overview of the CEGB federated suite is given in Whitmarsh-Everiss (1986). Very elaborate simulation of power system components is a speciality; 500–4000 states is a typical size range. To be able to do that, good, fast and robust numerical routines for integrating large systems of stiff equations have been developed. For references see the travel report Mattsson et al (1987).

Benelux

In March 1983 a working group named "Working Group on Software (WGS)" with members from the academic society and industry in the Netherlands and Belgium was formed (WGS, 1985a).

Its first aim is the production and maintenance of a basic software library for CACSD. They name it SYCOT (SYstem and COntrol Tools). Implementation and documentation standards have been established. Contacts are made with persons and institutions all over the world to start cooperation and to get routines and information about existing packages. The result of this work will be an inventory of existing, available software which possibly can be used as a basis for SYCOT. The first list has already been published (WGS, 1985b) and a first release of SYCOT is expected in 1987 (WGS, 1986). Then the gaps in SYCOT will be filled in by own or external improvements of existing routines or by development of new routines. The long term aim is an interactive, user-friendly and flexible design software.

France

In France at INRIA Delebecque and Steer develop an interactive system for control engineering called Blaise (Delebecque and Steer, 1985; Delebecque 1986). The starting material has been MATLAB (Moler, 1980). Blaise has both real and complex matrices. Data structures for strings and polynomial matrices have been introduced. Rational function matrices are not supported due to the lack of good numerical algorithms. A linear system can be represented as a triple of matrices or as a pair of polynomial matrices. Function macros make it possible define parameter and time dependent matrices and nonlinear systems. Facilities for analysis and design in the time domain as well as in the frequency domain are available. It is also possible to simulate ordinary differential equations and solve optimization problems. Macros can be used to interactively define new algorithms. An interesting feature of Blaise is that it has an interface to the symbolic formula manipulation program MACSYMA (1982). They plan to implement an automatic generation of Blaise macros from MACSYMA expressions. The desire to have a portable system and the fact that most of the available numerical packages as EISPACK (Smith et al, 1976; Garbow et al, 1977) and LINPACK (Dongarra et al, 1979) are written in Fortran have imposed them to choose Fortran for the implementation.

We have a version of Blaise on our VAX and we have common interests

in graphics, symbolic calculation and design of high level languages.

We have also contact with another project at INRIA. It is a joint project with the Electrical Engineering Department, University of Maryland, USA. They develop a prototype expert system for the treatment of stochastic control and nonlinear filtering (Gomez et al., 1984). It accepts user input in natural language or symbolic form and then carries out the basic analysis of the problem in symbolic form and outputs Fortran code for the numerical reduction of the problem. The system itself is written in MACSYMA and Lisp.

Switzerland

In Switzerland at the Institute of Automatic Control at ETH, Zürich there is a project to develop a CAD-program for control systems. The program is named IMPACT (Interactive Program for Automatic Control Theory) (Rimvall and Cellier, 1984). An interesting fact about IMPACT is that it is implemented in Ada. However, Fortran packages like EISPACK and LINPACK will be used until appropriate packages are available in Ada.

IMPACT is meant to be a general purpose CACSD package. It is inspired of MATLAB. However, unlike MATLAB it also supports other data structures than complex matrices like polynomials and rational functions and matrices thereof (Rimvall, 1986). There is also a data structure for aggregating the A, B, C and D matrices of a linear state space description. Function macros make it possible define parameter and time dependent matrices and nonlinear systems. Facilities for analysis and design in the time domain as well as in the frequency domain are available. Macros can be used to interactively define new algorithms. The standard command language is syntactically similar to Ada. A facility of concurrent sessions using window techniques are suggested to improve the user interface (Rimvall and Bomholt, 1985). The user can then treat different aspects of the problem in different windows. He can also easily create scratch pads without cluttering the rest of the calculations. Macros are normally run in command mode, but a facility for running macros in question/answer mode is suggested. The system then outputs information supplied by the macro and prompts for the parameters.

USA

The Research and Development Center, General Electric Company, Schenectady, NY has played an important role. A workshop held in Schenectady and Troy, NY sponsored by GE and RPI initiated a number of IEEE CSS activities as described above.

GE has developed what they call the Federated Computer-Aided Control Design System (Spang, 1984). It consists of several separately developed subsystems tied together by a unified database. The Federated System includes Idpac for modelling, CLADP (Cambridge Linear Analysis and Design Program) and SSDP (GE State Space Design Package) for analysis and design and Simnon for nonlinear simulation. MATLAB has also been added to provide flexible means of handling matrix equations. The Federated System also includes programs to go from one subsystem to another. To simplify the use of the Federated System there is a joint project with the United States Military Academy, West Point and RPI to develop an expert system (Taylor and Frederick, 1984; James, Taylor and Frederick, 1985).

The general purpose, interactive, optimization-based CAD-system DE-

LIGHT (Polak et al., 1982; Nye and Tits 1982; Nye, 1983), which has been developed by professor Polak's group at University of California, Berkeley is another source of inspiration. DELIGHT has some interesting graphical representations for gradients, constraints and criteria while solving a multi criteria optimization problem. DELIGHT is actually a successor to Intrac. We provided prof. Polak with Intrac in source code and programming assistance. Our programmer Tommy Essebo was invited to Berkeley for two three month periods to get the DELIGHT project started. We have also been kept informed about the progress with DELIGHT continuously and we have access to the software.

The Lawrence Livermore National Laboratory has also been involved in the DELIGHT project and their experiences are used in the EAGLES project (Lawver and Poggio, 1985; EAGLES, 1986) and the M language project (Gavel et al, 1985; EAGLES, 1986). EAGLES is the Environment for Advanced Graphical (LLNL) Engineering Systems on workstation computers. It is meant to support a user friendly interface to engineering tools with an emphasis on graphical interfaces. The EAGLES interactive environment interprets M language constructs. The M language handles general expressions involving complex variables, polynomials, transfer functions and matrices of these types. The part of the M language dealing with matrix manipulation is quite similar to MATLAB. String manipulation is also supported. The language includes also ordinary if-then-else, for, and while statements. Functions may be matrix-valued. M also has Simnon-like constructs for defining ordinary differential and difference equations for dynamic simulation. We have got several preliminary versions, the latest is dated October 1986. It will be possible to get a system for a nominal fee, since it is public domain software.

We will of course continue to have contact with Dr. Herget and his research group at Lawrence Livermore National Laboratory. They have most types of workstations and we are invited to test different workstations, and learn about their experiences.

We have been in very close contact with the Systems Research Center at University of Maryland from its conception. We were also listed as one of their international collaboration partners in their application to NSF (Baras, 1984) and a formal agreement of collaboration has been signed. We strongly believe that a liason with University of Maryland can be extremely valuable.

The Department of Energy (DOE) funds a large project to develop CACSD software named CASCADE (Computer-Aided Systems and Control Analysis and Design Environment). The first phase of this project has been the specification of the requirements (Birdwell, 1984). Several organizations participated. The Oak Ridge National Laboratory acted as the principal contractor and provided expertise in energy systems, numerical analysis and optimization theory. The University of Tennessee provided program support services and expertise in digital systems, computer science and expert systems. The Lawrence Livermore Laboratory provided expertise in computer graphics, systems theory and CAD. The Charles Stark Draper Laboratory provided expertise in CAD and large systems. Professor Alan Laub of the University of California, Santa Barbara provided expertise in numerical analysis. Professor Michael Athans, MIT provided expertise in estimation and Control Theory. Dr. Gunter Stein, Honeywell and MIT and Dr. Austin Spang, General Electric and MIT have also been advisors.

Most of the recommendations are generally agreed upon like the need of

reliable numerical software, interactive input and output graphics support. However, there are also recommendations which are open to discussion. It is recommended to use the Unix environment as a host for CASCADE and to use C and Fortran for implementation. It is recommended that the design procedures should be based on optimization rather than depend on the user's intuition to choose. It is stated: "The converse to this recommendation is that the CASCADE cannot do all the design work; the engineer must remain in control in an interactive design approach". From our point of view this is an empty statement, because this assumes a fixed criterion and fixed constraints. This is usually not the case, but the criterion is just another way to parameterize the design problem and the designer changes the parameters of the criterion until he is satisfied. This is also the recommended way of using the interactive optimization-based system DELIGHT, where the user can trade off the constraints against the performance objectives.

The second phase, which is under way, will produce a prototype CASCADE for linear systems analysis and design. The time for this phase is 1.5 years. Five features are emphasized: expert system interface (Birdwell et al., 1985a,b), standardized device-independent graphics, reliable and robust numerical software, distributed computing, large scale system modeling and design. The objective of the third phase is to design and implement a production grade CASCADE. This effort is expected to require a 25 to 30 man-year effort and four years.

LMI Process Systems, Cambridge, Massachusetts has an very interesting project for development of a real-time expert system for process control called PICON (Moore et al. 1985). The process control group at LMI has split off and started a new company Gensym which will produce a system similar to PICON running on several different machines. Systems Control Technology in Palo Alto, California runs projects for development of expert systems for control design. One system is called RIFCS Reconfigurable Flight Control System (Trankle and Markosian, 1985a,b) and another is ACOLADES (Trankle et al, 1986). Reasoning Systems, Palo Alto is working on a very advanced concept for a knowledge-based environment for software design and synthesis called REFINE (REFINE, 1985). It is based on ideas developed by Cordell Green at the Kestrel Institute (Smith et al, 1985; Rockmore, 1985). The idea is based on giving formal executable specifications. The code is then generated by a knowledge based code generation system. There is a variety of tools for validation. The code generation can also be made in many different way depending on the purpose, rapid prototyping, memory efficient code etc. We have good personal relations with the group at Reasoning Systems.

Dr. Gary Kopec, Fairchild Laboratory for Artificial Intelligence, Palo Alto, California has developed some interesting system for signal processing. First, there is the integrated signal processing system ISP (Kopec, 1984a) with the signal representation language SRL (Kopec, 1983). It provides an unified environment for signal data processing and development of signal processing algorithms. ISP provides a signal stack, display windows, signal pictures which are used to create and view selected signals from the universe defined by SRL. Second, there is the signal database system SDB (Kopec, 1984b), which is a set of facilities for managing a corpus of signal data and the results of operations performed on them. These two systems are interesting because they may give us ideas on how to use graphics, how to represent data and how to design databases, query systems and high level languages. These systems are written

in Zetalisp and run on Symbolics 3600. It would probably be possible to get the systems for a nominal fee for non-commercial use. They have been released for use at a number of universities in the USA.

We have well-established collaboration with prof. Alan Laub of the University of California, Santa Barbara. He works with numerical linear algebra and Riccati equation solvers. Contact has also been established with Dr. Linda Petzold, Lawrence Livermore National Laboratories. She is expert on numerical solution of differential- algebraic equation (DAE) systems. Her solver DASSL (Petzold, 1982) has the reputation of being one of the best and most robust numerical solvers for DAE systems. We use it in the simulator Hibliz (Elmqvist and Mattsson, 1986; Mattsson et al, 1986).

3. Conclusions

Several trends can be recognized. First, there are efforts to make extensions to MATLAB by introducing new data types besides complex matrices. Examples of such efforts are Blaise (Delebecque and Steer, 1985), IMPACT (Rimvall and Cellier, 1984) and the M language (Gavel et al, 1985). The need to consider standardization to make it possible to switch between different software is also recognized by international organizations like IEEE and IFAC. There is also a desire to view software as tools and have possibilities to aggregate these tools in an open and extensible toolbox and to avoid as closed packages. This aspect plays an important role in the programme supported by SERC in the United Kingdom. Second, there are efforts to introduce expert knowledge in CACE systems. Most of these efforts are currently done in USA; General Electric, the United States Military Academy, West Point and RPI (Taylor and Frederick, 1984; James, Taylor and Frederick, 1985), the CASCADE project funded by the DOE (Birdwell et al., 1985a,b), the PICON project (Moore et al., 1985), and Systems Control Technology (Trankle and Markosian 1985a,b; Trankle et al, 1986). The use of expert system techniques for CACE is still in its infancy and our project to develop an expert system interface for Idpac is certainly in the front line. Third, everybody talks about graphics. However, the use of graphics in the control community is still at a low level; typically trend curves, Bode and Nyquist plots etc. People who we have talked with and who attended CACSD'83 in Boston remember still Hilding Elmqvist's presentation of LICS (Elmqvist, 1985) where he use hierarchical block diagrams and information zooming to describe the system structure and its decomposition. These ideas have been used in Hibliz (Elmqvist and Mattsson, 1986). Lack of graphics hardware has hampered the use of graphics. The new workstations with high performance graphics to reasonable cost have changed the situation drastically and a number of projects have been started: Barker et al (1986, 1987), EAGLES (1986), King and Gray (1985) etc. The Macintosh is a major source of inspiration. Fourth, numerics is as always an important area and symbolic manipulation has been introduced in CACE, for example Gomez et al. (1984).

The international exchange program is running well. Contacts are established with most of the important groups. We have benefitted from long-established contacts and the reputation of the department gained from older projects. The outcomes of the older projects have been valuable goods. Most people we have had contacts with has found our new CACE project to be both extensive and exciting and will therefore cooperate with us.

4. References

BARAS J.S. (Ed.) (1984): "Proposal for the establishment of a Systems Research Center submitted to National Science Foundation," College of Engineering, University of Maryland, Maryland, USA.

BARKER, H.A., M. CHEN, C.P. JOBLING and P. TOWNSEND (1986): "Interactive Graphics for the Computer-Aided Design of Dynamic Systems," IEEE International Conference on Systems, Man and Cybernetics, Atlanta, Georgia, October 14–17, 1986.

BARKER, H.A., M. CHEN, P.W. GRANT, C.P. JOBLING and P. TOWNSEND (1987): "The Development of an Intelligent Man-Machine Interface for Computer-Aided Design, Simulation and Implementation of Control Systems," IFAC World Congress, Munich, Germany, 1987.

BIRDWELL, J.D. (1984): "Issues in the Design of a Computer-Aided Systems and Control Analysis and Design Environment (CASCADE)," ORNL/TM-9038, Oak Ridge National Laboratory, Oak Ridge, Tennessee, USA.

BIRDWELL, J.D. ET AL. (1985a): "CASCADE: Experiments in the Development of Knowledge-Based Computer-Aided Systems and Control Analysis and Design Environments," *Proceedings of the 2nd IEEE Control Systems Society Symposium on Computer-Aided Control System Design (CACSD)*, Santa Barbara, California, March 13–15, 1985.

BIRDWELL, J.D., J.R.B. COCKETT, R. HELLER, R.W. ROCHELLE, A.J. LAUB and M. ATHANS (1985b): "Expert Systems Techniques in a Computer Based Control System and Design Environment," *CADCE '85, 3rd IFAC/IFIP Symposium on Computer Aided Design in Control and Engineering Systems*, Lyngby, Denmark, July 31 - August 2, 1985, pp. 1–8.

DELEBECQUE, F. and S. STEER (1985): "The Interactive System Blaise for Control Engineering," *CADCE '85, 3rd IFAC/IFIP Symposium on Computer Aided Design in Control and Engineering Systems*, Lyngby, Denmark, July 31–August 2, 1985, pp. 44–46.

DELEBECQUE, F. (1986): "Some Remarks About the Design of an Interactive CACSD Package: The Blaise Experience," *IEEE Control Systems Society, Third Symposium on Computer-Aided Control Design*, Arlington, Virginia, September 23–26, 1986.

DONGARRA, J.J., C.B. MOLER, J.R. BUNCH, and G.W. STEWART (1979): *LINPACK - Users' guide*, SIAM, Philadelphia.

EAGLES (1986): "EAGLES/Controls – Documentation," Lawrence Livermore National Laboratory, Livermore, California, USA.

ELCS (1986): "The Extended List of Control Software, ELCS," Number 2, June 1986, (Eds.) D.K. Frederick, C.J. Herget, R. Kool and M. Rimvall, Dept. of Automatic Control, ETH, CH-8092 Zürich, Switzerland.

ELMQVIST, H. (1985): "LICS - Language for Implementation of Control Systems," CODEN: LUTFD2/TFRT-3179, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. and S.E. MATTSSON (1986): "A Simulator for Dynamical Systems Using Graphics and Equations for Modelling," *Proceedings of*

the IEEE Control Systems Society Third Symposium on Computer-Aided Control System Design (CACSD), Arlington, Virginia, September 24-26, 1986.

FREDERICK, D.K. and M. RIMER (1985): "Benchmark Problems for CACSD Packages," *Proceedings of the 2nd IEEE Control Systems Society Symposium on Computer-Aided Control System Design (CACSD)*, Santa Barbara, California, March 13–15, 1985.

FREDERICK, D.K. and M. RIMER (1986): "IEEE Benchmarks for CACSD Packages," *IEEE Control Systems Society, Third Symposium on Computer-Aided Control Design*, Arlington, Virginia, September 23–26, 1986.

GAVEL, D.T., C.J. HERGET and B.S. LAWVER (1985): "The M Language, An Interactive Tool for Manipulating Matrices, Systems, and Signals," Lawrence Livermore National Laboratory, Livermore, California, USA.

GARBOW, B.S., ET AL. (1977): *Matrix eigensystem routines - Eispack Guide Extension*, Lecture Notes in Computer Science, Vol. 51, Springer-Verlag, New York.

GOMEZ, C., J.P. QUADRAT, A. SULEM, (INRIA, FRANCE), G.L. BLANKENSHIP, P. KUMAR, A. LAVIGNA, D.C. MAC ENANY, P. PAUL and I. YAN (UNIVERSITY OF MARYLAND) (1984): "An Expert System for Control and Signal Processing with Automatic Fortran Code Generation," *Proc. IEEE CDC, Las Vegas, Dec 1984*, pp. 716–723..

GOODFELLOW, S.D. and N. MUNRO (1985a): "Control System Computer Aided Engineering and Unix," *Proceedings of the 2nd IEEE Control Systems Society Symposium on Computer-Aided Control System Design (CACSD)*, Santa Barbara, California, March 13–15, 1985.

GOODFELLOW, S.D. and N. MUNRO (1985b): "An Integrated Environment for Computer Aided Control Systems Engineering," *CADCE '85, 3rd IFAC/IFIP Symposium on Computer Aided Design in Control and Engineering Systems*, Lyngby, Denmark, July 31–August 2, 1985, pp. 34–37.

GOODFELLOW, S.D. and N. MUNRO (1985c): "INTEGRA, An Input Translator Facility for Computer Aided Control Systems Engineering," *CADCE '85, 3rd IFAC/IFIP Symposium on Computer Aided Design in Control and Engineering Systems*, Lyngby, Denmark, July 31–August 2, 1985, pp. 104–109.

HAWLEY, P.A. and T.R. STEVENS (1986): "Two Sets of Benchmark Problems for CACSD Packages," *IEEE Control Systems Society, Third Symposium on Computer-Aided Control Design*, Arlington, Virginia, September 23–26, 1986.

IEEE CSS (1985): "News, Technical Committee on Computer-Aided Control Systems Design," IEEE Control Systems Society, Vol. 1, No. 2, Summer 1985.

JACOBS O.L.R. (1985): "Computer Aided Design of Control System in the United Kingdom," *CADCE '85, 3rd IFAC/IFIP Symposium on Computer Aided Design in Control and Engineering Systems*, Lyngby, Denmark, July 31–August 2, 1985, pp. 62–67.

JAMES, J.R., J.H. TAYLOR and D.K. FREDERICK (1985): "An Expert System Architecture for Coping with Complexity in Computer-Aided Control

Engineering," *CADCE '85, 3rd IFAC/IFIP Symposium on Computer Aided Design in Control and Engineering Systems*, Lyngby, Denmark, July 31–August 2, 1985, pp. 47–52.

KING, R.A. and J.O. GRAY (1985): "A Flexible Data Interpreter for Computer Aided Design & Simulation of Dynamic Systems," *CADCE '85, 3rd IFAC/IFIP Symposium on Computer Aided Design in Control and Engineering Systems*, Lyngby, Denmark, July 31–August 2, 1985, pp. 87–91.

KOPEC G. (1983): "The Signal Representation Language SRL," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Boston, MA, 1983.

KOPEC G. (1984a): "The Integrated Signal Processing System ISP," *Proceedings ICASSP 84 - IEEE International Conference on Acoustics, Speech, and Signal Processing*, March 19–21, 1984 or IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-32, No. 4, August 1984, pp. 842–851.

KOPEC G. (1984b): "The Signal Data Base System SDB," *Proceedings ICASSP 84 - IEEE International Conference on Acoustics, Speech, and Signal Processing*, March 19–21, 1984.

LAWVER, B. and POGGIO, P. (1985): "EAGLES Requirements," Computer Systems Research Group, Engineering Research Division, Lawrence Livermore National Laboratory, Livermore, California.

LITTLE, J.N., S. HERSKOVITZ, A.J. LAUB and C.B. MOLER (1986): "MATLAB and Control Design on the Macintosh," *IEEE Control Systems Society, Third Symposium on Computer-Aided Control Design*, Arlington, Virginia, September 23–26, 1986.

MACIEJOWSKI, J.M. (1984): "Data structures for control system design," *EUROCON 84*, Brighton UK, September 1984.

MACIEJOWSKI, J.M. (1985a): "Data structures and software tools in CACSD," *Proceedings of the 2nd IEEE Control Systems Society Symposium on Computer-Aided Control System Design (CACSD)*, Santa Barbara, California, March 13–15, 1985.

MACIEJOWSKI, J.M. (1985b): "A Core Data Model for Computer-Aided Control Engineering," Draft, Engineering Department, University of Cambridge, UK, June 7, 1985.

MACSYMA (1983): "MACSYMA Reference Manual," M.I.T., Cambridge, Massachusetts, USA.

MATTSSON, S.E., K.J. ÅSTRÖM, D. BRÜCK and T. SCHÖNTHAL (1987): "A Trip to the University College Swansea and the Central Electric Generating Board, Gloucester," Travel report CODEN: LUTFD2/TFRT-8045, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E., H. ELMQVIST and D.M. BRÜCK (1986): "New Forms of Man-Machine Interaction," Final Report 1986-09-30, STU project 84-5069, STU program: Computer Aided Control Engineering, CACE, Report CODEN: LUTFD2/TFRT-3181, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MOLER, C.B. (1980): "MATLAB – User's Guide," Department of Computer Science, University of New Mexico, Albuquerque, USA.

MOORE, R.L., L.B. HAWKINSON, C.G. KNICKEBOCKER and L.M. CHURCHMAN (1984): "Expert systems applications in industry," *Proc. ISA Int. Conf.*, Houston, Texas.

MUNRO, N., Z. PALASKAS and D.K. FREDERICK (1986): "An Adaptive CACSD Dialogue Facility," *IEEE Control Systems Society, Third Symposium on Computer-Aided Control Design*, Arlington, Virginia, September 23–26, 1986.

NYE, W.T. and TITS, A. (1982): "DELIGHT for Beginners," Memorandum No. UCB/ERL M82/55, Electronics Research Laboratory, College of Engineering, University of California, Berkeley.

NYE, W.T. (1983): "DELIGHT. An Interactive System for Optimization-Based Engineering," Memorandum No. UCB/ERL M83/33, Electronics Research Laboratory, College of Engineering, University of California, Berkeley.

PANG, G.K.H. J.M. BOYLE and A.G.J. MACFARLANE (1986): "An Expert System for Computer-Aided Linear Multivariable Control System Design," *IEEE Control Systems Society, Third Symposium on Computer-Aided Control Design*, Arlington, Virginia, September 23–26, 1986.

PETZOLD, L.R. (1982): "A Description of DASSL: A Differential/Algebraic System Solver," *Proceedings of IMACS World Congress 1982*, Montreal, Canada.

POLAK E., P. SIEGEL, T. WUU, W.T. NYE and D.Q. MAYNE (1982): "DELIGHT.MIMO: An Interactive Optimization-Based Multivariable Control System Design Package," *IEEE Control Systems Magazine* **2**, No. 4, Dec 1982, 9–14, Also in Jamshidi, M. and C.J. Herget (Eds.) (1985): Computer-Aided Control Systems Engineering, North-Holland, Amsterdam, The Netherlands, pp. 139–147.

REFINE (1985): *REFINE Software Development Tool*, Reasoning Systems, Palo Alto, California.

RIMVALL, M. and F. CELLIER (1984): "IMPACT Interactive mathematical program for automatic control theory," in Bensoussan and Lions (Eds.): *Analysis and Optimization of Systems*, Springer Lecture Notes in Control and Information Sciences, Springer, Berlin.

RIMVALL, M. AND L. BOMHOLT (1985): "A Flexible Man-Machine Interface for CACSD Application," *CADCE '85, 3rd IFAC/IFIP Symposium on Computer Aided Design in Control and Engineering Systems*, Lyngby, Denmark, July 31–August 2, 1985, pp. 98–103.

RIMVALL, M. (1986): "The Different Modelling Capabilities of IMPACT," *IEEE Control Systems Society, Third Symposium on Computer-Aided Control Design*, Arlington, Virginia, September 23–26, 1986.

ROCKMORE, A.J. (1985): "The knowledge-based design and synthesis approach to software development," Reasoning Systems Inc., Palo Alto, California, USA, Also in Electronic Design Magazine, July 1985.

SERC (1986): "Control and Instrumentation – Subcommittee Review," August 1986, Science and Engineering Research Council Engineering

Board, Information Engineering Committee, Control and Instrumentation Subcommittee, Swindon, UK.

SMITH, B.T., ET AL. (1976): *Matrix eigensystem routines - Eispack guide, 2nd ed*, Lecture Notes in Computer Science, Vol. 6, Springer-Verlag, New York.

SMITH, D.R., G.B. KOTIK and S.J. WESTFOLD (1985): "Research on Knowledge-Based Environments at Kestrel Institute," Kestrel Institute, Palo Alto, California, USA.

SPANG, H., III (1984): "The Federated Computer-Aided Control Design System," *Proc. of the IEEE* **72**, No. 12, Dec 1984, 1724–1731.

TAYLOR, J.H. and D.K. FREDERICK (1984): "An Expert System Architecture for Computer Aided Control Engineering," *Proc. of the IEEE* **72**, No. 12, Dec 1984, 1795–1805.

TRANKLE, T.L. and L.Z. MARKOSIAN (1985a): "Control Design Using an Expert Planning System," *Proceedings of the 2nd IEEE Control Systems Society Symposium on Computer-Aided Control System Design (CACSD)*, Santa Barbara, California, March 13-15, 1985.

TRANKLE, T.L. and L.Z. MARKOSIAN (1985b): "An Expert System for Control Design," *IEE Control 85*, Conference Publication Number 252, Vol. 2, pp. 495–499.

TRANKLE, T.L., J.D. PEHOUSHEK and P. SHEU (1986): "ACOLADES: A Control Law Designing Expert System," *IEEE Control Systems Society, Third Symposium on Computer-Aided Control Design*, Arlington, Virginia, September 23–26, 1986.

WGS (1985a): "WGS Newsletter," No. 1, Jan 1985, Benelux Working Group on Software, Dept. of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands.

WGS (1985b): "An Inventory of Basic Software for Computer Aided Control System Design (CACSD)," Benelux Working Group on Software, WGS-Report-85-1, Benelux Working Group on Software, Dept. of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands.

WGS (1986): "WGS Newsletter," No. 4, Dec 1986, Benelux Working Group on Software, Dept. of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands.

WHITMARSH-EVERISS, M.J. (1986): "The CEGB federated computer design suite for operational dynamics, control system design, fault analysis and code validation using plant operational data and its relation to the plant design analyser and graphics systems," PKR/SE/214, CEGB, Barnwood, Gloucester, UK.

Paper presented at IEEE Control Systems Society Third Symposium on Computer-Aided Control System Design (CACSD), Arlington, Virginia, September 24-26, 1986.

A SIMULATOR FOR DYNAMICAL SYSTEMS USING GRAPHICS AND EQUATIONS FOR MODELLING

Hilding Elmqvist

SattControl AB
Malmö, Sweden

Sven Erik Mattsson

Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

Abstract

The new workstations with high performance graphics offer new possibilities for man-machine interaction. This paper presents a prototype simulator for dynamical systems, called Hibliz which explores some features of modern computer graphics. Hibliz supports hierarchical block diagrams to describe the model decomposition and the interconnection structure. The user can scroll, pan and zoom the block diagram continuously in real-time. Zooming controls the amount of information displayed. When zooming in on a block, it changes from an annotated box to a representation showing internal structure with increasing detail. Since the block diagrams can be hierarchical, it is possible to make the description at each level simple and clear. Hibliz also simplifies model development by allowing submodels in the form of ordinary differential and algebraic equations rather than assignment statements for derivatives and algebraic variables.

1. INTRODUCTION

It is difficult for a human to develop and handle models of large and complex systems, because he is unable to deal with many entities simultaneously. Consequently, a system for model development and simulation ought to have structuring facilities so that the user can view a model from different viewpoints, each having only a small number of entities. Up to now, it has been difficult to design and implement such facilities because structural properties are not easy to represent textually. The decomposition of a model into submodels with interconnections are much easier described graphically. The current trends in scientific personal computers will make graphical displays commonly available. This will revolutionize the man-machine interaction. Apple's Macintosh is a clear indication of what we may expect. Such machines will offer great opportunities to invent more efficient problem solving tools.

This paper presents a prototype system for model development and simulation. The simulator explores some of the possibilities of new workstations with high performance, real-time graphics. Special attention has been given to the use of graphics to describe structural properties. Hierarchical block diagrams are used to describe the model decomposition and the interconnection structure. By moving a mouse and pressing its buttons, the user can scroll, pan and zoom the block diagram continuously in real time. Information zooming is used to control the amount of information displayed. When zooming in on a block, it changes from an annotated box to a representation showing the internal structure with increasing detail. The concept of information zooming was introduced by Elmqvist (1985). The prototype simulator is called Hibliz (HIerarchical BLock diagrams with Information Zooming).

Hibliz is a simulator for dynamical systems, described by sets of ordinary differential equations and algebraic equations. The structuring concepts proposed can also be used for more general systems with both continuous time and discrete time submodels, however, the algorithm presently included for simulation can only handle continuous time models. Hibliz allows the submodels to be described in the form of equations instead of assignment statements to facilitate modelling and use of model libraries.

The paper is organized as follows. The model description concepts of Hibliz are discussed in Section 2. The operation of Hibliz is described in Section 3 and the implementation in Section 4. Some conclusions are given in Section 5.

2. MODEL DESCRIPTION CONCEPTS

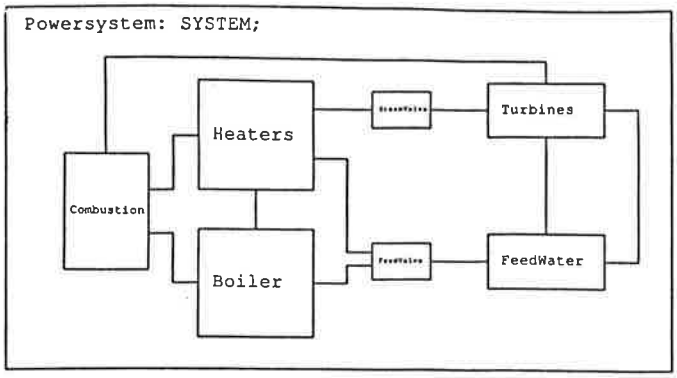
Two basic principles can be used to structure a model: abstraction and modularization. The essence of abstraction is to extract important properties while omitting insignificant details. Different levels of abstraction are defined, allowing the system to be viewed with increasing detail. The first abstraction level for a model might be just its name or icon. The next level might contain a description of usage and external behaviour, and a third level details about its internal behaviour. The amount of information increases at lower abstraction levels. Modularization can be used in order to maintain useful views with a limited number of related concepts. Modularization means that the information at a certain abstraction level is decomposed into smaller entities.

Our concept is that a graphical description of the structure is easier to understand than a textual description. Modularization is achieved by use of block diagrams. To support abstraction we propose information zooming and hierarchical block diagrams. Multiple windows are used to further support multiple views of the model.

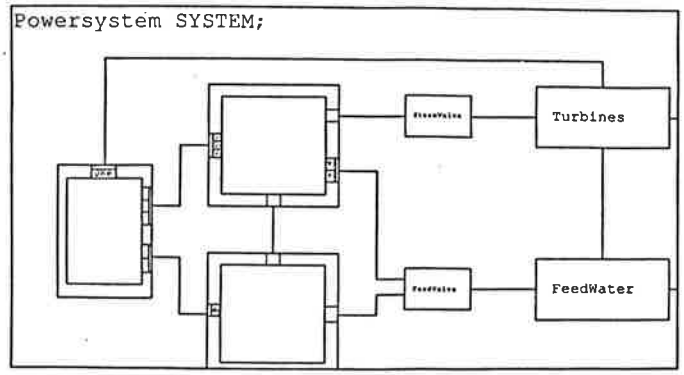
The use of hierarchical block diagrams and information zooming will be illustrated by an example. Please remember that when sitting at the terminal you can scroll, pan and zoom in the windows, but that the dynamical aspects are lost in a paper which only can show snapshots of the screen.

An Example

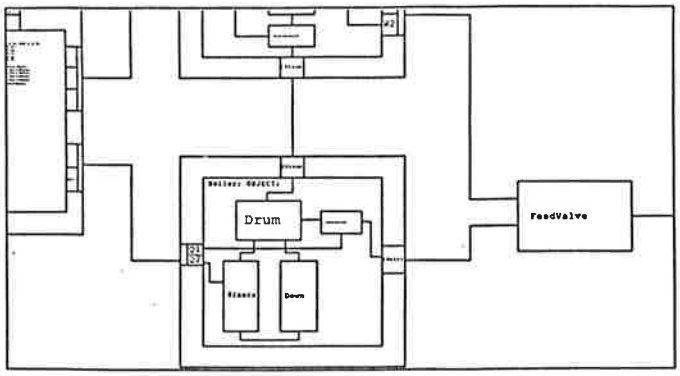
As an example consider a model of a thermal power plant. The block diagram in Picture 1 shows the major components and their interaction. The annotated boxes represent submodels and the lines between the boxes indicate interaction between the submodels.



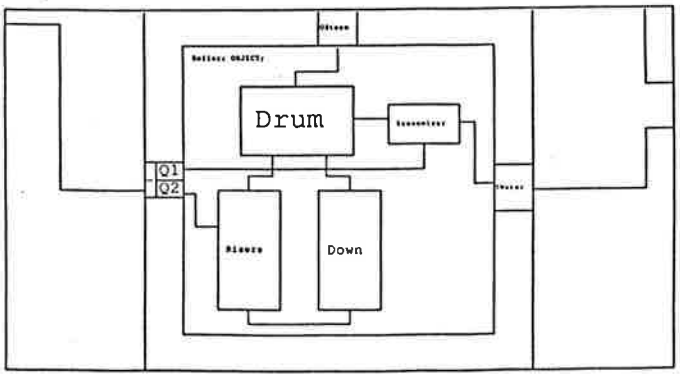
Picture 1. Power system model.



Picture 2. The power system model zoomed-in a little.



Picture 3. The power system model zoomed-in further.



Picture 4. The power system model zoomed-in at Boiler.

To the left in Picture 1, we find the model for the combustion chamber. It delivers energy to the boiler, the heaters and the reheaters in the turbine part. The boiler produces steam, which is heated in the superheaters. The steam then goes to the turbines via the steam valve. From the turbine system the steam enters the feed water system. Extract steam from the turbines is used to preheat the feed water. The feed water goes to the boiler, but also to sprayers in the heater. The equations describing the system are typically mass and energy balances. External functions for interpolating in steam tables are also required. The model has 470 variables and in its textual form it is 1200 lines long, including layout information.

By pressing the right mouse button when moving the mouse, we scroll and pan the picture continuously. If we press both the right and the middle button, we zoom. Let us do that (Pictures 2 and 3). When we zoom in on the diagram, the blocks open up and show their internal details.

Let us zoom further (Picture 4). Now we can see the internal description of the boiler. It is a new block diagram.

Model Decomposition

Hibliz supports the use of hierarchical block diagrams as a tool to handle complexity. A basic rule is that a block diagram should be simple and contain only a small number of blocks. The selection of module boundaries is guided by our perception of the problem space. If our first attempt of structuring results in too many blocks, it is advisable to introduce a new hierarchy. There is almost always interaction (coupling) between modules. In order to be useful, a decomposition must be chosen in such a way that the external interaction complexity is small compared with the internal complexity. There should not be criss cross lines between the blocks. Furthermore, the entities in a module should be related (cohesion).

Modularization gives many advantages. It simplifies the modelling, it makes the model more flexible and easier to adapt and manage. We can also build and use libraries of models. Technical systems are often built in a modular way composed of standard components. Their behaviour may be well-known. Good, generally accepted models may already exist.

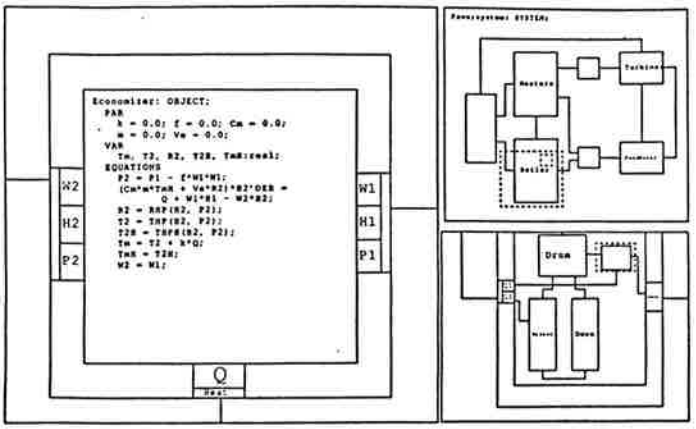
Multiple Windows

The user can create new windows for viewing a model. One of these windows is the current interaction window for scrolling, panning and zooming. To help the user keep track of where he is, rectangles outline the parts of a window that are also shown in other windows (See Picture 5, dashed rectangles). If the user wants to move fast he can point at an object in any window and ask for automatic scrolling, panning and zooming to this object in the interaction window.

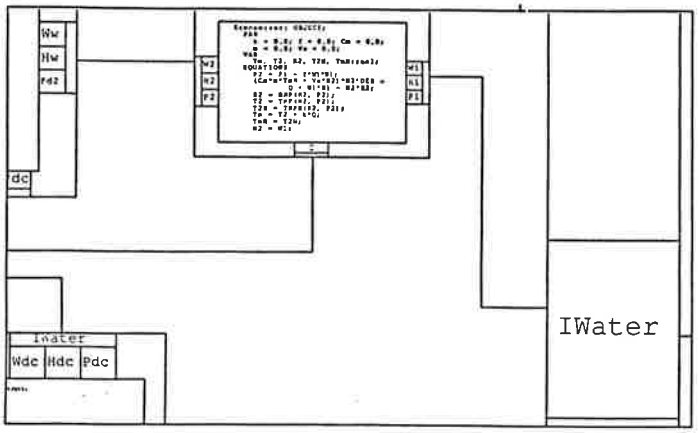
Interaction Structure

Let us consider the interaction between models. The first level, the block diagram (Picture 1) shows which models interact. The next level (Picture 4) is concerned with how models interact, i.e. which variables that are involved. The graphical representation for a submodel consists of two rectangles; one inside the other. The descriptions of interfaces are placed at the border of the submodel between the inner and outer rectangle. At the most detailed level, the effect of interaction can be seen in the equations containing interaction variables.

A model is an encapsulated entity and the interaction variables are the only visible from outside. The interaction variables



Picture 5. Multiple windows give different views simultaneously.



Picture 6. Zoomed-in picture of Boiler model.

are associated with submodels. They cannot be associated with connections, because it should be possible to develop a submodel without knowing in what environment it will be used. This is necessary in order to allow for model types and model libraries. A model often interacts with several other models, implying that the formal interaction variables should be grouped corresponding to the different possible connections. Such groups are called interfaces. Interfaces may have a hierarchical structure.

Consider Picture 6. It shows a zoomed-in picture of the model Boiler. To the right, the interface Iwater of Boiler can be seen. It models the incoming feed water to the boiler. This interface is connected to the right interface of Economizer to model that the feed water goes to the economizer. The feed water is heated in the economizer. It then leaves the economizer and flows into the drum. This is modelled by the connection between the left interface of Economizer and the right interface of Drum. A connection between two structured interfaces means that their corresponding components are connected. The number of components must be the same in the two interfaces. Primitive interface components may also be used to pass through a structured connection to submodels. The user need only specify the interfaces of non-primitive models to the degree of detail that is necessary to draw the block diagram within the actual block. The connection between the interface Iwater of Boiler and the right interface of Economizer illustrates this.

Behaviour

Before discussing the semantics of connected variables, let us have a look at a primitive submodel. The model Economizer is one. We can as before zoom manually, but there is also a facility for quick automatic zooming. If we point with the cursor at the outer rectangle of a model and press the middle button, Hibliz automatically zooms in on the model (Picture 7).

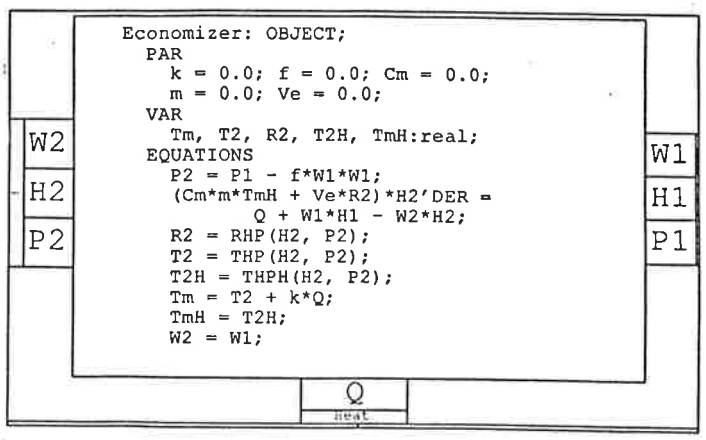
The behaviour of models at the lowest hierarchical level is described by equations in textual form. An equation should have the form

$$expression = expression$$

We will not describe the syntax and semantics of expressions in full detail (see Elmqvist 1985). Expressions have the usual syntax with arithmetic, relational and boolean operators. Conditional parts (`if-then-else` expressions) as in for example Algol 60 are also allowed. Common mathematical functions like sin, cos and exp are available. The language supports simple integer, real and boolean variables. Hibliz also provides a mechanism for incorporating additional functions written in Pascal. These functions can be used directly in the model. For example RHP, THP and THPH used in the model Economizer (see Picture 7) are such external functions implementing steam tables.

There are four kinds of variables; constants, parameters, interface variables and internal variables. Constants and parameters are considered to be constant during a simulation, but the user can change the values of parameters on-line between simulations. The values of interface variables and internal variables may of course vary with time. The time derivative $\dot{x}$ of a variable x is written as `x'der`. The scope rules for variables are very simple because of the powerful connection concept. Variables can only be referenced from equations in the submodel where they are declared.

It should be noted that the basic concept is not assignment statements but general equations. A model can thus be represented as $f(t, \dot{x}, x, p) = 0$. Many integration algorithms and simulation packages require that the derivatives are solved explicitly by the user; $\dot{x} = F(t, x, p)$. The support of general equations simplifies the model development and the documentation becomes better, since equations are closer to first principles. When developing a model for a physical system one uses fundamental laws such as mass balances, energy balances and phenomenological equations. These are either algebraic equations or ordinary



Picture 7. Economizer model.

differential equations which relates certain variables. Compared with the assignment form, it is easier to check that the model is entered correctly and the risk of introducing errors during manual transformation into assignment statements is reduced.

Furthermore, as thoroughly motivated by Elmqvist (1978), the equation form is the only reasonable representation for model libraries. With models on assignment form, it must be decided for each submodel which of its variables that are inputs (in other words are known) and which of its variables that are outputs (defined by the model). As a simple example consider a resistor. Ohm's law states $V_1 - V_2 = RI$, where V_1 and V_2 are the voltages at the ends of the resistor, I is the current through the resistor and R is the resistance. The model has three variables V_1, V_2 and I. The resistance R is in this model a given parameter. If we should write the model on assignment form there are three possibilities

$$I := (V_1 - V_2)/R$$
$$V_1 := V_2 + RI$$
$$V_2 := V_1 - RI$$

The first variant assumes that V_1 and V_2 are inputs and defines I. This model is appropriate if for example one end of the resistor is connected to a voltage source and the other to ground. The second and third variants assume that the current and the voltage at one end is known. These models are appropriate if the resistor is connected to a current source and ground. Consequently, for models on assignment form we need several different models for a resistor, depending on how it is connected to the environment. This makes both use and maintenance of a model library messy. Furthermore, other environments may result in algebraic loops so that equation systems with equations from several submodels must be solved to transform the model into assignment form. Two resistors connected in series between a voltage source and ground is a simple example of this. Submodels cannot be transformed into assignment form individually, but the transformation is a global problem.

A more sophisticated connection mechanism can be introduced when equations are allowed. Hibliz supports two kinds of connection semantics depending on the character of the interaction variable. Consider, for example, three connected electrical wires. Each wire is represented by a voltage and a current. The constraints at the junctions are that the voltages are equal and the sum of the currents is zero. These two types of cut variables are sometimes called across variables and through variables. Other examples of across variables are pressure and temperature. Mass and energy flow, thrust and torque are examples of through variables. It is natural to associate a direction with a through variable. The directions are important when putting up the equation for connected variables so that the terms in the "zero sum equation" are given correct signs. The interfaces are presently declared textually, for example, the interfaces of a resistor may be declared as

```
End1:  (V1:  real, I1:  IN real)
End2:  (V2:  real, I2:  OUT real)
```

An interface variable is defined to be a through variable by the keyword `IN` or `OUT` as indicated above for `I1` and `I2`.

Hibliz interprets the connections drawn by the user and generates appropriate equations automatically.

3. HOW HIBLIZ IS OPERATED

The user operates Hibliz via a keyboard and a mouse with three buttons. He can create and edit hierarchical block diagrams, inspect the model and simulate.

Scrolling, Panning and Zooming

The mouse is used for pointing on the screen; a cursor follows the movements of the mouse. We indicated earlier how the mouse is used when scrolling, panning and zooming. In order to scroll and pan, press the right button and move the mouse. To zoom, press first the right button and then the middle button. Now moving the cursor up means zooming in and down zooming out. An object (model, interface, curve or text) can be zoomed-in by pointing at it and pressing the middle button only. Zooming is done smoothly and the final size of the object is chosen as large as possible while still being contained in the window. Text objects are treated specially. Their final sizes are chosen so that the longest line of the text matches the width of the zoom window. The text line pointed at is scrolled to the center of the zoom window. This allows convenient scrolling within a window. A connection is considered to belong to the enclosing model. Pointing at a connection is therefore a convenient way of zooming out to a higher hierarchical level. If the middle button is pressed without pointing at anything, a zoom-out to 70% is performed. Note that it is possible to point at an object in a window other than the current zoom window. Pointing in an overview window thus allows rapid inspection of different objects. A new window is selected to be the current zoom window by pointing at it and pressing the left and right buttons.

Two-Button-Stretching

To layout objects like windows, models and interfaces on the screen is a common operation. The layout with respect to position and size is done with "two-button-stretching". The lower left corner and the upper right corner of the objects follows the mouse in different ways depending on which of the left and middle buttons that are pressed. If the left button is pressed, the cursor points at the lower left corner which follows the movement of the mouse. The same thing applies for the upper right corner when the middle button is pressed. If both buttons are pressed the cursor points at the center of the object and the whole object moves. The stretching is finished when both buttons are released. It should be noted that the objects are completely redrawn, even in the two-button-stretching mode.

Commands

Commands are chosen from a pop-up menu. The menu is shown when the left button is pressed. The desired command is selected by pointing at the corresponding menu entry. Hibliz highlights the selected entry and performs the command when the button is released. If no entry is selected when releasing the button, the menu just disappears. Command actions sometimes require additional input from keyboard or mouse. Hibliz then prompts in the command area at the bottom of the screen.

As an example we will describe the Model command in more detail. This command creates a model and its graphical representation (two rectangles and the name). Hibliz prompts for name, which the user should type on the keyboard, and the enclosing model, which the user should select by pointing at it with the mouse and pressing the left button. The layout is done with "two-button-stretching" as described above.

The Interface command is used to declare and position an interface. An interface is presently declared textually and its graphical layout is done automatically. The user positions and stretches it using two-button-stretching.

The Connect command makes it possible to draw connections between interfaces by using the mouse and the left button to input a sequence of line segments. The start interface is first selected by pressing the left button. Break points are given by releasing and pressing the button. The last line segment is refreshed while the mouse is moved (rubber-band drawing) and the interface structure is searched for the destination interface.

The Text command is used to edit the textual parts of models. The editing is performed using a simple screen oriented editor. When the user leaves the editor the text description is parsed. Error messages are currently given as text in the command window. Here there are lots of possibilities to use graphics and color to explain the error to the user.

The Remove command allows deletion of models, interfaces and connections. The Copy command copies a model and all its submodels, interfaces, connections and the text. The Save command stores the current model hierarchy as a text file. Such a file can be read by the Get command to recreate the model. If a model has been defined earlier, the hierarchical position and layout is given as for the Model command. The Copy, Save and Get commands make it possible to build simple model libraries.

The Layout command is used for changing the position and size of windows and models. There is no facility to change layout of interfaces and connections. The View command creates a new window for viewing the model.

The Compile command analyses the model and prepares for simulation. The Simulate command prompts for start and stop time and then starts the simulation.

The Display command creates a display for presentation simulation results of any variable. The presentation of simulation results is very primitive at present. Only simple trend curves are implemented.

The Hardcopy command creates a description of the current content of the screen in the form of a PostScript program (Adobe Systems Incorporated, 1985). The program can then be used to create hard copies on for example Apple's LaserWriter. The pictures in this paper have been made in this way.

The Exit command stops Hibliz and returns to the operating system.

4. IMPLEMENTATION

The code for Hibliz consists of some 28 000 lines of Pascal. The software is highly modular. Related types, variables and procedures are grouped together. Machine dependent parts like file and string handling are isolated to improve portability. A preprocessor which we call Packman is used to produce a standard Pascal program from module files, since Pascal does not allow mixed declarations of global constants, types, variables and procedures.

Packman

Packman accepts files consisting of sections of code preceded by headings such as `.PROGRAM`, `.LABEL`, `.CONST`, `.TYPE`, `.VAR`, `.FORWARD`, `.PROCEDURE`, `.INIT` and `.MAIN`. Packman outputs the contents of all sections labeled `.CONST` on a file named `CONST.SEC` and so on.

Typically the file issued to Packman is a short file containing a number of commands to include files. Inclusions may be nested.

Hibliz is built like a transparent onion consisting of six layers. Outer layers can use elements of inner layers. Packman is designed to promote separate compilation. By the commands `.DEFINITION` or `.IMPLEMENTATION` it is possible to specify what should be visible outside the compilation unit and what should be hidden. Portability with respect to separate compilation facilities of different Pascal compilers can be handled by modifying Packman.

Data Structure

A model is represented as a record containing lists of submodels, variable declarations, connections and equations. A general list package for doubly linked lists with headers is used. It has operations such as `NewList`, `Into`, `First` and `SuccElem`. A node is a Pascal record with variants. The common area contains information like forward and backwards pointers for list manipulation, and the variant part contains a pointer to a record describing models, interfaces, connections etc. Note that this way of implementing lists makes it possible for a model description etc. to be a member of several lists at the same time. The list package makes it easy to handle and manipulate lists.

Compiler

The model descriptions at lowest level are parsed at exit from the editor or when read from a file by the Get command. The parser builds a syntax tree for each equation. The Compile command links the identifiers to their declarations. Interface variables which are connected to each other are put into a list. These lists of connected interface variables are then analyzed to generate the proper equations, their syntax trees and the links to declarations. It also checks that across and through variables are not connected to each other. The type consistency of all expressions is checked.

When Hibliz has collected all equations, it has a differential/algebraic system of the form

$$g(t, \dot{x}, x, v, p, c) = 0$$

where t is the time, x and v are vectors of unknown variables, p is a vector of known parameters and c is a vector of known constants. The vector v contains those unknown variables which do not appear differentiated in the equations. Hibliz uses the differential/algebraic system solver DASSL (Petzold, 1982). DASSL has a reputation of being one of the best and most robust numerical solvers for differential/algebraic systems. DASSL accepts problems on this form if it is provided with a routine for calculating the residual $\Delta = g(t, \dot{x}, x, v, p, c)$ when the arguments are known. However, to decrease the order and the complexity of the problem, simple symbolic formula manipulation is performed as follows. Connections of across variables lead to simple identities of the form $A = B$. It is easy to explore these entities and eliminate variables. The record describing a variable has an element called `alias` with an initial value implying that it is its own alias. When simple equations are found, Hibliz modifies the alias elements accordingly and removes the equation from the list of interesting equations.

After the elimination of simple equations, Hibliz assumes that $\dot{x}$ and v are unknown and sorts the equations and the variables so that the problem becomes block lower triangular with minimal diagonal blocks. If a block is scalar and the variable assigned to the equation is a component of the vector v, and the equation is of the form $v_i = \langle$expression independent of $v_i\rangle$ the variable v_i is eliminated from the vector of unknown variables

passed to DASSL. The routine for evaluating the residual Δ can in this case calculate v_i itself. The partition to lower block triangular form may fail. An error message is then given listing unassigned variables and redundant equations. The problem is then either singular or it has algebraic relations between the components of the vector x.

To make the calculation of the residual vector Δ more efficient, Hibliz generates code for a virtual stack machine. The code is interpreted by a Pascal procedure. The values of constants, parameters and variables are stored in a global array.

The Run command sets the initial values of states as given by the initial section of the models and then uses DASSL to solve the system.

Graphics

Routines for handling graphics are an essential part of Hibliz. A local coordinate system is assigned to each model such that the lower left corner of the rectangle has the coordinates (0,0) and the upper right corner has the coordinates (1,1). The positions of its interfaces, submodels, equations etc. are expressed in this coordinate system. When moving and scaling a model, this hierarchy of coordinate systems makes it almost trivial to scale and move its submodels properly. All coordinates are stored as real numbers since continuous zooming requires high resolution coordinates.

Hibliz currently runs on an IRIS 2400 from Silicon Graphics, Inc. (Clark and Davis, 1983). The IRIS is a high performance engineering workstation designed for interactive color graphics and computing applications. The program interface to the graphics is the IRIS Graphics Library. It has routines for definition and manipulation of objects in (local) world coordinate systems and projection of these objects onto the screen. The IRIS has special graphics hardware for transformations from local world coordinates into screen coordinates. Clipping and scan conversion are also done in hardware.

Fonts

A block diagram contains text, and to make continuous zooming possible it is necessary to display characters of different sizes. The IRIS Graphics Library supports one fixed pitch raster font of height 16 and width 9. New raster fonts can be defined, but, due to memory constraints, large character has to be viewed as graphical objects consisting of straight and curved lines.

We have developed a support program to generate new fonts. This program is based on ideas given in Knuth (1979). The user defines the shape of a character as a number of line segments and the size and form (rectangular, circular, oval etc.) of the pen to be used. A line segment is defined by its start and end point and its tangents in these points. Intermediate points on the line segment are defined by the cubic spline function given in Knuth (1979, pp. 24-26).

Hibliz uses both raster and graphical fonts; for characters up to 16×20 pixels, raster fonts are used. The use of both raster and graphical fonts makes outputting of text somewhat more complex. There are, however, two good reasons for using raster fonts and not only graphical fonts. First, when drawing a small character the quantization may deform the character so that it looks ugly and is difficult to recognize. If a character is moved over the screen, its form changes due to the quantization and for example an 'o' looks like an amoeba. Second, it is important to make the graphics as fast as possible. When the characters are small there may be many of them on the screen. If the characters are defined as graphical objects this implies drawing of many line segments. For example, to draw a nice looking O, at least 20 line segments are needed. The IRIS can at maximum draw 65 000 line segments per second whereas it can display up to 150 000 raster characters per second.

5. CONCLUSIONS

The structural properties of a model are very important, particularly when working with large and complex systems. We believe that it is easier to describe structural properties when graphics is used than when a purely textual description must be used. In this paper we propose the use of hierarchical block diagrams, which can be scrolled, panned and zoomed continuously. The block diagram describes the model decomposition and the interconnection structure. The zooming controls the amount of information displayed. When zooming in on a block, the block changes from an annotated box into a representation showing internal structure with increasing detail. Since the block diagram can be made hierarchical, it is possible to make the description at each level simple and clear.

The new workstations with fast, high performance graphics make it possible to implement the man-machine interface proposed. To demonstrate the feasibility of our proposal, we have implemented a prototype system, called Hibliz. Hibliz also simplifies model development by allowing submodels in the form of ordinary differential and algebraic equations rather than assignment statements for derivatives and algebraic variables.

The only reasonable way to evaluate a user interface is to let a number of persons use it when they try to solve their problems. Such an evaluation has not been done yet. However, many demonstrations have been met with very positive response.

6. ACKNOWLEDGEMENTS

The authors would like to thank Professor Karl Johan Åström for many useful discussions and for his strong support. Many thanks to Tommy Essebo and Dag Brück, who have programmed parts of the system. This work has been supported by the National Swedish Board for Technical Development (STU) under contract 84-5069.

7. REFERENCES

ADOBE SYSTEMS INCORPORATED (1985): *PostScript Language, Reference Manual*, Addison-Wesley Publishing Company.

CLARK, J.H. AND T. DAVIS (1983): "Work station unites real-time graphics with Unix, Ethernet," *Electronics*, Oct. 20, 1983, 113–119.

ELMQVIST, H. (1978): *A Structured Model Language for Large Continuous Systems*, Ph.D-thesis, Report CODEN: LUTFD2/TFRT-1015, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. (1985): *LICS—Language for Implementation of Control Systems*, Report CODEN: LUTFD2/TFRT-3179, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

KNUTH, D.E. (1979): *TEX and Metafont - New Directions in Typesetting*, American Mathematical Society and Digital Press.

PETZOLD, L.R. (1982): "A Description of DASSL: A Differential/Algebraic System Solver," Proceedings of IMACS World Congress, Montreal, Canada, 1982.

Paper presented at IEEE Control Systems Society Third Symposium on Computer-Aided Control Systems Design (CACSD), Arlington, Virginia, September 24-26, 1986

SYSTEM REPRESENTATIONS

Karl Johan Åström

Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

Wolfgang Kreutzer

Department of Computer Science
University of Canterbury
Christchurch, New Zeeland

Abstract

The representation of systems is a key issue in system theory and in computer aided control engineering. This paper discusses different ways to represent systems and suggests an approach based on object-oriented programming which admits hierarchical descriptions of system structure and behaviour. A prototype implementation of the ideas is described together with experiences of using it.

1. INTRODUCTION

The notion of system is an essential element of control theory. The representation of systems is also a key issue in computer aided control engineering (CACE). Systems can be represented in many different ways. There are graphical representations like block diagrams, signal flow diagrams and bond graphs. There are also mathematical representations like state space models and input-output relations which come in many different forms, matrix fractions, impulse responses, frequency responses. When working with control systems it is frequently useful to use several different representations of a system.

Only fairly primitive ways of representing systems are used in current CACSD systems. Typical examples are the Matlab derivatives where systems are described by matrices. A slightly more sophisticated representation is used in the simulation language Simnon. This representation recognizes that a system has the properties, inputs, outputs and states. Simnon also allows a system to be described as an interconnection of subsystems. However only flat interconnections are allowed.

This paper presents a more flexible way to describe systems which is based on object-oriented programming. It is shown that a general structural description of hierarchically connected systems can be constructed from simple ingredients by making a system an object with the properties Name, Inputs, Outputs, Subsystems and Connections.

It is also necessary to add behavioural descriptions to obtain a useful tool. This is done by creating new objects which describe behaviour. A system can then inherit both structure and behaviour. Behaviour can be characterized in many different ways. A state description is one of the simpler alternatives. This can be covered by introducing the object StateBehaviour with the properties States, StateTransitionMap and OutputMap. The behavioural descriptions should also allow a given system to be described by models of different complexity. Apart from the detailed quantitative descriptions it is also useful to be able to deal with qualitative descriptions of behaviour.

The paper also describes a small prototype implementation which was designed to experiment with the ideas. This prototype which is written in Lisp admits hierarchical system representation and symbolic manipulation of the system descriptions. The experiments with the prototype indicate that the approach is one way towards implementation of powerful CACE systems.

The paper is organized as follows. Some system representations in current CACSD packages are described in Section 2. Requirements on system representations are given in Section 3. Sections 4 and 5 deal with representation of system structure and behavior. The prototype implementation is described in Section 7. Some conclusions are drawn in Section 8.

2. EXAMPLES

Some examples of system representations used in current CACSD packages are given in this chapter.

Matrix languages

Linear timeinvariant systems can be described using arrays. Such systems are conveniently handled in some matrix language like MATLAB, (Moler 1981) or its derivatives MatrixX, (Walker et al. 1982), CTRL-C, (Little et al. 1984). A system is represented as a matrix quadruplet in CTRL-C. In MatrixX it is represented as a system matrix and an integer which gives the order of the system. It is, however, clear that it is not sufficient to only have matrices. A detailed discussion of this is found in Åström (1984). There are a few more data types like polynomials and transfer functions in Blaise (Delebecque and Steer, 1985), Impact, (Rimvall and Cellier 1984) and Eagles, (Gavel et al 1986). A more sophisticated data structure for systems was used in the Lund packages (Åström 1985). Our experiences indicate that it would be very useful to have even more flexible concepts.

Simnon

The system description used in the simulation language Simnon, (Elmqvist 1977), includes system descriptions for continuous and discrete time systems. A continuous system corresponds to a state models described by an ordinary differential equation like

$$\begin{aligned} \frac{dx}{dt} &= f(x, u, t) \\ y &= g(x, u, t) \end{aligned} \tag{1}$$

where x is the state vector, u the input vector and y the output vector. The Simnon representation is

```
CONTINUOUS SYSTEM <system identifier>
INPUT <list of inputs>
OUTPUT <list of outputs>
STATE <list of states>
DER <list of derivatives>
TIME <variable>
Computation of outputs
Computation of derivatives
Parameter assignment
Initial value assignment
END
```

The standard state space model for a discrete time system is

$$\begin{aligned} x(t_{k+1}) &= f(x_k, u_k, t_k) \\ y(t_k) &= g(x_k, u_k, t_k) \end{aligned} \tag{2}$$

where $\{t_k\}$ is a sequence of sampling points. In Simnon such a system is described as

```
DISCRETE SYSTEM <system identifier>
INPUT <list of inputs>
OUTPUT <list of outputs>
STATE <list of states>
NEW <list of new states>
TIME <variable>
TSAMP <variable>

Computation of outputs
Computation of new values of the states
Update the TSAMP-variable
Modify states in continuous subsystems
Parameter assignment
Initial value assignment
END
```

Notice that this description is analogous to continuous systems. There is however a new variable TSAMP which gives the next time that the system should be sampled. In Simnon it is also possible to connect subsystems by using a connecting system which is described by

```
CONNECTING SYSTEM <system identifier>
TIME <variable>
Computation of inputs
Parameter assignment
END
```

The notation in Simnon is very natural for a control engineer. Long experience of using it has also shown that it is very easy to teach and use.

Discussion

The matrix based languages lack a proper system concept. This means that it is difficult to implement operations which are naturally viewed as operations on a system. We cannot make the natural abstractions used in system theory. Low level matrix operations have to be used instead. Simnon has a notion of systems. A drawback with this notion is however that it only admits flat interconnections. Particularly when dealing with large systems it would be desirable to have an hierarchical interconnection of sybsystem. One possibility to do this was suggested in Åström (1985). A more flexible approach is suggested in this paper. Since Simnon is a simulation language there are also only a limited number of system operations which are supported.

3. REQUIREMENTS

We will now briefly discuss some key issues in system representation. An important requirement is that the descriptions introduced should admit hierarchical system representations. Another is that it should be convenient to express systems composed of regular patterns of similar components in a convenient way.

To discuss suitable system descriptions we must also know how they are typically used. Typical operations on a system may be to:

Combine several subsystems into a new subsystem.
Expand a system into its subsystem.
Find interconnected loops.
Compute steady state operating points.
Compute steady state input-output relations.
Simulate.
Linearize.
Describe region of validity of linearized model.
Analyse stability, reachability and observability.
Make a Kalman decomposition.
Compute system inverses.
Compute sensitivity functions.
Compute well-conditioned linear representations.
Find linear characteristics like:
  poles and zeros,
  transfer functions,
  frequency curves.
Transform system representations.
Perform and validate control design.
Make model reductions.
Fit parameters to experimental data.
Find graphical representations.

Some of these operations are conveniently done numerically. Others require formula manipulation. It is therefore essential that the system can support numerical as well as formal calculations.

To describe systems it is thus necessary to have a rich structure which makes it possible to describe hierarchical interconnections of subsystems where each subsystem in turn is composed of subsystems. The subsystems may be of different types. They may be described in terms of state models, as input-output relations like impulse responses or transfer functions. We also have a need for descriptions of different complexity.

4. SYSTEM STRUCTURE

Representation of system structure is a key element when dealing with complex systems. Graphical representations, like block diagrams, signal flow diagrams and bond graphs are common for this purpose. They can be used to present details of subsystems as will as to give an overview of systems. In a block diagram description a subsystem is represented by a box and interconnections by lines between the boxes. See Figure 1. A line can represent a simple connection which tells that the variables at the connections are the same. It can also represent a more complex situation where several variables are involved. It is common practice to introduce arrows to indicate causality when this is possible. There may also be special symbols to denote simple operations like addition and multiplication of signals. There are many related descriptions like signal flow graphs and bond graphs.

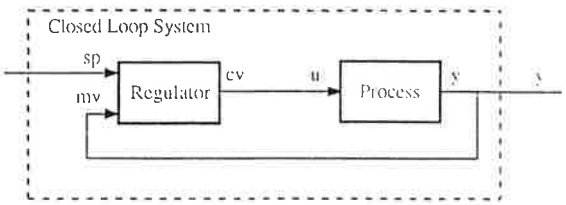


Figure 1. Example of a hierarchical block diagram. The ClosedLoopSystem is composed of two subsystems Process and Regulator.

To capture descriptions like block diagrams it is necessary to introduce the notions of subsystems and interconnections. In this paper we will only consider systems with well defined inputs and outputs. Such a system can be regarded as an abstraction of a box with input and output terminals. It can be represented as an object with the variables

```
Name
Inputs
Outputs
Subsystems
Connections
```

The inputs and outputs can be simple variables but they could also be objects with properties like units, range, etc. A primitive connection is a pair of input and output terminals. The connection implies that the corresponding terminal variables are the same. To avoid ambiguity the name of the associated system is also given. The notation (Regulator sp) denotes the terminal sp of the regulator. The system in Figure 1 can be represented as follows

```
Name ClosedLoopSystem
Input r
Outputs y
Subsystems Regulator Process
Connections r (Regulator sp)
(Regulator cv) (Process u)
(Regulator mv) (Process y)
y (Regulator y)
```

The regulator has the representation

```
Name Regulator
Input mv sp
Outputs cv
Subsystems nil
Connections nil
```

It thus has two inputs, mv the measured value and sp the set point. It has one output the controlled variable cv. The regulator has no subsystems and consequently no connections. In such a case the corresponding variables will be not be given.

The process in Figure 1 has the representation

```
Name Process
Input u
Outputs y
```

This notation is simple, natural and quite powerful.

Methods

A system structure has a number of associated operations. Examples of basic low level constructor and destructor function are

```
MakeSystem
AddInputs
DelInputs
```

Basic query and selector functions of the type

```
Inputs?
Inputs
```

are also needed to work with a system structure. The function `Inputs?` returns true if the subsystem has inputs and the function `Input` returns all inputs to a given subsystem. These functions operate on one level only. There are also primitive display functions like

```
ShowSystem
```

and a system editor which admits a structured editing of a system.

The functions discussed so far relate to a particular system only. For a system with subsystems it is also of interest to find all attributes of the system and of all associated subsystems. This is done by the functions

```
AllInputs
AllOutputs
AllSubsystems
AllConnections
```

It is sometimes also desirable to show the attributes hierarchically corresponding to the the subsystem hierarchy. This is done by the function.

```
HierarchyOfInputs
HierarchyOfOutputs
HierarchyOfSubsystems
HierarchyOfConnections
```

Several functions are useful in order to explore the structure of a system. There are some auxiliary functions which are useful to explore the connections. The functions

```
InputsConnectedTo
OutputsConnectedTo
```

return the systems which are connected to the input and output terminals of a given system. The function

```
ContainedIn
```

gives all the systems which contain a given system as a subsystem.

A loop is a closed path obtained by scanning a connection of subsystems in the direction defined by the input-output causality. The functions

```
Loop?
Loop
AllLoops
```

tell if a given subsystem is contained in a loop, gives a loop associated with a given subsystem and all loops associated with a given system. These functions can be used to trace a given collection of subsystems.

System Operations

It is convenient to have a number of operations which act on a system and generate new systems. This is accomplished by the functions

```
Aggregate
Disaggregate
```

The function `Aggregate` applies to a collection of subsystems and gives a new subsystem. The appropriate connections are generated from the aggregated subsystems.

There are also a number of other system operations which are useful to form composite systems from simple ingredients. Typical examples are

```
Invert
ParallelConnect
SeriesConnect
FeedbackConnect
```

These operations will have to operate on many different properties of a system. They will create new system with the appropriate properties.

5. SYSTEM BEHAVIOR

Only topological properties of a system i.e. structure and interconnections, have been discussed so far. To describe a system it is also necessary to describe how it behaves. Systems behavior is a very rich field. Examples of categories of behavior are

```
Static
Qualitative
StateSpace
StochasticStateSpace
StochasticInputOutput
LinearStateSpace
TransferFunction
TransientResponse
DescribingFunction
```

These categories can be described as objects. The static behavior can be described by a nonlinear function. Several methods are needed to work with static behavior e.g.

```
FindOutput
FindInput
Linearize
MaxGain
MinGain
DegreeOfLinearity
OperatingRange
```

Qualitative behavior attempts to describe some gross properties of a system like gain, time constants, and estimate of largest dynamic gain, some measure of nonlinearity, and some measure of how deterministic a system is. Associated with these properties we also need methods to obtain these properties from the more detailed representations. The ideas developed for automatic tuning of regulators are quite useful for this purpose. See Åström and Hägglund (1984). It is quite useful to allow qualitative system descriptions because it allows qualitative reasoning about system properties. In large systems composed of many subsystems we may e.g. neglect an interconnection if a system with a very low gain is connected in parallel with a system with a very high gain. In a simulation where we are exploring phenomena in a time scale of minutes we may use static models for subsystems whose time constants are less that one tenth of a second.

The state space and the transfer function behavior are well described in standard texts on control engineering. In this paper we will only use some simple forms of nonlinear state space behavior.

It is also very useful to introduce the `ValidityRange` property to indicate the region of validity of the model. This can be described as a subset of the product of the input spaces and the state spaces. With such a feature it is possible to write a simulation program which will raise an exception if the state of the system goes outside the region of validity during a simulation.

6. A PROTOTYPE IMPLEMENTATION

A small prototype program was written to test the ideas described in the previous sections. The program was written in ExperLisp on the Macintosh. The main goal was to experiment with descriptions of system structure. A secondary goal was to try some formula manipulation. The prototype can only handle a simple type of system behavior namely continuous time state model behavior. The version of ExperLisp that we used does unfortunately only support object-oriented programming to a limited degree. Our implementation was based on the `defstruct` function which can be used to generate a structure.

A system was implemented as structure with seven named components (slots)

```
Name
Inputs
Outputs
Subsystems
Connections
States
Behavior
```

The first five slots are used to describe system structure and the last two to describe system behavior. The system behavior is characterized in terms of the functions f and g in equation (1). The function f gives the rate of change of the state and the function g is the output map. These functions are given as Lisp functions.

The prototype has primitive constructor, destructor, query, selector and display functions for the system properties. The higher level operations include functions like

```
ShowSubsystemHierarchy
GetAllSubsystems
```

for all system variables. There are also operations to explore the system structure and tools for linearization of system behavior.

An Example

Consider the model following control system shown in the block diagram in Figure 2. The system named S1 has three subsystems Model, FF and S2. The system S2 has also three subsystems Reg, Proc and Sensor. The inputs, outputs and subsystems are shown in the figure. The connections can also be read from the figure. State variables are introduced to describe the behavior of the systems. These are also shown in the figure. The behavior of each system is described by the functions f and g in the model (1). The properties of the subsystems are given below.

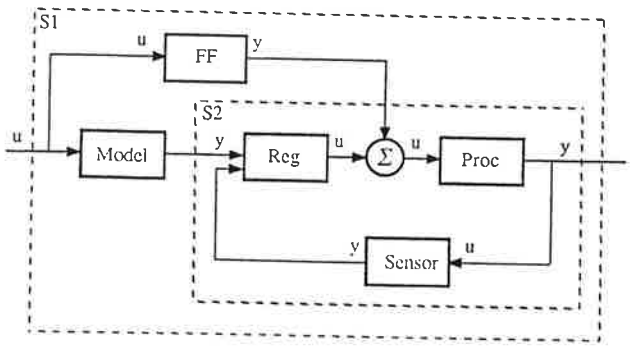


Figure 2. Block diagram of a model following control system.

Model Desired closed behavior.

```
Input u
Output c
States x1 x2
Behaviour
```

$$\frac{dx_1}{dt} = \omega x_1$$

$$\frac{dx_2}{dt} = \omega(-x_1 - 2\zeta x_2 + c)$$

$$y = x_1$$

FF Feedforward compensation.

```
Input u
Output y
State x
Behaviour
```

$$\frac{dx}{dt} = -ax + (b - a)x$$

$$y = k(x + u)$$

Reg Simple PID regulator.

```
Input r y
Output u
State i d
Behaviour
```

$$\frac{dd}{dt} = \frac{N}{T_d}(y - d)$$

$$\frac{di}{dt} = \frac{1}{T_i}(r - y)$$

$$u = k(r - y + i + N(d - y))$$

Process Process dynamics.

```
Input u
Output y
State x1 x2
Behaviour
```

$$\frac{dx_1}{dt} = -a\sqrt{x_1} + bu$$

$$\frac{dx_2}{dt} = a\sqrt{x_1} + c\sqrt{x_2}$$

$$y = x_2$$

Sensor Sensor dynamics.

```
Input u
Output y
State x
Behaviour
```

$$\frac{dx}{dt} = \frac{1}{T}(u - x)$$

$$y = \frac{x}{1 + x^2}$$

In our prototype we have not put much effort into the user interface. We are simply using the ordinary Lisp functions.

Sample Dialog

The simple dialog below illustrates how the system works. The function call (ShowSystem S1) generates the following listing of the description of the system S1

```
==============================
SYSTEM: S1
==============================
Inputs :  (c)
Outputs :  (y)
States :  nil
*** Subsystems ***
++++++++++
(S2 Model FF)
*** Connections ***
++++++++++
(c S1)
(S1 y)
*** Behaviour ***
+++++++++
+++ no state equations defined +++
+++ no output equations defined +++
nil
```

There are several tools to explore the system structure. The function call

(AllSubsystems S2)

returns all subsystems of S2 i.e.

```
(Reg Proc Sensor)
```

If the function is applied to a system whose subsystems also have subsystems all lower level subsystems are returned. An example is

(AllSubsystems S1)

which returns

```
(S2 Model FF Reg Proc Sensor)
```

The function AllSubsystems AllSubsystems can also be applied to a list of several systems as in

(GetAllSubsystems '(S1 S2))

which returns

```
(S2 Model FF Reg Proc Sensor)
```

The function GetSubsystemHierarchy can be used to get the detail of the subsystem structure. An example is

(GetSubsystemHierarchy '(S1))

which returns

```
(S1 (S2 Model FF) S2 (Reg Proc Sensor) Reg nil
Proc nil Sensor nil Model nil FF nil)
```

To list the hierarchy of subsystems we can use the function

(ShowSubsystemHierarchy S1)

which generates

```
++++++++++++++++++++++++++++++++++++++++
Subsyststems of:  S1
++++++++++++++++++++++++++++++++++++++++
S2
  Reg
  Proc
  Sensor
Model
FF
nil
```

There are similar functions to investigate the other system variables. The function call

(AllStates 'Model)

returns all states of a system i.e.

```
((x1 x2) nil nil nil)
```

and the function

(GetAllStates '(Reg Proc Sensor))

returns all states of a list of systems

```
((i d) nil ((x1 x2) nil ((x) nil nil)))
```

The function call

(TableOfConnections Proc)

returns the table of connections for the system Proc i.e.

```
((u Proc)(Proc y))
```

The connections are printed using

(ShowConnectionTable (TableOfConnections Proc))

which generates

```
(u Proc)
(Proc y)
nil
```

The function call

(StateEqns (Behaviour proc))

returns the function f which gives the rate of change of the state, i.e.

```
(((- (a * (x1 ** (1 / 2 )))) + (b * u))
((a * (x1 ** (1 / 2 ))) - (a * (x2 ** (1/ 2 )))))
```

The output map g of the system proc is obtained by

(OutputEqns (Behaviour Proc))

which returns

```
((x2))
```

The functions f and g which give the behaviour of the system are listed by the function call

(ShowBehaviourList (Behaviour Proc))

which results in

```
State-Eqns
----------
((- (a * (x1 ** (1 / 2 )))) + (b * u))
((a * (x1 ** (1 / 2 ))) - (a * (x2 ** (1 / 2 ))))
Output-Eqns
-----------
(x2)
nil
```

A system can be linearized through

(ShowBehaviourTable (Linearize Proc))

The system is then first linearized using the the function Linearize which generates a list of linearized equations. This list is then printed using the command ShowBehaviourTable. The result is as follows

```
System - Matrix
---------------
(a * ((1 / 2 ) * (x1 ** ((1 / 2 ) - 1 )))) 0;
(a * ((1 / 2 ) * (x1 ** ((1 / 2 ) - 1 ))))
(- (a * ((1 / 2 ) * (x2 ** ((1/ 2 )- 1 )))))
Input - Matrix
--------------------------
b
0
Measurement - Matrix
--------------------
0 1
nil
```

7. CONCLUSIONS

The main purpose of this paper has been to explore new ways to describe interconnected systems using object-oriented programming. It is relatively easy to implement a system like the prototype in Lisp. Implementation is much easier in a programming environment which supports object-oriented programming with a powerful inheritance mechanism.

Our experiences indicate that the system descriptions proposed are natural and easy to work with. A complete system for describing system structure can be implemented following the ideas outlined in the paper. A few additional functions are needed to explore system structure. The extension to systems with bidirectional interaction is straightforward. The user interface can be improve considerably by incorporating graphics of the type described in Elmqvist and Mattsson (1986). Work along these lines is under way.

Much more work is required to obtain a system which can describe system behavior in a reasonably complete way. Our experiments indicate that such systems are much easier to implement in an object-oriented environments like Flavors, Loops or Object Lisp which support inheritance.

Acknowledgements

The results of this paper were obtained as part of the project Computer Aided Control Engineering (CACE) at Lund Institute of Technology. We are grateful to the National Swedish Board of Technical Development (STU) who has supported this project under contract 85-4808. We also would like to thank Sven Erik Mattsson and Karl Erik Årzén for many useful discussions.

REFERENCES

ÅSTRÖM, K.J. (1983): "Computer Aided Modelling, Analysis and Design of Control Systems—A perspective," *IEEE Control Systems Magazine*, Vol. 3, No. 2, May 1983, 4–16.

ÅSTRÖM, K.J. and HÄGGLUND, T. (1984): "Automatic Tuning of Simple Regulators with Specifications on Phase and Amplitude Margins," *Automatica* **20**, No. 5, 645–651.

ÅSTRÖM, K.J. (1985): "Computer Aided Tools for Control System Design—A perspective," in M. Jamshidi and C.J. Herget (Eds.): *Computer-Aided Control Systems Engineering*, North-Holland.

DELEBECQUE, F. AND S. STEER (1985): "The Interactive System Blaise for Control Engineering," CADCE '85, 3rd IFAC/IFIP Symposium on Computer Aided Design in Control and Engineering Systems, Lyngby, Denmark.

ELMQVIST, H. (1975): "SIMNON - User's Manual," Report CODEN: LUTFD2/TFRT-3091, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. (1978): "A Structured Model Language for Large Continuous Systems," Report CODEN: LUTFD2/TFRT-1015, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. (1985): *LICS - Language for Implementation of Control Systems*, Report CODEN: LUTFD2/TFRT-3179, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. AND S.E. MATTSSON (1985): "A Simulator for Dynamical Systems Using Graphics and Equations for Modelling," *Proc. of the IEEE Control Systems Society Third Symposium on Computer-Aided Control System Design (CACSD)*, Arlington, Virginia.

GAVEL, D.T., C.J. HERGET, AND B. S. LAWYER (1986): "The M Language—An Interactive Tool for Manipulating Matrices, Systems and Signals.," Dynamics and Controls Group, Engineering Research Division, Lawrence Livermore National Laboratory, Livermore, California.

INTEGRATED SYSTEMS, INC (1984): "Matrix-X Users's Guide, Matrix-X Reference Guide, Matrix-X Training Guide, Command Summary and On-line Help," Integrated Systems, Inc. 101 University Avenue, Palo Alto, California.

LAWVER, B. and P. POGGIO (1985): "EAGLES Requirements," Computer Systems Research Group, Engineering Research Division, Lawrence Livermore National Laboratory, Livermore, California.

LITTLE, J.N., A. EMAMI-NOEINI, and S.N. BANGERT (1984): "CTRL-C and Matrix Environments for the Computer-Aided Design of Control Systems," in Bensoussan and Lions (Eds.): *Analysis and Optimization of Systems*, Lecture Notes in Control and Information Sciences, Springer-Verlag, Berlin, Also in Jamshidi, M. and C.J. Herget (eds) Computer-Aided Control Systems Engineering, North-Holland, Amsterdam, The Netherlands, 111–124.

MOLER, C.B. (1980): "MATLAB - User's Guide," Department of Computer Science, University of New Mexico, Albuquerque, USA.

RIMVALL, M. (1983): "IMPACT, Interactive Mathematical Program for Automatic Control Theory, User's Guide," Department of Automatic Control, Swiss Federal Institute of Technology, ETH-Zentrum, Zurich, Switzerland.

RIMVALL, M and F. CELLIER (1984): "IMPACT Interactive Mathematical Program for Automatic Control Theory," in Bensoussan and Lions (Eds.): *Analysis and Optimization of Systems*, Springer Lecture Notes in Control and Information Sciences, Springer, Berlin.

RIMVALL, M. and F.E. CELLIER (1985): "A Structural Approach to CACSD," *Computer-Aided Control Systems Engineering*, North-Holland, Amsterdam, The Netherlands, pp. 149–158, In Jamshidi, M. and C.J. Herget (Eds.).

SHAH, S.C., M.A. FLOYD AND L.L. LEHMAN (1985): "MATRIX-X: Control and Model Building CAE Capability," in Jamshidi, M. and C.J. Herget (Eds.): *Computer-Aided Control Systems Engineering*, North-Holland, Amsterdam, The Netherlands, pp. 181–207.

SYSTEMS CONTROL TECHNOLOGY (1984): "CTRL-C A Language for the Computer-Aided Design of Multivariable Control Systems, Users Guide," Systems Control Technology, 1801 Page Mill Road, Palo Alto, CA.

VANBEGIN, M. and P. VAN DOOREN (1985): "MATLAB-SC, Appendix B: Numerical Subroutines for Systems and Control Problems," Technical Note N168, Philips Research Laboratories, Bosvoorde, Belgium.

WALKER, R., C. GREGORY, JR. AND S. SHAH (1984): "MATRIX-X: A Data Analysis, System Identification, Control Design and Simulation Package," Integrated Systems, Inc., Palo Alto, California.

WALKER, R., S.C. SHAH AND N.K. GUPTA (1984): "Computer-Aided Engineering (CAE) for System Analysis," *Proc. of the IEEE*, **72**, No. 12, 1732–1745.

An Expert System Interface for Idpac

Jan Eric Larsson
and
Per Persson

Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

A project in the Computer-Aided Control Engineering project
sponsored by
the National Swedish Board for Technical Development

2

Presentation of the problem

- A goal related help facility
- Keep the command dialog
  The command spy
  Scripts + rules
- Data base of files, automatic documentation

Three kinds of knowledge is involved,

- Idpac and its commands
- System identification
- Data files and documentation

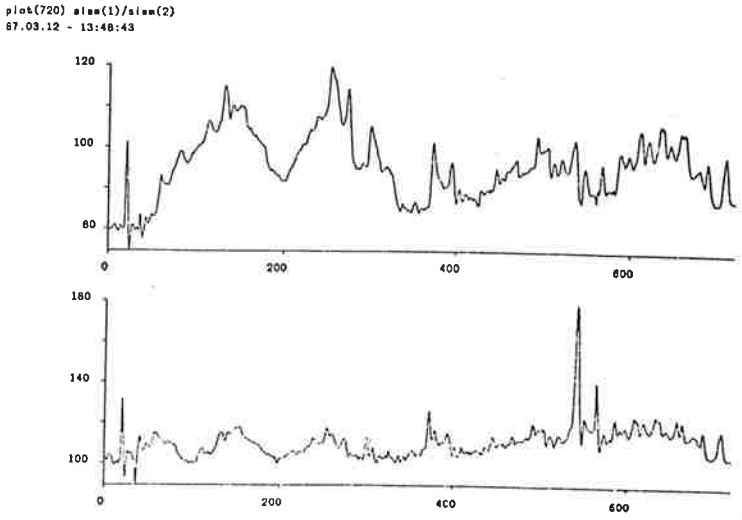
3

The Idpac commands

conv	edit	format	list
move	bode	hcopy	plmag
plot	acof	ccof	conc
cut	insi	pick	sclop
slide	stat	trend	vecop
aspec	cspec	dft	frop
idft	deter	dsim	filt
ranpa	resid	sptrf	ls
ml	sqr	struc	delet
fhead	ftest	turn	stop

```
conv of < if (1 3) 2 2
resid rs < sy is 3
```

Running Idpac can be difficult
Help gives only name and syntax

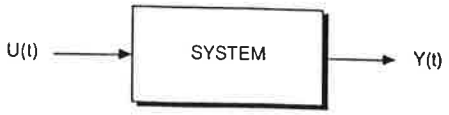


4

```
Directory USE:[PELP.IDENTIFY]

BODEMOD.T;1          COMPLOSS.T;2         CR1.D;1              CR2.D;1
CR3.D;1              CR4.D;1              CR5.D;1              CRVAL.T;16
D1.D;1               D10.D;1              D100.D;1             D14.D;1
D200.D;1             D24.D;1              D34.D;1              D44.D;1
D5.D;1               D50.D;1              DC14.D;1             DC24.D;1
DC34.D;1             DC44.D;1             DC54.D;1             DF14.D;1
DF24.D;1             DF34.D;1             DF44.D;1             DML1.D;1
DML14.D;1            DML2.D;1             DML24.D;1            DML3.D;1
DML34.D;1            DML4.D;1             DML44.D;1            DML5.D;1
DML6.D;1             DML7.D;1             DML8.D;1             FLS1.D;1
FLS10.D;1            FLS100.D;1           FLS5.D;1             FLS50.D;1
FML1.D;1             FML10.D;1            FML100.D;1           FML200.D;1
FML5.D;1             FML50.D;1            FMLI1.D;1            FMLI10.D;1
FMLI100.D;1          FMLI5.D;1            FMLI50.D;1           FMODLS.D;1
FR1.D;1              FR2.D;1              FR3.D;1              FR4.D;1
GENHAC.T;4           GENSIG1.T;7          GENSIG2.T;5          IDENTI.T;5
IDENTIF.T;6          IDPAC.SPY;42         IDPAC.T;2            L1.T;1
LS10.T;1             LS100.T;1            LS5.T;1              LS50.T;1
LSIBID.T;12          LSIR.D;1             MODYSIG.T;6          ML1.T;2
ML10.T;1             ML100.T;1            ML2.T;1              ML200.T;1
ML24.T;1             ML3.D;1              ML3.T;2              ML34.T;1
ML4.T;1              ML44.T;1             ML5.T;2              ML50.T;1
MLC14.T;1            MLC24.T;1            MLC34.T;1            MLC44.T;1
MLC54.T;1            MLF14.T;1            MLF24.T;1            MLF34.T;1
MLF44.T;1            MLI1.T;1             MLI10.T;1            MLI100.T;1
MLI5.T;1             MLI50.T;1            MLID.T;7             MLIDI2.DS;1
MLIID.T;6            MLX.D;1              MODLS.T;1            PCID.T;4
PFID.T;16            PID.T;10             R134.T;1             R1TFML34.D;3
R234.T;1             R2TFML34.D;1         R334.T;1             R3TFML34.D;1
R434.T;1             R4TFML34.D;1         RAND.T;2             RE5.T;9
RESI.T;2             RESULT.T;1           RTFML3.D;1           S1.D;1
S10.D;1              S100.D;1             S200.D;1             S5.D;1
S50.D;1              TEST.D;1             TFML2.D;1            TFML24.D;1
TFML3.D;1            TFML34.D;1           W4.D;1               WORK4.D;1
WRK.D;1              WRK1.D;1             WWRK.D;1             X.T;3

Total of 140 files.
```

Given input $u(t)$ and output $y(t)$

Try to estimate the parameters of the model, e.g., a transfer function

- Experiment design
- Data validation
- Estimation
- Validation
  Bode plots
  Simulation
- Cross validation

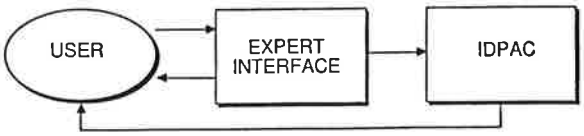
Rationale for the project

- The problem area should be well known
- Experts must be available
- The problem area should be easy to isolate
- The problem must reach over the "embarrassment level"
- The problem must be reasonably limited

Idpac is well fitted for the problem

- Simple command structure
- No complex control structures
- Simple data types

How does one combine a CAD program and an expert system?



Our solution

Use the expert system as a "command spy" interface in an intelligent help system.

- Eavesdrops on the user
- Gives help on demand only
- Gives help depending on previous commands; a dynamic help system

Different kinds of knowledge

- Knowledge of methods and procedures
  Command sequences
- Diagnostic and interpretative knowledge
  Interpretation of loss functions, AIC, &c.

Knowledge representation of procedures

- Finite state machine implemented with rules only
  State stored in the fact database
- A grammar for running Idpac
  A language to describe command sequences

The script language

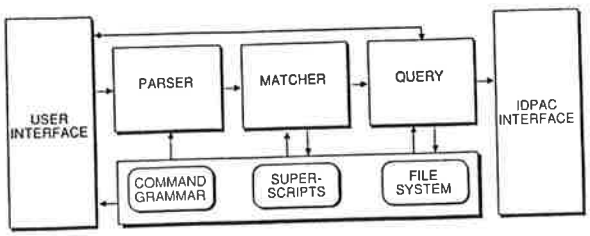
(<clauses>...)
(command ...)
(assign *new old*)
(kscall < *fact* >)
(scriptmacro *name inparam outparam*)
(repeat (...))
(repeat* (...))
(or (...) ...)
(all (...) ...)

An example

```
((command conv (outfile DATA-T) (infile WORK))
 (command plot (infile DATA-T))
 (or
  ((command trend (outfile DATA) (infile DATA-T)))
  ((command cut (outfile DATA) (infile DATA-T))))
 (kscall (fact trends removed from (parameter DATA)))
 (command plot (infile DATA)))
```

A script is matched against the actual command history, incremental or from scratch for each command.

Implementation of the system



Implementation tools

- Franz Lisp
- Flavors and YAPS (Maryland)
- C
- VAX 11/780 with VMS and Eunice

≈ 5000 lines of code

Transformations of a Command

Grammar specification of `conv`

```
conv dname < fnam [(c1...)] ncolx [tsamp]
```

Two scripts match the command sequence so far, `ml-script` and `corana-script`.

File bindings

```
(file-bindings (file-2 WRK) (file-1 WRK))
```

The user types

```
co
```

After the parser

```
(conv
 (outfile *unknown*)
 <
 (infile *unknown*)
 (numlist ALL idpac-default)
 (number *unknown*)
 (number delta idpac-default))
```

After the script matcher

```
(conv
 (outfile (ml-script (file-3 *unknown*)
                     (file-4 *unknown*))
          (corana-script (file-5 *unknown*)))
 <
 (infile  (ml-script (file-2 WRK)
                     (file-1 WRK))
          (corana-script (file-1 WRK)))
 (numlist ALL idpac-default)
 (number *unknown*)
 (number delta idpac-default))
```

After the query module

```
(conv
 (outfile (ml-script (file-3 DATA)
                     (file-4 DATA))
          (corana-script (file-5 DATA)))
 <
 (infile (ml-script (file-2 WRK)
                    (file-1 WRK))
         (corana-script (file-1 WRK)))
 (numlist ALL idpac-default)
 (number 1)
 (number delta idpac-default))
```

To the Idpac interface

```
(conv DATA < WRK 1)
```

13 B

The matching of a script

```
(((command x1)
  (kscall (A test call))
  (repeat
   ((or
     ((command x2)
      (command x3))
     ((command x4)))
    (command x5)))))
```

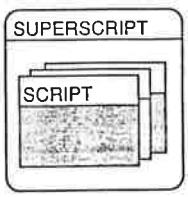
```
(((repeat
   ((or
     ((command x2)
      (command x3))
     ((command x4)))
    (command x5)))))
```

```
(((or
   ((command x2)
    (command x3))
   ((command x4)))
  (command x5)
  (repeat*
   ((or
     ((command x2)
      (command x3))
     ((command x4)))
    (command x5)))))
```

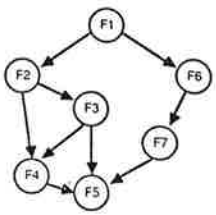
14 A

```
(((command x2)
  (command x3)
  (command x5)
  (repeat*
   ((or
     ((command x2)
      (command x3))
     ((command x4)))
    (command x5))))
 ((command x4)
  (command x5)
  (repeat*
   ((or
     ((command x2)
      (command x3))
     ((command x4)))
    (command x5)))))
```

The Database



- Superscripts



- File system
  May ask for relations between files
- Command grammar
- Miscellaneous

15

14 B

Low Level Interface Between Lisp and Idpac

```
#include <stdio.h>

struct stdescr
{
  int i;
  char *c;
};

char   txtnam[9] = "HELP_BOX";
struct stdescr txtbox;
int    txtchan;

int    io_writevblk = 48;
int    io_readvblk  = 49;

initialize()
{
  txtbox.c = txtnam;
  txtbox.i = 8;

  return(sys$crembx(0, &txtchan,
    0, 0, 0, 0, &txtbox));
}
```



```
sendwait (message, msglen)
char *message;
int *msglen;
{
  return(sys$qiow(0, txtchan, io_writevblk,
    0, 0, 0, message, *msglen, 0, 0, 0, 0));
}

sendnowait (message, msglen)
char *message;
int *msglen;
{
  return(sys$qio(0, txtchan, io_writevblk,
    0, 0, 0, message, *msglen, 0, 0, 0, 0));
}
```

Command procedure for starting Idpac

```
$ assign/user_mode help_box sys$input
$ run idpac
```


A Recipe for System Identification

1. Convert ASCII files to Idpac binary format and plot the data.
2. Cut the data files in two halves, and remove biases.
3. Check that the coherence between input signal and output signal is sufficiently large.
4. Estimate parameters. Check the residuals and transfer functions. Simulate the system.
5. Pick out data for cross validation.
6. Perform the cross validation.
7. STOP.

An example of a script for ML identification

```
((command plot (infile INSI) (infile OUTSI))
 (command coh (outfile COF) (infile INSI))
 (repeat
  ((command mlid (outfile SYST)
                 (infile INSI) (infile OUTSI) (number N))
   (kscall (Estimation of order (parameter N) performed))
   (or
    ((command residu (outfile RES)
                (infile SYST) (infile INSI) (infile OUTSI)))
    ((command sptrf (outfile FREQ) (infile SYST))
     (kscall (Recommend multi-Bode plot with
                (parameter FREQ)))
     (command bode (infile FREQ))))))
 (kscall (Give advice on most probable order)))
```

An Example Run

```
>plot in / out
```
A plot of the in and out files appears on the screen.
```
>??
Next command: coh
>coh tmp
```
A plot of the coherence appears.
```
>??
Next command: mlid
>mlid syst1 < in out 1  " Create first order model
```
A model printout appears.
```
>?
Look at the residuals of in and out, using the system syst1.
>residu                  " Arguments left out
 residu ... < syst1 in out " Some are defaulted
 Residuals outfile? > res1 " The system prompts for the last
```
A plot of the residuals appears.
```
>??
Next command: mlid or sptrf
>sptrf freq1             " Compute transfer function
>bode                    " Plot it
```
A Bode plot appears.
```
>ml syst2                " Create a new model
 mlid syst2 < in out ...  " Model order can't be defaulted
 Model order? > ?         " Further explanation
 The order of the transfer function
 to be fitted to the indata.
```



```
Model order? > 2
```
A model printout appears.
```
>res res2
```
A plot of the residuals appears.
```
>??
Next command: mlid or sptrf
>sptrf freq2 < syst2 b / a  " Compute a transfer function
>?
Perform the command BODE freq2 freq1.
>bode freq2 freq1
```
A Bode plot of the first and second order models appears.
```
>?
The model order is probably high enough when the Bode
plots start to coincide in the middle frequencies.
>stop
```


Further developments

- Window and mouse communication
- Redesign Idpac's command language
- Redesign the Idpac program structure
- Script and Rule verifier
- Automatic generation of file names
- Statistics of sessions
- Undo
- Error recovery

Conclusions

- It is possible to use an expert system and retain a command style dialog.
- In this case it is natural to represent knowledge with scripts *and* rules.
- It is possible to build an intelligent help system, and it has proved to be useful.
- The current implementation is not stable. Much work is required to make it a product for distribution.
- Building a major knowledge base remains to be done.

An Intelligent Help System for Idpac

Jan Eric Larsson M.Sc. & Per Persson M.Sc.
Department of Automatic Control
Lund Institute of Technology
Box 118, S-221 00 LUND, Sweden

Abstract

This paper describes an expert interface for system identification. It works as an intelligent help system for the CAD program Idpac using the *command spy* strategy. This means that the system uses the previous command history to understand what the user is doing and gives help according to this. *Scripts* are used to represent procedural knowledge and production rules for diagnostic knowledge. The system has been implemented and a small knowledge base developed. An example run with the system is shown.

Introduction

In order to use expert system techniques in a CAD program, several problems must be solved.

- CAD programs usually have a flexible command dialog. This way of communication should be retained when the expert system is added to the program.
- The expert system should be totally non-invasive, allowing the user to fall back on the plain CAD program in case it is not able to give the user any help.
- An inexperienced user often has a general idea of what he wants to do, but does not know exactly how to do it. The expert system should be able to guide the user from general ideas to specific commands, i. e. a goal related help facility is needed.
- An expert system used as an intelligent help system interface must be able to handle both procedural and diagnostic knowledge. We propose to represent procedural knowledge with scripts, and diagnostic knowledge with production rules.
- An intelligent help system should have facilities to teach the user about the target program and to transfer knowledge from the knowledge database to the user.

According to these design criteria an expert system interface has been developed. It contains a command parser, a script matching device with a database for scripts and rules, a query module, a file system, an on-line dictionary and interfaces to the user and Idpac.

Idpac is a command driven program for system identification, see [1,2,3]. This project was originally outlined in [4,5]. A previous system was described in [6]. A thorough description of this project is given in [7]. There are some other projects in the area of intelligent help systems, see [8]and [9].

System Identification

System identification is the process of finding a mathematical model of a real world process. It involves experiments in which input and output signals are measured, choice of a suitable model, validation of data, numerical fitting of model parameters to these data and several validation tests for the verification of the results. System identification demands skill and experience and the validity of the results strongly depends on the user's knowledge. For thorough readings on system identification, see [10,11]. An overview is found in [12].

Scripts

Running Idpac requires a lot of procedural knowledge, i. e. knowledge about sequences of commands, and, on a higher level, knowledge of sequences of sub-tasks of Idpac

sessions. In order to represent sequences, the concept of *scripts* was introduced. We were inspired by a data structure used in natural language understanding, [13, 14]. Scripts may be implemented in several ways, e. g. with production rules or as Lisp lists. The latter approach was taken in this project. Here is an example of a script.

```
((command plot (infile INSI) (infile OUTSI))
 (repeat
  ((command mlid (outfile SYST) (infile INSI) (infile OUTSI) (number N))
   (kscall (Estimation of order (parameter N) performed))
   (or
    ((command residu (outfile RES) (infile SYST) (infile INSI) (infile OUTSI)))
    ((command sptrf (outfile FREQ) (infile SYST))
     (kscall (Recommend multi-Bode plot with (parameter FREQ)))
     (command bode (infile FREQ))))))
 (kscall (Give advice on most probable order)))
```

This script performs a parameter estimation loop with increasing order of the fitted model. First the user should look at the signals with the PLOT command. The infiles INSI and OUTSI must already exist. A model is produced with MLID. The outfile SYST is created and N, the order of the model, is associated with the actual argument used. Next, a fact stating that a parameter estimation has taken place is sent to the production rule database. This is done with the KSCALL clause. Kscall stands for Knowledge Source Call and puts a fact in the fact database of the production rule system. After this, the user may either look at the residuals with the RESIDU command, or produce data for a Bode plot with SPTRF and plot it with BODE. The whole procedure may be repeated, and every time the rule system gives advice on whether the order is sufficiently high. During the process it may use facts put into the database by previous kscalls.

The Knowledge Based Command Spy

How does one combine a CAD program with an expert system while keeping the good features of both? The solution proposed in this paper is the *command spy* strategy.

The expert system is used as an interface to Idpac. In our solution it is placed before the command decoder of Idpac, but an alternative would be to build it into the outermost level of Idpac.

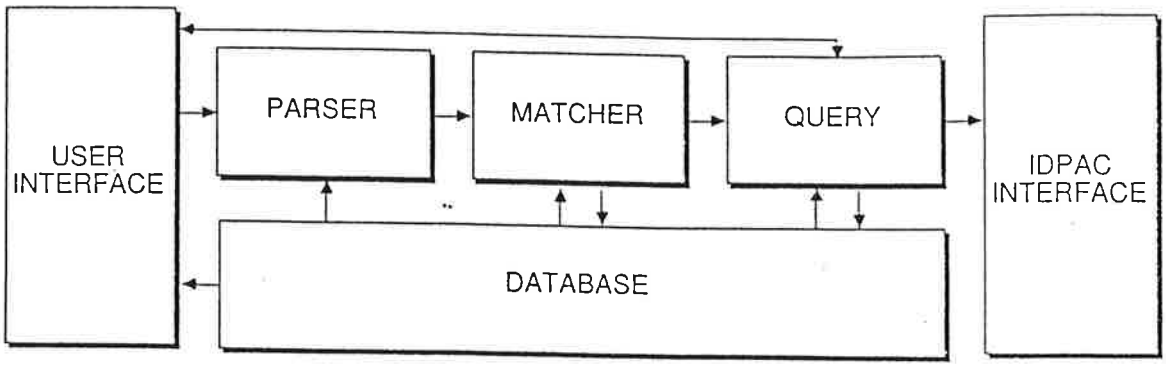
We keep the command language of Idpac. In this way a cumbersome Q/A dialog is avoided. The expert system traces the user without asking any questions, and gives help only on demand. Thus the expert system never forces a user to do anything. A user that does not need or want any help is not bothered and there is always the possibility of falling back on plain Idpac in case the help system does not have enough knowledge to work properly. The expert system may also use a Q/A dialog to find out facts about the experiment and expected results, but only if the user initiates one.

The command spy uses scripts in order to understand what the user is doing. By matching scripts against the actual command history, the expert interface is able to guess what the user wants to do. The scripts also provide information on the next step for reaching a desired goal.

Diagnostic reasoning is needed at certain points in an identification, notably when things go wrong or unexpected results arise. This is taken care of by production rules associated with each script. The rules also allow for automatic documentation by writing script based information to a text file.

Implementation

The expert interface is made up from several independent parts. Most of the parts work on a common database.



Layout of the system.

The user interface reads a command from the user and transforms it into a Lisp list. It provides all input and output functions used in other parts of the interface. In this way, all of the system's dependence on terminal types, graphics, etc. is collected in one place.

The command parser checks the command for syntax and supplies defaults in the same way that the parser of Idpac does. In this process it transforms the command into a more convenient form. This parser accepts commands with the arguments left out, as the other routines will fill information in, by defaulting from scripts and asking the user.

The script matcher keeps track of the scripts incrementally and updates them as it gets commands from the parser. The commands once again are transformed, and files may be defaulted using knowledge from the scripts.

The query module works through the command description and tries to fill in the remaining unknown entries by asking the user about them. In this way the user may give only the command name. If necessary he will then be prompted for all the parameters left out. The query module also sends messages to the file system about created files.

The database contains the command grammar used by the parser, the scripts and rules used by the script matcher, the file tree of the file system and state variables for keeping track of the user state, set a debug mode, etc.

The file system keeps track of all the data files created and used during an Idpac session. It does this by storing data about the files in a directed graph structure. This enables the file system to show e. g. the ancestors or descendants of a file.

The expert interface and Idpac reside in two different VMS processes. The Idpac interface sends the processed command to Idpac via a VMS mailbox. In this way no changes had to be done to the Idpac program itself. The interprocess communication routines are written in C.

The system is written in Franz Lisp extended with Flavors [15], and YAPS [16]. It consists of about 5000 lines of code and runs under VMS on a VAX 11/780.

An Example Run

This is an example of a run with the system. The script used is the one shown in the script section of this paper. A '?' asks for rule based information and '??' for the next command according to the script. Texts appearing after a '!' are comments.

```
>plot in / out
A plot of the in and out files appears on the screen.
>??
Next command: mlid
>mlid syst1 < in out 1          ! Create first order model.
A model printout appears.
>?
```

```
Look at the residuals of in and out, using the system syst1.
>residu                          ! The user leaves out all arguments.
 residu ... < syst1 in out       ! Some are defaulted from the script.
 Residuals outfile? > res1       ! The system prompts for the last one.
```
A plot of the residuals appears.
```
>??
Next command: mlid or sptrf
>sptrf freq1                     ! Compute a transfer function data file.
>bode                            ! Plot it.
```
A Bode plot appears.
```
>ml syst2                        ! Create a new model.
 mlid syst2 < in out ...         ! The model order cannot be defaulted.
 Model order? > ?                ! The user wants a further explanation.
 The order of the transfer function to be fitted to the indata.
 Model order? > 2
```
A model printout appears.
```
>res res2
```
A plot of the residuals appears.
```
>??
Next command: mlid or sptrf
>sptrf freq2 < syst2 b / a       ! Compute a transfer function data file.
>?
Perform the command BODE freq2 freq1.
>bode freq2 freq1
```
A Bode plot of the first and second order models appears.
```
>?
The model order is probably high enough when the Bode plots start to
coincide in the middle frequencies.
```

Conclusions

We have drawn several conclusions from the project. First, if one wants to combine an expert system with a CAD program, the expert system should be used as an intelligent and non-invasive help system. This retains the advantages of both the CAD program and the expert system. It may be accomplished by implementing a command spy, as outlined in this paper. Secondly, not all knowledge need be implemented with production rules. Scripts are a better way of representing sequences, especially in problems where both methods and goals are well known. A good rule is to use as much as possible of the structure of the problem in the solution. The use of scripts will also reduce the size of the knowledge databases considerably.

Acknowledgements

We would like to thank our supervisor Karl Johan Åström and our colleagues Sven Erik Mattsson, Lars Nielsen and Karl-Erik Årzén for supporting us in our work.

The project is part of the Computer-Aided Control Engineering project at the Department of Automatic Control, Lund Institute of Technology, supported by STU, the National Swedish Board for Technical Development, under contract no. 85–3042.

References

1. Wieslander, *Interaction in Computer-Aided Analysis and Design of Control Systems*, Doctoral thesis, TFRT–1019, Department of Automatic Control, Lund Institute of Technology, 1979.
2. Gustavsson, Nilsson, "Övningar för Idpac," Internal report, TFRT–7169, Department of Automatic Control, Lund Institute of Technology, Lund, 1979.
3. Gustavsson, "Några macros för Idpac,", Internal report, TFRT–7170, Department of Automatic Control, Lund Institute of Technology, Lund, 1979.

4. Larsson, *An Expert System Interface for Idpac*, Masters thesis, TFRT-5310, Department of Automatic Control, Lund Institute of Technology, 1984.
5. Larsson, Åström, "An Expert System Interface for Idpac," *Proceedings of the 2nd IEEE Control Systems Society Symposium on Computer-Aided Control System Design*, Santa Barbara, California, 1985.
6. Larsson, Persson, "Knowledge Representation by Scripts in an Expert Interface," *Proceedings of the 1986 American Control Conference*, Seattle, Washington, 1986.
7. Larsson, Persson, *An Expert System Interface for Idpac*, Licentiate thesis, Department of Automatic Control, Lund Institute of Technology, Lund. To appear.
8. Wilensky *et al*, "UC — A Progress Report," Report No. UCB/CSD 87/303, Computer Science Division (EECS), University of California, Berkely, California, 1986.
9. Waters, "KBEmacs — A Step Toward the Programmer's Apprentice," Technical report 753, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, 1985.
10. Eykhoff, *System Identification. Parameter and State Estimation.*, John Wiley & Sons, London, 1974.
11. Eykhoff, *Trends and Progress in System Identification*, Pergamon Press, Oxford, 1981.
12. Åström, "Modeling and Simulation Techniques," Agard Lecture Series No. 128, 1983.
13. Schank, Abelson, *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1977.
14. Schank, Riesbeck, *Inside Computer Understanding*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1981.
15. Allen *et al*, "The Maryland Artificial Intelligence Group Franz Lisp Environment," TR-1226, Department of Computer Science, University of Maryland, 1984.
16. Allen, "YAPS — Yet Another Production System," TR-1146, Department of Computer Science, University of Maryland, 1983.

COMBINATION OF FORMULA MANIPULATION AND NUMERICS

Ulf Holmberg

Motives for symbolic formula manipulation

- Structure information is very important
- Analytical expression may give good insights
- May eliminate numerical problem
- Automatic generation of code
- Education

SYMBOLIC FORMULA MANIPULATION IN MACSYMA

Linearization

$\dot{x} = f(x,u) \rightarrow \quad \dot{x} = Ax + Bu$

Stability analysis

- Routh's criterion
- Jury's criterion

Maximum Likelihood method

$E(\theta - \theta_o)^2$

Sampling

- Transfer function → Pulse transfer function

  $G(s) \overset{h}{\rightarrow} H(z)$

- State space continuous time → discrete time

  $(A, B, C, \tau) \overset{h}{\rightarrow} (\Phi, \Gamma_0, \Gamma_1, C)$

Polynomials

AX+BY=C "DAB equation"

GEOMETRY APPROACH (Time domain)

- Gauss elimimation PA=LU
- Kernel $\{X \mid AX = 0\}$
- Inverse Image $\{X \mid AX = B\}$ (A singular)
- Intersection of subspaces
- Gram-Schmidt (orthogonal base)
- A-invariant subspaces
- (A,B)-invariant subspaces

FACTORIZATION APPROACH (Frequency domain)

Polynomial matrices

- Smith form
- Smith-McMillan form
- Hermite form
- Matrix Fraction Decompositions, MFD
  - Right (Left) MFD
  - Irreducible Right (Left) MFD
- Column (Row) reduction (-proper)

Multivariable realizations → back to State space

EXAMPLE 1.

Linearize the following nonlinear system

$$\begin{cases} \dot{x}_1 = x_1^2 - u_1 x_2 \\ \dot{x}_2 = u_1 x_2^2 + u_2 \end{cases}$$

$$\begin{cases} y_1 = x_1^2 + u_2 x_1 \\ y_2 = x_2^3 - u_1 \end{cases}$$

around $\{x_0 = (1 \quad 1)^T, u_0 = (1 \quad -1)^T\}$ and determine the poles and zeros of the resulting multivariable linear system.

```
(c2) load("login.mac")$
(c3) f:matrix([x1*x1-x2*u1],[x2*x2*u1+u2]);
```

(d3)
$$\begin{bmatrix} x_1^2 - u_1 x_2 \\ u_1 x_2^2 + u_2 \end{bmatrix}$$

```
(c4) g:matrix([x1*x1+x1*u2],[x2^3-u1]);
```

(d4)
$$\begin{bmatrix} x_1^2 + u_2 x_1 \\ x_2^3 - u_1 \end{bmatrix}$$

```
(c5) x:matrix([x1],[x2])$
(c6) x0:matrix([1],[1])$
(c7) u:matrix([u1],[u2])$
(c8) u0:matrix([1],[-1])$
(c9) linearize(f,g,x,x0,u,u0);
```

(d9)
$$\left[a = \begin{bmatrix} 2 & -1 \\ 0 & 2 \end{bmatrix}, b = \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix}, c = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}, d = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

```
(c10) ss2tf(%);
```

(d10)
$$\begin{bmatrix} -\frac{s-1}{(s-2)^2} & \frac{(s-3)(s-1)}{(s-2)^2} \\ -\frac{s-5}{s-2} & \frac{3}{s-2} \end{bmatrix}$$

```
(c11) smith_mcmillan(%);
```

(d11)
$$\left[smith - mcmillan = \begin{bmatrix} \frac{1}{(s-2)^2} & 0 \\ 0 & (s-6)(s-1) \end{bmatrix}, mcmillan_degree = 2\right]$$

EXAMPLE 2.

Sample the following time-delayed system with a sampling period h and determine how the delay influences the discrete time system.

$$\dot{x}(t) = \begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u(t-\tau) \qquad \text{where} \quad \tau = ah \quad \text{and} \quad 1 < a < 2$$

Note line (c4) where we specify the variations of the parameter a.

```
(c1) load("login.mac");
```

(d1) *login.mac*

```
(c2) aa:matrix([0,1],[0,-1]);
```

(d2)
$$\begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}$$

```
(c3) bb:matrix([0],[1]);
```

(d3)
$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

```
(c4) assume(a > 1 AND a < 2);
```

(d4)
$$[a > 1, a < 2]$$

```
(c5) sampstate(aa,bb,a*h,h);
```

$$x(kh+h) = \Phi x(kh) + \Gamma_0 u(kh - dh + h) + \Gamma_1 u(kh - dh)$$

(d5)
$$\left[\Phi = \begin{bmatrix} 1 & 1-e^{-h} \\ 0 & e^{-h} \end{bmatrix}, \Gamma_0 = \begin{bmatrix} \left((2h-ah)e^{2h-ah} - e^{2h-ah} + 1\right)e^{ah-2h} \\ \left(e^{2h-ah} - 1\right)e^{ah-2h} \end{bmatrix},\right.$$
$$\left.\Gamma_1 = \begin{bmatrix} e^{h-ah}\left((ah-h)e^{ah-h} - e^{ah-h} + 1\right) + e^{h-ah}\left(1 - e^{ah-2h}\right)\left(e^{ah-h} - 1\right) \\ e^{-h}\left(e^{ah-h} - 1\right) \end{bmatrix}, d = 2\right]$$

```
(c1) load("login.mac")$
(c2) demo("realizations.dem");
/* This demo describes a couple of examples in Kailath, chapter 6.
Example 6.2-1. Alternative MFDs for a Transfer Function. p. 368-9.
Example 6.4-1. Controller-Form Realization of a Right MFD. p. 407-8.
Example 6.4-2. Observer-Form Realization of a Left MFD. p. 416
Example 6.4-6. Constructing Canonical Controllability Forms. p. 433-4. */
(c3) g:matrix([s/((s+1)*(s+2))^2,s/(s+2)^2],[-s/(s+2)^2,-s/(s+2)^2]);
```

(d3)
$$\begin{bmatrix} \frac{s}{(s+1)^2(s+2)^2} & \frac{s}{(s+2)^2} \\ -\frac{s}{(s+2)^2} & -\frac{s}{(s+2)^2} \end{bmatrix}$$

```
/* Example 6.2-1. Alternative MFDs for a Transfer Function. p. 368-9. */
(c4) rmfd(g);
```

(d4)
$$\left[dr = \begin{bmatrix} (s+1)^2(s+2)^2 & 0 \\ 0 & (s+2)^2 \end{bmatrix}, nr = \begin{bmatrix} s & s \\ -s(s+1)^2 & -s \end{bmatrix}\right]$$

```
(c5) ev(rightcoprime(dr,nr),%);
```

(d5)
$$\left[dr = \begin{bmatrix} (s+1)^2(s+2)^2 & -(s+1)^2(s+2) \\ 0 & s+2 \end{bmatrix}, nr = \begin{bmatrix} s & 0 \\ -s(s+1)^2 & s^2 \end{bmatrix}, rr = \begin{bmatrix} 1 & 1 \\ 0 & s+2 \end{bmatrix}\right]$$

```
(c6) ev(columnreduce(dr,nr),%);
```

(d6)
$$\left[dr = \begin{bmatrix} 0 & -(s+1)^2(s+2) \\ (s+2)^2 & s+2 \end{bmatrix}, nr = \begin{bmatrix} s & 0 \\ -s & s^2 \end{bmatrix}, u = \begin{bmatrix} 1 & 0 \\ s+2 & 1 \end{bmatrix}\right]$$

```
/* Example 6.4-1. Controller-Form Realization of a Right MFD. p. 407-8. */
(c7) ev(real:controller(dr,nr),%);
```

(d7)
$$\left[a = \begin{bmatrix} -4 & -4 & 0 & -1 & -2 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -4 & -5 & -2 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, b = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ -1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \end{bmatrix}\right.$$

```
/* Example 6.4-6. Constructing Canonical Controllability Forms. p. 433-4.
*/
/* Search by Crate 2 */
(c8) ev(controllability(a,b,c),%);
```

(d8)
$$\left[a = \begin{bmatrix} 0 & 0 & -2 & 0 & 0 \\ 1 & 0 & -5 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 \\ 0 & 0 & 2 & 0 & -4 \\ 0 & 0 & 1 & 1 & -4 \end{bmatrix}, b = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, c = \begin{bmatrix} 0 & 0 & 1 & 1 & -4 \\ -1 & 4 & -12 & -1 & 4 \end{bmatrix}\right.$$

```
/* Search by Crate 1 */
(c9) crate_nr:1;
```

(d9) 1

```
(c10) ev(controllability(a,b,c),real);
```

(d10)
$$\left[a = \begin{bmatrix} 0 & 0 & 0 & -4 & 2 \\ 1 & 0 & 0 & -12 & 5 \\ 0 & 1 & 0 & -13 & 4 \\ 0 & 0 & 1 & -6 & 1 \\ 0 & 0 & 0 & 0 & -2 \end{bmatrix}, b = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, c = \begin{bmatrix} 0 & 0 & 1 & -6 & 1 \\ -1 & 4 & -12 & 32 & -1 \end{bmatrix}\right.$$

```
/* Example 6.4-2. Observer-Form Realization of a Left MFD. p. 416 */
(c11) lmfd(g);
```

(d11)
$$\left[dl = \begin{bmatrix} (s+1)^2(s+2)^2 & 0 \\ 0 & (s+2)^2 \end{bmatrix}, nl = \begin{bmatrix} s & s(s+1)^2 \\ -s & -s \end{bmatrix}\right]$$

```
(c12) ev(leftcoprime(dl,nl),%);
```

(d12)
$$\left[dl = \begin{bmatrix} (s+1)^2(s+2)^2 & 0 \\ (s+1)^2(s+2) & s+2 \end{bmatrix}, nl = \begin{bmatrix} s & s(s+1)^2 \\ 0 & s^2 \end{bmatrix}, rl = \begin{bmatrix} 1 & 0 \\ -1 & s+2 \end{bmatrix}\right]$$

```
(c13) ev(rowreduce(dl,nl),%);
```

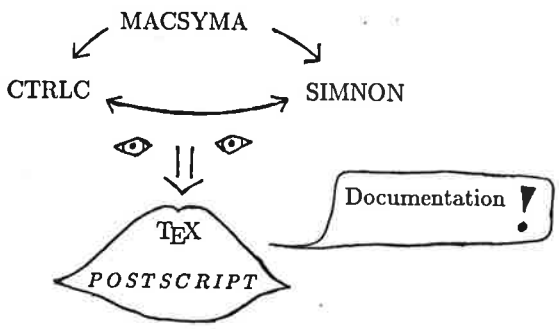
(d13)
$$\left[dl = \begin{bmatrix} 0 & -(s+2)^2 \\ (s+1)^2(s+2) & s+2 \end{bmatrix}, nl = \begin{bmatrix} s & s \\ 0 & s^2 \end{bmatrix}, u = \begin{bmatrix} 1 & -(s+2) \\ 0 & 1 \end{bmatrix}\right]$$

```
(c14) ev(observer(dl,nl),%);
```

(d14)
$$\left[a = \begin{bmatrix} -4 & 1 & 0 & 0 & 0 \\ -4 & 0 & 0 & 0 & 0 \\ 0 & 0 & -4 & 1 & 0 \\ 1 & 0 & -5 & 0 & 1 \\ 2 & 0 & -2 & 0 & 0 \end{bmatrix}, b = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, c = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \end{bmatrix}\right.$$

INTEGRATING SYMBOLIC AND NUMERICAL TOOLS FOR LINEAR SYSTEM ANALYSIS

Ulf Holmberg
Mats Lilja
Bengt Mårtensson



Presented at the SIAM Conference
on Linear Algebra, 12-14 Aug. 1986.

MACSYMA → Simnon, CTRL-C, TEXetc.

```
(c1) load("login.mac")$
(c2) a:matrix([1,2],[3,4])$
(c3) b:matrix([5,6],[7,8])$
(c4) c:matrix([9,10],[11,12])$
(c5) d:matrix([13,14],[15,16])$
(c6) sys:makesys(a,b,c,d);
```

(d6)

$$\left[a=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}, b=\begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix}, c=\begin{bmatrix}9 & 10\\ 11 & 12\end{bmatrix}, d=\begin{bmatrix}13 & 14\\ 15 & 16\end{bmatrix}\right.$$

```
(c7)
Break Entering Lisp:
<1>: (load "tomimo.l")
t
<1>: (tomimo $sys 'abcd.mim)
t
```

The MIMO system has now been written in the file `ABCD.mim`. This file looks as follows:

```
nmp =

2 2 2

a =

1 2
3 4

b =

5 6
7 8

c =

9 10
11 12

d =

13 14
15 16
```

Root Locus

Old Method

$$A(s) + kB(s) = 0$$

Solve for equidistant k-values

Mark the roots by an '×' for each k

Disadvantages

Time consuming

Bad resolution near multiple roots

Root Locus, (MACSYMA → Simnon)

New Method

$$f(s,k) = 0$$

Use MACSYMA to calculate

$$\frac{d}{dk}s_i(k) = -\frac{\frac{\partial f}{\partial k}}{\frac{\partial f}{\partial s}}$$

and multiple roots

Generate

Simnon system

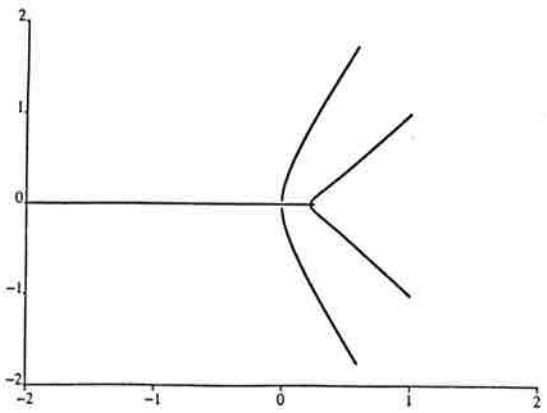
Intrac macro

Integrate and plot in Simnon

Root Locus—Example

Enter Simnon

```
> syst ode
> axes h -2 2 v -2 2
> rootloc
```



Root Locus—Example

Enter MACSYMA

```
(c1) load("rootloc.mac")$
(c2) g:matrix([1/s^2,1/s],[-1/s^2,0]);
```

(d2)
$$\begin{bmatrix} \frac{1}{s^2} & \frac{1}{s} \\ -\frac{1}{s^2} & 0 \end{bmatrix}$$

```
(c3) print_ode(g);
```

(d3) *ode.rl*

```
(c4) print_kxy(g,-2,2,100);
```

(d4) *rootloc.rl*

Conclusions

Combination of symbolic manipulation and numerics can give:

- insight of structural form and dependencies
- tools for more complex problems
- better understanding of theory
- interactive control design

The following has been made

◊ An analysis package for multivariable linear systems

◊ Programs for interaction between MACSYMA, CTRL-C, Simnon, TEX and Postscript

Suggestion for future work

→ A design package for multivariable linear systems

CODEN: LUTFD2/(TFRT-3186)/1-018/(1987)

Combination of Symbolic Manipulation and Numerics

Ulf Holmberg
Mats Lilja
Sven Erik Mattsson

STU project 85–4809

Department of Automatic Control
Lund Institute of Technology
March 1987

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

Document name
Report

Date of issue
March 1987

Document Number
CODEN: LUTFD2/(TFRT-3186)/1–018/(1987)

Author(s)
Ulf Holmberg
Mats Lilja
Sven Erik Mattsson

Supervisor

Sponsoring organisation
The National Swedish Board of Technical Development (STU contract 85-4809)

Title and subtitle
Combination of Symbolic Manipulation and Numerics

Abstract

Symbolic manipulation will play an important role in future tools for Computer Aided Control Engineering (CACE). The purpose of this project has been to experiment with and gain experiences of using a program for symbolic manipulation. As the tool we used the symbolic manipulation program MACSYMA and as the application analysis of multivariable systems was selected. A small package for analysis of multivariable linear systems is constructed inside MACSYMA. Manipulations of polynonial matrices are given as example. Also it is shown how it is possible to transfer results from MACSYMA to numerical programs, like the matrix-manipulation program CTRL-C and the differential equation program Simnon. By combining symbolic and numeric computations we can solve more complex and composit problems. This is demonstrated by solving for multivariable root locus.

Key words
Symbolic Manipulations; MACSYMA; Computer Aided Control Engineering; Multivariable Linear Systems; Polynomial Matrices.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
18

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

Contents

1. Introduction 1

2. Motives 2
Insight is Desirable 2
Support Users' Concepts 3
Multi-Purpose and Reusable Descriptions 3
Improve the Numerical Properties 4

3. An Analysis Package in MACSYMA 6
Available Functions 6
Example—Polynomial Matrix Manipulations 7

4. Generation of a MIMO System Text File 10

5. Root Locus 12
A General Problem 12
Implementation 12
An Example 13

6. Conclusions 14

Acknowledgements 15

References 15

1. Introduction

Symbolic manipulation will play an important role in future CACE tools. Unfortunately, today's systems for Computer Aided Control Engineering (CACE) allow basically the user to perform numerical calculations. They do not support symbolic calculations. One important reason for this is that the tools were designed for computers with what is today considered as moderate computing power and symbolic manipulation calls for computing power. The increasing capacity of computers and workstations makes it now worthwhile to introduce symbolic calculations in CACE systems.

[Pavelle et al., 1981] give a popular scientific introduction to computer algebra. There are commercial general-purpose systems for symbolic systems available:

MACSYMA	developed at the MIT Laboratory for Computer Science, USA
REDUCE	developed at Stanford University, the University of Utah and the Rand Corporation, USA
Scratchpad	developed by IBM
SMP	developed at the California Institute of Technology, USA
Maple	developed at the University of Waterloo, Canada
muMATH	developed by the Soft Warehouse, Honolulu

The main purpose of the project "Combination of Symbolic Manipulation and Numerics" was to experiment with and gain experiences of using a program for symbolic manipulation. As the tool we used MACSYMA and as the application analysis of multivariable systems was selected. There is a framework for analysis and design of multivariable systems using polynomial matrices. A standard text book is [Kailath, 1980]. Unfortunately, these methods have poor numerical properties. Methods based on state space representations have better numerical properties. However, in many cases it is desirable to be able to work in the frequency domain. It may be easier to formulate and analyse properties of interest in the frequency domain than in the state space. MACSYMA is good at polynomials and rational functions.

Motives for supporting symbolic calculations are presented in Chapter 2. The package developed in MACSYMA for analysis of multivariable linear systems is described in Chapter 3. A Lisp function in MACSYMA can be used to establish interaction with other programs, such as Simnon and CTRL-C. This is presented in Chapter 4. In Chapter 5 a new method for calculating root loci is demonstrated. This serves as an example of the idea to combine symbolic and numerical computation and thereby solve more complex and composite problems. Conclusions are given in Chapter 6.

2. Motives

There are several good reasons for including symbolic manipulation in CACE tools. First, structure is important and an analytic answer may give a better insight. Second, the user interface could be improved, since the user's original formulation is usually not a computational procedure but rather equations and relations on symbolic form. Third, models could be multi-purpose and reusable independent of what is known and what should be computed. Forth, symbolic manipulation could be used to facilitate numerical solution. Below we will discuss these motives in more detail.

Insight is Desirable

You may think that the ultimate CACE program is an automatic procedure which outputs a VLSI chip that implements the optimal controller. Life is not that easy. You must at least specify your desires and requirements; a specification of what **you** think is optimal. Unfortunately, this may be a laborious and demanding task. Many problems are not that well-defined. If it is a new type of plant, it might be difficult to know which are the decisive requirements and which that are easy to fulfil. A given constraint may be totally decisive for the outcome of the control design. A designer may be willing to adjust the requirements to achieve other benefits, but he is not willing to consider every case or combination. He wants to work in an iterative way and be able to eliminate bad approaches early. He also wants to know why a certain approach fails. He wants to get insight. For example, it may be easy and favourable to remove a constraint by redesigning the plant. It is in most cases favourable to take the interaction between process design and control design into account and consider them simultaneously.

A designer is happy when he has a profound understanding of the system dynamics. He then knows the possibilities and the limitations and can make the proper compromises during the design. He can justify why it is not possible to make a better design according to the circumstances. In many cases insight is the key to design. If you can pinpoint the critical parts and if you understand the difficulties, you can often solve or avoid the problems and make a good design.

In real life most plants have significant non-linear behavior, while most available software for analysis and synthesis assumes linear models. It is difficult to analyse non-linear systems. The simulation model could be used for empirical studies concurrently with a mathematical analysis. Possibilities to include and exclude different features in the model by changing the model for one part or by making parameter changes are useful when studying their importance. To have some success with the analysis we are more or less forced to work mainly with linear models and to estimate the effects of nonlinearities. Linearization is tedious to do with paper and pen. A good formula manipulation program which takes the nonlinear equations and outputs the linearized ones would be a real time-saver. If there also was a program that took the linear model and intervals for the parameters and made proper approximations, the analysis would be even simpler to carry out. A nice thing with linear models is that they can be transformed into the frequency domain where many dynamical properties are easier to understand. When analysing a system it is useful to have different viewpoints and possibilities to transform back and forth between different representations.

Support Users' Concepts

It is important that a user can describe his problems on a for him natural form. The user interface of a CACE system could be viewed as consisting of a language and environment. The language should be more than just a means for instructing the computer to perform tasks. It should also serve as a framework within which we organize our ideas. It should be a high level problem solving language.

It is important that the user can give the mathematical description of a submodel on a natural form. When deriving models from first principles the result is often a system of differential algebraic equations (DAE):

$$g(t, \dot{x}, x, v, p, c) = 0$$

where t is the time, x and v vectors of unknown variables, p a vector of known parameters and c a vector of known constants. It is natural to require that interactive software for model development and simulation supports DAE systems. The prototype simulator Hibliz ([Elmqvist and Mattsson, 1986]; [Mattsson, Elmqvist and Brück, 1986]) which was developed in another CACE project (STU project 84-5069) accepts mathematical descriptions given as DAE systems. However, most simulation packages of today do not allow models given as DAE systems, but require assignment statements for derivatives and algebraic variables. The user must solve for the derivatives and put the model on the form

$$\dot{x} = f(t, x, p, c)$$

He is often allowed to introduce sequences of auxiliary variables and to give the assignment statements in any order:

$$\dot{x} = f_1(t, \dot{x}, x, v, p, c)$$
$$v = f_2(t, \dot{x}, x, v, p, c)$$

as long as it possible to sort them so that all derivatives and auxiliary variables are calculated before use. This means that the user has to manipulate his model manually. This is a non-trivial task. Errors may be introduced.

When DAE systems are supported, the model becomes more readable since the user can recognize fundamental relations as mass and energy balances and other phenomenological equations. It is easier to check that the model is entered correctly and the risk of introducing errors during manual transformation is reduced.

Multi-Purpose and Reusable Descriptions

It is a laborious and time-consuming task to develop good models of plants and various phenomena. Consequently, it is important that the investments in model development can be reused. A model can be used for different purposes as simulation, analysis and design. The status of a variable may vary. Sometimes it is considered to be known, while in other situations we want to solve for it. For example, when solving for a stationary operating point the derivatives are set to zero and the states are to be solved for. When a numerical ODE solver is used, the states are considered to be known and we should solve for the derivatives. When designing the plant or the control system, some of the parameters are considered to be unknown by the designer.

Furthermore, as thoroughly motivated by [Elmqvist, 1978], the equation form is the only reasonable representation for model libraries. With models on assignment form, it must be decided for each submodel which of its variables that are inputs (in other words are known) and which of its variables that are outputs (defined by the model). As a simple example consider a resistor. Ohm's law states $V_1 - V_2 = RI$, where V_1 and V_2 are the voltages at the ends of the resistor, I is the current through the resistor and R is the resistance. The model has three variables V_1, V_2 and I. The resistance R is in this model a given parameter. If we should write the model on assignment form there are three possibilities

$$I := (V_1 - V_2)/R$$
$$V_1 := V_2 + RI$$
$$V_2 := V_1 - RI$$

The first variant assumes that V_1 and V_2 are inputs and defines I. This model is appropriate if for example one end of the resistor is connected to a voltage source and the other to ground. The second and third variants assume that the current and the voltage at one end is known. These models are appropriate if the resistor is connected to a current source and ground. Consequently, for models on assignment form we need several different models for a resistor, depending on how it is connected to the environment. This makes both use and maintenance of a model library messy. Furthermore, other environments may result in algebraic loops so that equation systems with equations from several submodels must be solved to transform the model into assignment form. Two resistors connected in series between a voltage source and ground is a simple example of this. Submodels cannot be transformed into assignment form individually, but the transformation is a global problem.

Improve the Numerical Properties

It is favourable if a CACE system accepts problems on forms preferred by users. By exploring symbolic manipulation the problems can in many cases be simplified and transformed to a form more suitable for numeric solution.

As an example consider the problem of finding the optimum of a function. The numerical solution procedure could be made faster and more robust if analytic procedures for calculating the gradient and the Hessian are given. However, in many cases it is laborious for the user to provide these procedures. It is much more convenient for him if they are generated automatically.

You may say that a problem is ill-conditioned if a small perturbation in the equations can lead to a large deviation in the solution. The main question is, however, what perturbations we have to consider in a particular case. If we have a fully parameterized model, where all explicit numbers are exact (structural ones, zeros etc), the perturbations of interest are those described by perturbations in the parameters. If we want to perform a numerical calculation and substitute the parameters with numbers, then it is of interest to consider unstructured and random perturbations to model for example quantization. Then a larger class of problems becomes ill-conditioned. For a fully parametrized problem the condition number is not a problem invariant, but it depends on the formulation and may be decreased by symbolic manipulation.

It is important to consider the structural properties of a problem when deciding

whether it is well-posed or not. For example the problem $\varepsilon\dot{y}+y=1$ where we know that the system is stable, is well-posed. The only perturbations that we have to consider are perturbations in ε which lead to an ε greater or equal zero. Even from a numerical view, it must be considered to be well-posed. It is a minimum demand that a non-negative number is represented by the computer as a non-negative number.

Possibilities to use symbolic manipulation to handle DAE systems are discussed in [Mattsson, 1986].

3. An Analysis Package in MACSYMA

The framework of polynomial matrices is useful for analysis of multivariable linear systems, see [Kailath, 1980]. However, polynomial matrices are not easily manipulated by hand. It is thus very important that good analysis tools for polynomial matrices are available. We have tried to fill the gap between theory and practice by implementing a package for analysis of multivariable linear systems in MACSYMA. The functions of this package is listed below. Then a MACSYMA demo with matrix fraction decompositions, co-prime factorizations, multivariable realizations, etc., will illustrate the beauty of symbolic manipulations. The examples are taken from [Kailath, 1980]. For further examples and details on the implementation including listings of the functions we refer to [Holmberg, 1986].

Available Functions

The following functions for analysis of multivariable linear systems have been implemented in MACSYMA.

Linearization

`LINEARIZE`	Linearizes the dynamical system $\dot{x} = f(x,u), \quad y = g(x,u)$

Stability analysis

`ROUTH`	Generates the stability conditions for a continuous time system
`JURY`	Gives the stability conditions of discrete time systems and the steady state output variance

Sampling

`SAMP`	Sampling from transfer function to pulse-transfer function
`SAMPSTATE`	Sampling from state space to state space

Geometry functions — state space

`HERMITE`	Gaussian elimination when applied to a constant matrix
`KER`	Computes the Kernel $\{X \mid AX = 0\}$
`INVERSE_IMAGE`	Calculates the inverse image $\{X \mid AX = B\}$ (A possibly singular)
`INTERSECTION`	Computes the intersection of two subspaces
`GRAM_SCHMIDT`	Calculates an orthogonal base for a subspace
`AINV`	Computes the maximal A-invariant subspace in a given subspace
`ABINV`	Computes the maximal (A,B)-invariant subspace in a given subspace

Factorization — Frequency domain

`SMITH`	Calculates the Smith form together with transformation matrices
`SMITH_McMILLAN`	Calculates the Smith-McMillan form with transformation
`HERMITE`	Calculates the Hermite form
`COLUMNREDUCE`	Makes a denominator polynomial matrix column reduced
`ROWREDUCE`	Makes a denominator polynomial matrix row reduced
`RMFD`	Right Matrix Fraction Decomposition (MFD) of a transfer matrix
`LMFD`	Left MFD of a transfer matrix
`RIGHTCOPRIME`	Gives a right coprime MFD from a noncoprime MFD
`LEFTCOPRIME`	Gives a left coprime MFD from a noncoprime MFD

SS2TF	State space to transfer function conversion
MAKESYS	Makes a list of the A, B, C, D matrices to represent a system

Multivariable Realizations

CONTROLLER	Calculates a controller form realization
OBSERVER	Calculates an observer form realization
CONTROLLABILITY	Calculates a controllability form realization
OBSERVABILITY	Calculates an observability form realization

Generation of the $\mathcal{S}(A, B, C, D)$-file

TOMIMO	Generates the file ABCD.MIM from A, B, C, and D.

Example—Polynomial Matrix Manipulations

The following example is a MACSYMA *Demo* that illustrate the use of polynomial matrices for analysis of multivariable linear systems. The cumbersome manipulations are done by the above functions. The Demo starts with a transfer function matrix, describing a multivariable linear system. The description is transfered into a matrix fraction decomposition, MFD, i.e. a polynomial matrix description. Extraction of different polynomial matrix factors of the MFD are made. Also, different multivariable realizations are presented. For terminology and a background the reader is refered to [Kailath], especially Chapter 6.

The file shown is a MACSYMA log file, output with the `typeset` switch `true`. The resulting Troff/EQN typesetting code has automatically translated to TEX by the program MacEQ2TEX (see [Mårtensson, 1986]).

```
(c1) load("login.mac")$
(c2) demo("realizations.dem");
/* This demo describes a couple of examples in Kailath, chapter 6.
Example 6.2-1. Alternative MFDs for a Transfer Function. p. 368-9.
Example 6.4-1. Controller-Form Realization of a Right MFD. p. 407-8.
Example 6.4-2. Observer-Form Realization of a Left MFD. p. 416
Example 6.4-6. Constructing Canonical Controllability Forms. p. 433-4. */
(c3) g:matrix([s/((s+1)*(s+2))^2,s/(s+2)^2],[-s/(s+2)^2,-s/(s+2)^2]);
```

$$\text{(d3)} \qquad \begin{bmatrix} \frac{s}{(s+1)^2(s+2)^2} & \frac{s}{(s+2)^2} \\ -\frac{s}{(s+2)^2} & -\frac{s}{(s+2)^2} \end{bmatrix}$$

```
/* Example 6.2-1. Alternative MFDs for a Transfer Function. p. 368-9. */
(c4) rmfd(g);
```

$$\text{(d4)} \qquad \left[dr = \begin{bmatrix} (s+1)^2(s+2)^2 & 0 \\ 0 & (s+2)^2 \end{bmatrix}, nr = \begin{bmatrix} s & s \\ -s(s+1)^2 & -s \end{bmatrix} \right]$$

```
(c5) ev(rightcoprime(dr,nr),%);
```

(d5)

$$\left[dr=\begin{bmatrix}(s+1)^2(s+2)^2 & -(s+1)^2(s+2)\\ 0 & s+2\end{bmatrix},nr=\begin{bmatrix}s & 0\\ -s(s+1)^2 & s^2\end{bmatrix},rr=\begin{bmatrix}1 & 1\\ 0 & s+2\end{bmatrix}\right]$$

```
(c6) ev(columnreduce(dr,nr),%);
```

(d6)

$$\left[dr=\begin{bmatrix}0 & -(s+1)^2(s+2)\\ (s+2)^2 & s+2\end{bmatrix},nr=\begin{bmatrix}s & 0\\ -s & s^2\end{bmatrix},u=\begin{bmatrix}1 & 0\\ s+2 & 1\end{bmatrix}\right]$$

```
/* Example 6.4-1. Controller-Form Realization of a Right MFD. p. 407-8. */
(c7) ev(real:controller(dr,nr),%);
```

(d7)

$$\left[a=\begin{bmatrix}-4 & -4 & 0 & -1 & -2\\ 1 & 0 & 0 & 0 & 0\\ 0 & 0 & -4 & -5 & -2\\ 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\end{bmatrix},b=\begin{bmatrix}0 & 1\\ 0 & 0\\ -1 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix},c=\begin{bmatrix}1 & 0 & 0 & 0 & 0\\ -1 & 0 & 1 & 0 & 0\end{bmatrix}\right]$$

```
/* Example 6.4-6. Constructing Canonical Controllability Forms. p. 433-4.
*/
/* Search by Crate 2 */
(c8) ev(controllability(a,b,c),%);
```

(d8)

$$\left[a=\begin{bmatrix}0 & 0 & -2 & 0 & 0\\ 1 & 0 & -5 & 0 & 0\\ 0 & 1 & -4 & 0 & 0\\ 0 & 0 & 2 & 0 & -4\\ 0 & 0 & 1 & 1 & -4\end{bmatrix},b=\begin{bmatrix}1 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 1\\ 0 & 0\end{bmatrix},c=\begin{bmatrix}0 & 0 & 1 & 1 & -4\\ -1 & 4 & -12 & -1 & 4\end{bmatrix}\right]$$

```
/* Search by Crate 1 */
(c9) crate_nr:1;
```

(d9) 1

```
(c10) ev(controllability(a,b,c),real);
```

(d10)

$$\left[a=\begin{bmatrix}0 & 0 & 0 & -4 & 2\\ 1 & 0 & 0 & -12 & 5\\ 0 & 1 & 0 & -13 & 4\\ 0 & 0 & 1 & -6 & 1\\ 0 & 0 & 0 & 0 & -2\end{bmatrix},b=\begin{bmatrix}1 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 1\end{bmatrix},c=\begin{bmatrix}0 & 0 & 1 & -6 & 1\\ -1 & 4 & -12 & 32 & -1\end{bmatrix}\right]$$

```
/* Example 6.4-2. Observer-Form Realization of a Left MFD. p. 416 */
(c11) lmfd(g);
```

$$(d11)\qquad \left[dl=\begin{bmatrix}(s+1)^2(s+2)^2 & 0\\ 0 & (s+2)^2\end{bmatrix},nl=\begin{bmatrix}s & s(s+1)^2\\ -s & -s\end{bmatrix}\right]$$

```
(c12) ev(leftcoprime(dl,nl),%);
```

$$(d12)\qquad \left[dl=\begin{bmatrix}(s+1)^2(s+2)^2 & 0\\ (s+1)^2(s+2) & s+2\end{bmatrix},nl=\begin{bmatrix}s & s(s+1)^2\\ 0 & s^2\end{bmatrix},rl=\begin{bmatrix}1 & 0\\ -1 & s+2\end{bmatrix}\right]$$

```
(c13) ev(rowreduce(dl,nl),%);
```

$$(d13)\qquad \left[dl=\begin{bmatrix}0 & -(s+2)^2\\ (s+1)^2(s+2) & s+2\end{bmatrix},nl=\begin{bmatrix}s & s\\ 0 & s^2\end{bmatrix},u=\begin{bmatrix}1 & -(s+2)\\ 0 & 1\end{bmatrix}\right]$$

```
(c14) ev(observer(dl,nl),%);
```

$$(d14)\qquad \left[a=\begin{bmatrix}-4 & 1 & 0 & 0 & 0\\ -4 & 0 & 0 & 0 & 0\\ 0 & 0 & -4 & 1 & 0\\ 1 & 0 & -5 & 0 & 1\\ 2 & 0 & -2 & 0 & 0\end{bmatrix},b=\begin{bmatrix}1 & 1\\ 0 & 0\\ 0 & 1\\ 0 & 0\\ 0 & 0\end{bmatrix},c=\begin{bmatrix}0 & 0 & 1 & 0 & 0\\ -1 & 0 & 0 & 0 & 0\end{bmatrix}\right]$$

4. Generation of a MIMO System Text File

It will now be demonstrated how a MIMO system in MACSYMA can be transfered into a text file of a special form. The special form of the text file is chosen to be the same as the print format from CTRL-C. This makes it possible to load results form MACSYMA into CTRL-C. It should also be mentioned that there is a Pascal program written by [Mårtensson, 1986] that generates Simnon code from this text file representation. The generation of the text file from MACSYMA is made by the function TOMIMO. This is a LISP program and consequently we have to enter the LISP mode before we apply it to our MIMO system.

```
(c1) load("login.mac")$
(c2) a:matrix([1,2],[3,4])$
(c3) b:matrix([5,6],[7,8])$
(c4) c:matrix([9,10],[11,12])$
(c5) d:matrix([13,14],[15,16])$
(c6) sys:makesys(a,b,c,d);
```

$$\text{(d6)} \qquad \left[a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, b = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}, c = \begin{bmatrix} 9 & 10 \\ 11 & 12 \end{bmatrix}, d = \begin{bmatrix} 13 & 14 \\ 15 & 16 \end{bmatrix}\right]$$

```
(c7)
Break Entering Lisp:
<1>: (load "tomimo.l")
t
<1>: (tomimo $sys 'abcd.mim)
t
```

The MIMO system has now been written in the file `ABCD.mim`. This file looks as follows:

```
nmp =

2 2 2

a =

1 2
3 4

b =

5 6
7 8
```

```
c =

 9 10
11 12

d =

13 14
15 16
```

---

5. Root Locus

In this section we will demonstrate a method for computing root loci by combining symbolic and numerical computations. Only a very brief description will be given. A fuller description is given in [Holmberg,Lilja,Mårtensson, 1986].

The naive way of plotting root loci of the type

$$A(s) + kB(s) = 0$$

where A and B are polynomials and k real, is to solve the equation for a number of equi-distant k-values and then mark the roots by an '$\times$' for each k. This method has severe disadvantages: Firstly it is rather time consuming and secondly it gives a very bad resolution near multiple roots. To be presentable, the plots also need heavy manual paste-up. In the following subsection, a method based on the implicit function theorem is suggested. A non-linear differential equation, that describes the root locus locally, is obtained by some manipulations of the transfer function (done in MACSYMA for example) and a package for solving the differential equation (e.g. Simnon) can then be utilized to compute and plot the root locus. This method is both faster and gives a better performance near multiple roots than the method mentioned above.

A general problem

This subsection proposes a method for plotting the locus of points s in the complex plane satisfying the equation

$$f(s,k) = 0, \qquad s,k \in \mathbb{C} \tag{♭}$$

where f is analytic in s and k and where k is restricted to the real axis. Several common control theory problems are covered in this formulation: Ordinary root loci, LQG root loci (k = control weighting), zeros of sampled systems (k = the sampling interval), etc.

The method is based on the implicit function theorem applied to (♭). The idea is the following: The problem is to compute $\{s|f(s,k) = 0, \quad k \in [a,b] \subset \mathbb{R}\}$. For this, compute the k's such that (♭) has multiple roots in s. Away from these, the branches $s_i(k)$ satisfies

$$\frac{d}{dk}s_i(k) = -\frac{\dfrac{\partial f}{\partial k}}{\dfrac{\partial f}{\partial s}} \tag{♯}$$

Implementation

The transfer function $G(s)$ is specified in MACSYMA and the closed loop characteristic polynomial $p(s,k)$ is calculated. The real and imaginary part of the right hand side of (♯) are then computed and written to a file using the function `print_ode`. To avoid divide overflow in Simnon when a multiple root is encountered one has to stop the integration before the multiple root. For each multiple root, one therefore has to find the local behavior of s with respect to k. A graphical method to do this is to plot a

Newton diagram. This is equivalent to making the substitution $s := bk^a$ in the characteristic polynomial and then finding pairs of dominating terms. The function `newton` implements this and returns two lists. The first list contains the possible k^a-alternatives and the second list gives the corresponding coefficients (expressed as a polynomial in b). The function `near_multiple_roots` uses `newton` for calculating the values of k for which $|\frac{d}{dk}s_i(k)| = d_{max}$. The function `print_kxy` uses `near_multiple_roots` to print out these k values and the corresponding solutions in s.

The differential equation for the real and imaginary parts of the root locus is written into the file `ode.rl`. The k-values specifying the intervals for which the root locus is to be plotted for are written (together with the corresponding initial values for the branches of the root locus) into the file `rootloc.rl`. These two files are then processed by a procedure written in the "editor language" TPU (Text Processing Utility) in VMS generating one Simnon system description file `ode.t` (the "dynamics" file) and one file `rootloc.t` containing the commands for setting initial values and integrating.

An Example

The following MACSYMA dialogue shows an example where the functions `print_ode` and `print_kxy` are used. In the example the interval for the gain k is chosen to $-2 \leq k \leq 2$ and the maximum derivative to $d_{max} = 100$.

```
(c1) load("rootloc.mac")$
(c2) g:matrix([1/s^2,1/s],[-1/s^2,0]);
```

(d2)
$$\begin{bmatrix} \frac{1}{s^2} & \frac{1}{s} \\ -\frac{1}{s^2} & 0 \end{bmatrix}$$

```
(c3) print_ode(g);
```

(d3) *ode.rl*

```
(c4) print_kxy(g,-2,2,100);
```

(d4) *rootloc.rl*

The resulting files `ode.rl` and `rootloc.rl` are then processed by the TPU file `rootloc.tpu` to get the Simnon system description file `ode.t` and the Simnon command file ("macro" file) `rootloc.t`. The Simnon commands required to plot the root locus are:

```
> syst ode
> axes h -2 2 v -2 2
> rootloc
```

The result is shown in Figure.

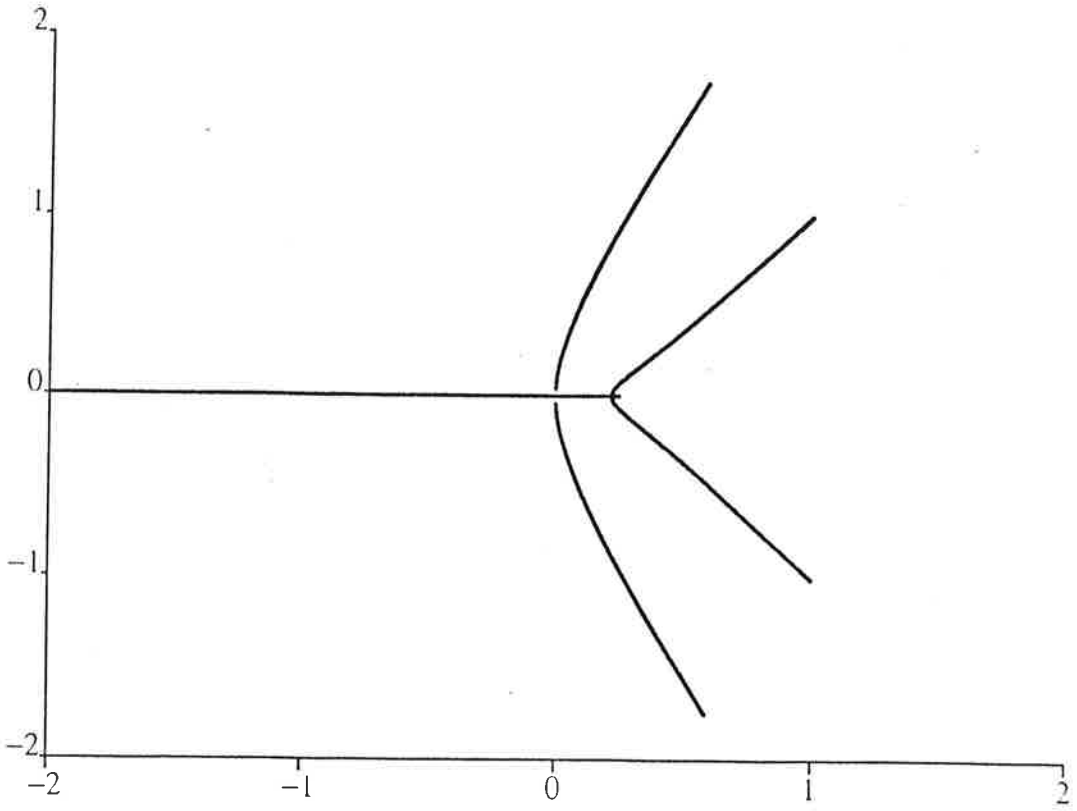


Figure The Root Locus Plot in the Text.

6. Conclusions

As motivated and illustrated symbolic manipulation could be very useful. The user interface could be improved by allowing the user to work on a higher level. He can present his problem on a for him suitable form. Results on analytic or symbolic form could give a better insight into structural properties than numerical tables. Even when it is not possible to carry the symbolic calculations all the way through, symbolic manipulation could be useful. Symbolic manipulation could simplify the problem and transform it to a form better suited for numerical solution. Symbolic manipulation could also be used for automatic generation of procedures for calculating gradients, Jacobians, Hessians etc. thereby relieving the user's burdon and hopefully decreasing the possibilities of introducing errors.

Our experiences of MACSYMA are that it is a powerful tool and can do a lot with proper guidance from the user. One advantage with with MACSYMA is that it is written in Lisp. This makes it possible to extend the program with Lisp functions. As you remember from Chapter 4, this enabled us to establish an interaction between MACSYMA and other programs, like Simnon and CTRL-C. The drawbacks are that it is a large program and that it consumes a lot of computer power. Unfortunately, MACSYMA is not modularized. For use in CACE systems it should be desirable to have modularized tools for symbolic calculation so that a user could select for him a proper set. We are eagerly searching for such a toolkit.

It is important to consider that there is a user in the loop. He can in many cases

improve the manipulation by proposing substitutions and by informing the system on what kind of forms he want the answer. In many cases an equation system can be simplified considerably if it can be assumed that a parameter or a certain expression is zero. It is difficult for the user to anticipate all such cases in advance, but he may well be able to answer those questions interactively. Also, if the model is modified there should be facilities to take care of assumptions made before so he doesn't need to consider them once more when the manipulations are redone. To speed up the manipulation it is advisable to store the successful path and try it when the user has modified his model. The logging facility is also necessary for the explanation facility. If the numerical solution procedure fails, the error message should relate to the user's original formulation and not to the manipulated expressions.

Acknowledgements

The work has been supported by the Swedish Board of Technical Development in the project "Combination of formula manipulation and numerics" (STU–85–4809).

References

ELMQVIST, H. (1978): "A Structured Model Language for Large Continuous Systems," Ph.D-thesis CODEN: LUTFD2/TFRT-1015, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. and S.E. MATTSSON (1986): "A Simulator for Dynamical Systems Using Graphics and Equations for Modelling," *Proceedings of the IEEE Control Systems Society Third Symposium on Computer-Aided Control Systems Design (CACSD)*, Arlington, Virginia, September 24–26, 1986.

ELMQVIST, H., K. J. ÅSTRÖMand T. SCHÖNTHAL (1986): *Simnon—User's Guide for MS–DOS Computers*, Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

HOLMBERG U. (1986): "Some MACSYMA Functions for the Analysis of Multivariable Linear Systems," Report CODEN: LUTFD2/(TFRT-7333)/1–40/(1986), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

HOLMBERG, U., LILJA, M., MÅRTENSSON, B. (1986): "Integrating Different Symbolic and Numeric Tools for Linear Algebra and Linear System Analysis," Report CODEN: LUTFD2/(TFRT-7338)/ 1–17/(1986), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

KAILATH, T. (1980): *Linear Systems*, Prentice-Hall Information and System Sciences Series, Englewood Cliffs, New Jersey.

LILJA, M. (1986): "Symbolic and Numerical Computation of Implicit Functions, Especially Root Loci," Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden, to appear.

MÅRTENSSON, B. (1986): "MacEQ2TEX and *S*2TEX—Automatic TEX-code generation from MACSYMA and CTRL-C," Report CODEN: LUTFD2/(TFRT-7334)/1–11/(1986), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

THE MATHLAB GROUP LABORATORY FOR COMPUTER SCIENCE (1983): *MACSYMA Reference Manual*, M.I.T., Cambridge, MA.

MATTSSON, S.E. (1986): "On Differential/Algebraic Systems," Report CODEN: LUTFD2/TFRT-7327, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E., H. ELMQVIST and D.M. BRÜCK (1986): "New Forms of Man-Machine Interaction," Report CODEN: LUTFD2/TFRT-3181, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

PAVELLE, R., M. ROTHSTEIN and J. FITCH (1981): "Computer Algebra," *Scientific American* **245**, 6, 136–152.

SYSTEMS CONTROL TECHNOLOGY, INC., *CTRL-C, User's Guide*, 1801 Page Mill Road, Palo Alto, CA 94304.

KNOWLEDGE-BASED CONTROLLERS

Karl-Erik Årzén
Department of Automatic Control,
Lund Institute of Technology,
Lund, Sweden

- EXPERT CONTROL
- KNOWLEDGE-BASED CONTROLLERS
- A FRAMEWORK
- EXPERIMENTS
- CONCLUSIONS

EXPERT CONTROL

Potential expert system applications in process control

- *Process design*
- *Control design*
- *Process monitoring and diagnosis*
- *Alarm analysis*
- *Process scheduling*
- *Knowledge-based controllers*

Applications can usually be categorized according to the following criteria.

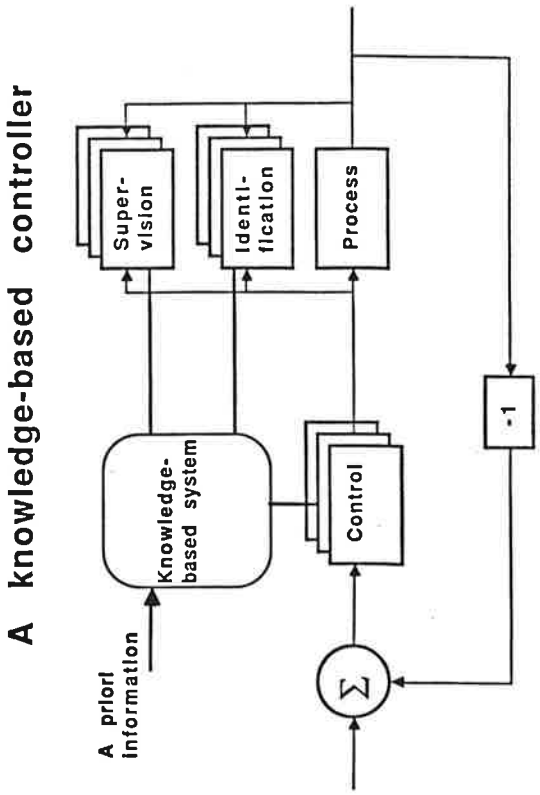
Plant-wide — single loop

Off-line operation — on-line operation

Operator assistance — closed loop operation

KNOWLEDGE-BASED CONTROLLERS

- The combination of a knowledge-based system with a set of numerical algorithms for control, identification and monitoring.
- On-line, closed loop application on the single control loop level.
- Applies control knowledge to build up process knowledge.
- Acquires process knowledge through on-line tuning experiments and from process operators.
- Orchestrates numerical algorithms.



THE ROLE OF PROCESS KNOWLEDGE

Good control always requires process knowledge.

Expressed as mathematical models with a certain structure and parameter values.

Control algorithm = control structure + coefficient values.

Process knowledge can be obtained in different ways:

- *Laws of nature*
- *Off-line identification*
- *Experience from similar loops, rules-of-thumb*
- *Auto-tuners*
- *Adaptive controllers*
- *Process operators*

THE HEURISTIC ASPECT

Heuristics play an important role.

Shows up as selectors and logical branching statements

Controller = control algorithm + logical safety net.

Extremely important, complex, difficult to develop.

The size increases when a controller is formed as a combination of different algorithms.

THE KNOWLEDGE-BASED APPROACH

FUNCTIONALITY:

Builds up process knowledge through tuning experiments.

Makes use of a priori information.

A combination of dedicated algorithms.

Belongs to the auto-tuning, adaptive class of controllers.

More flexible than existing auto-tuners.

IMPLEMENTATION:

Implemented with expert system techniques.

Manage heuristics in a structured way.

Explicit knowledge representation.

Modularity

EXPERT SYSTEM ASPECTS

Contains several different problem types; classification, monitoring, design, planning ...

Real-time application

Available knowledge?

OTHER ASPECTS

USAGES:

Industrial controller

Testbench for development of new controllers

USER INTERFACE:

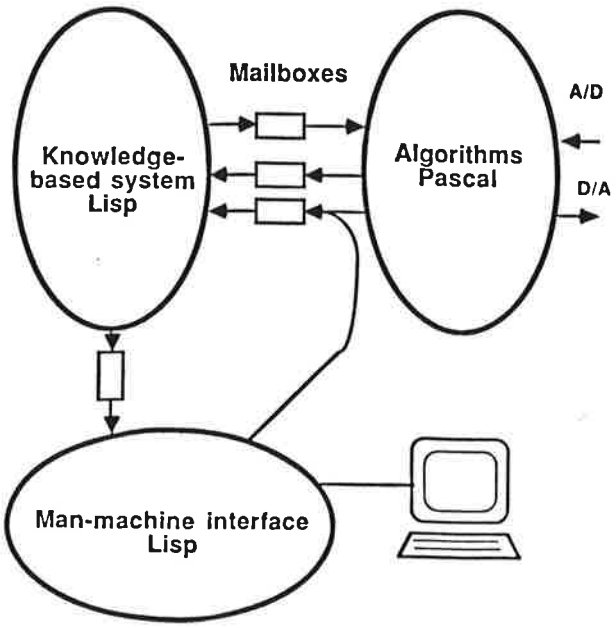
Knowledge extraction phase – tutoring aspect

Qualitative performance related knobs

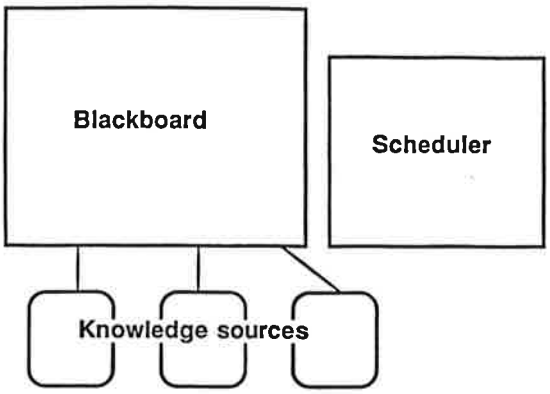
Extensive query facilities

IMPLEMENTATION

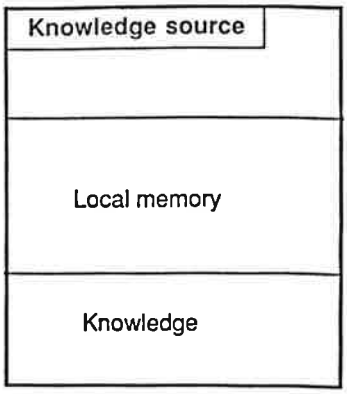
Vax 11/780 VMS & Eunice



KNOWLEDGE-BASED SYSTEM STRUCTURE



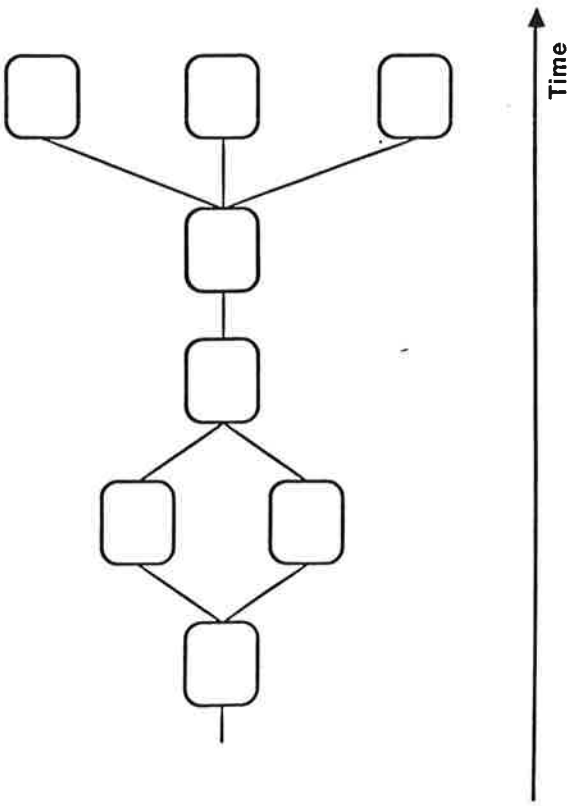
KNOWLEDGE SOURCES



Contains the domain knowledge for a certain task.

Different knowledge representation techniques.

Often associated with an algorithm.



OPERATING SYSTEM ANALOGY

Knowledge source — concurrent process

Scheduler — real-time operating system scheduler

Real-time primitives

BUILDING BLOCK: FLAVORS

Object-oriented programming package.

Supports hierarchies of classes (flavors) and class instances.

Methods and message passing implements generic interfaces.

Multiple inheritance of methods and instance variables.

Source code implementation from University of Maryland.

BUILDING BLOCK: YAPS

OPS4 type forward chaining production system from University of Maryland.

Source-code available.

Key features:

The conditions parts of the rules may contain arbitrary Lisp predicates.

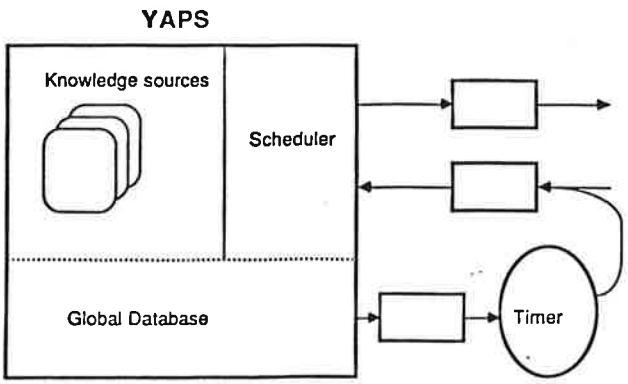
Implemented in Flavor and can have Flavor instances in the database.

Extensions:

HOW explanation facility.

Allows OPS5 type object structures in the database

KNOWLEDGE-BASED SYSTEM IMPLEMENTATION



Knowledge sources:

Forward chaining (YAPS)

Backward chaining (Emycin)

Procedural (Lisp subset)

SCHEDULER

```
(defobject knowledge-source
  (name
   type
   (status :default inactive)
   (state :default inactive)
   description
   flavor-instance)
  ())
```

A small message interface makes it easy to add new types of knowledge sources.

SCHEDULER cont.

```
(p schedule1

 "If a knowledge source is active
  and no other knowledge source is
  currently running then run this
  knowledge source"

 (object knowledge-source
   status active
   state ready
   flavor-instance -x)
(~ (object knowledge-source
     state running))
-->
  (modify 1 state running)
  (send -x 'run))
```

Knowledge sources may not be interrupted.

Waits for incoming messages when no knowledge source is active.

FORWARD CHAINING KS

Written in YAPS

Primitives for:

- Creating and modifying database elements both locally and globally
- Waiting a certain time
- Waiting for a specified database element

Periodic rules and database elements with duration times

BACKWARD CHAINING KS

Implemented with Flavors

EMYCIN inspired

Can ask questions to the user

Possible extension: Questions to the controlled process i.e. to other knowledge sources.

PROCEDURAL KS

Interpretor for a subset of Lisp

Real-time primitives available

KNOWLEDGE SOURCE COMBINATION

Sequences of knowledge sources, i.e. plans, can be formed in different ways.

- Knowledge sources suspend and resume each other.
- Pre-stored plans, e.g. as procedural knowledge sources.
- Dynamic plan generation. Pre- and postconditions are associated with each knowledge source. Planning problem with parallel activities.

CURRENT WORK

Knowledge-based PID controllers

Tuning methods: Relay, pulse and step response methods

Approximated process models: First order, first order with time delay, second order.

Different control design methods involved.

CONCLUSIONS

- Knowledge base of practical control issues
- Theoretical development
- Algorithm development
- Real-time expert system framework with wider applicability

Knowledge Based Controllers

Karl-Erik Årzén
Department of Automatic Control
Lund, Sweden

Expert systems, or knowledge-based systems, have many potential applications in process control. A few examples are process design, control design, process monitoring, alarm analysis, process scheduling, closed loop control etc. Many of these applications are commonly referred to as Expert Control. The nature of the applications are very diverse. They can, however, usually be categorized according to three different criteria. The first criterion states the size of the application. The extreme cases here are the global, plant-wide level and the single loop level. The second criterion states if the application is used off-line e.g. in an analysis or design phase, or if it is used for on-line operation. The last criterion tells whether the expert system's main task is to present information to the operator or if it is used in closed loop, i.e. can directly affect the process. This criterion is only applicable for the on-line case.

A knowledge-based controller is the combination of a knowledge-based system and a set of numerical algorithms for control, identification and monitoring. It is an on-line, closed loop application on the single loop level. The controller should build up enough process knowledge to choose and tune an appropriate controller. The knowledge is built up through different dynamic tuning experiment. The controller should also be able to exploit available a priori process knowledge. This knowledge could e.g. contain the type of control problem i.e. if it is a temperature loop or a flow loop etc. It could also contain reasonable time scales and qualitative statements of e.g. open loop stability. The main task of the knowledge-based system is to orchestrate the different algorithm e.g. to set parameters in the algorithms and to interpret results from algorithms. From a functional point of view, the knowledge-based controller could be compared with auto-tuners and adaptive controllers. It should, however, be more flexible than existing auto-tuners. As an example, it should be able to approximate the process dynamics with models of different complexity. Depending on the model structure, the model parameters and the specifications, the controller should have the possibility for different control designs. Existing adaptive controllers need much a priori information in order to be safely initialized. The knowledge-based controller could e.g. be used to built up this knowledge.

Fuzzy control or rule-based control is a related area. The difference is that in knowledge-based control conventional control algorithms are used, whereas in fuzzy control the control algorithm is replaced with a set of heuristic rules that determine how the control signal should be chosen in different situations.

An important part of any industrial controller is the safety network that surrounds the numerical algorithm. This network implements different heuristic aspects of the controller. The network shows up as logic branching statements. Experience has shown that the safety net part often dominates the code. They are difficult to design and test. The size of the network increases considerably when the controller is formed as a combination of several algorithms.

A controller with the described functionality can, as well as any expert system, without doubt be implemented with conventional programming techniques. The use of expert system techniques is an attempt to manage the heuristic elements of the controller in a structured way. The controller is separated in two parts, the knowledge-based system which implements the logic safety net, and the numerical algorithms. The most important expert system characteristics are the explicit,

declarative knowledge representation and the modularity. The explicit knowledge representation also simplifies the implementation of extensive query facilities. Possible questions are e.g. "What process model has you come up with?", "Which control design method was used and why?" etc.

The knowledge-based controller has two potential usages. As an actual industrial controller or as a testbench for experiments with new controller structures.

The majority of the existing expert system frameworks are not intended for real-time operation. From a logical standpoint, serious and unsolved theoretical problems arise, e.g. the non-monotonic reasoning problem. Our approach is to view expert systems as a high-level programming technique. Conventional expert system frameworks also lack real-time primitives.

A knowledge-based control framework has been developed. It is implemented on a VAX 11/780. It consists of three parallel subprocesses, the knowledge-based system, the numerical algorithms and the man-machine interface. The expert system and the man-machine interface are written in Lisp and the algorithm process is written in Pascal. The subprocesses communicate through message passing. The messages from the knowledge-based system typically contain orders to stop, start or change parameters in an algorithm. The messages in the opposite direction contain identification results or detected events. The algorithms can be viewed as filters that extract symbolic information from the numerical signal flow.

The knowledge-based system is built around the blackboard concept. A frame structured, global database is accessable from different knowledge sources. The knowledge sources contain the domain knowledge for a certain subtask of the problem. Different knowledge representation techniques are allowed in the knowledge sources. Examples are forward chaining, backward chaining and procedural knowledge sources. A knowledge source typically contains the logic surrounding one or a couple of algorithms. The tuning phase consists in a sequential and parallel combination of different knowledge sources in time. A separate scheduling module schedules the selection of knowledge sources at two different levels. The first level involves the scheduling between different knowledge sources that are active simultaneously. This resembles the operation of an ordinary real-time operating system. The knowledge sources correspond to concurrent processes. The second level involves the sequential scheduling among the knowledge sources. This can be formulated as a planning problem.

Several different real-time primitives have been implemented. A knowledge source can suspend the rule execution for a certain time or until a certain database element is entered in the global database. These primitives can be used to implement periodic rules, i.e. rules that are tested at a certain time interval, and to associate duration times with database elements.

The building blocks for the system are the YAPS framework and object-oriented programming as in Flavors.

The system is currently used for experiments with knowledge-based PID autotuners. The tuning methods which are used in the system are the relay method and the step response method. From these methods, PID parameters can be decided in several ways.

Exempel på STU - program m. väsentliga kopplingar till CACE

Projektstöd

Mät- o. reglertekn. forskn.

Industriell mät- o. reglertekn.

Forsknings - ramprogram

Datavetenskap

Signalbehandling

Insatsområden

Verkstadsföretagets informationssystem

Drifutvecklingssystem för processindustrin (DUP)

DUP
Drift- Utvecklingssystem för Processindustrin

BEHOVSBAKGRUND

Konkurrens med ökad kvalité och produktjämnhet

Arbetsinnehållet för operatörerna behöver ökas

ANVÄNDNINGSMÅL

Förbättrad kvalitet och produktjämnhet i processindustrin

Kvalificerat arbetsverktyg för driftsledning och operatör

Tillvarata och utveckla operatörernas yrkesskicklighet

Verktyg för val av reglerstrategier och dimensionering av system

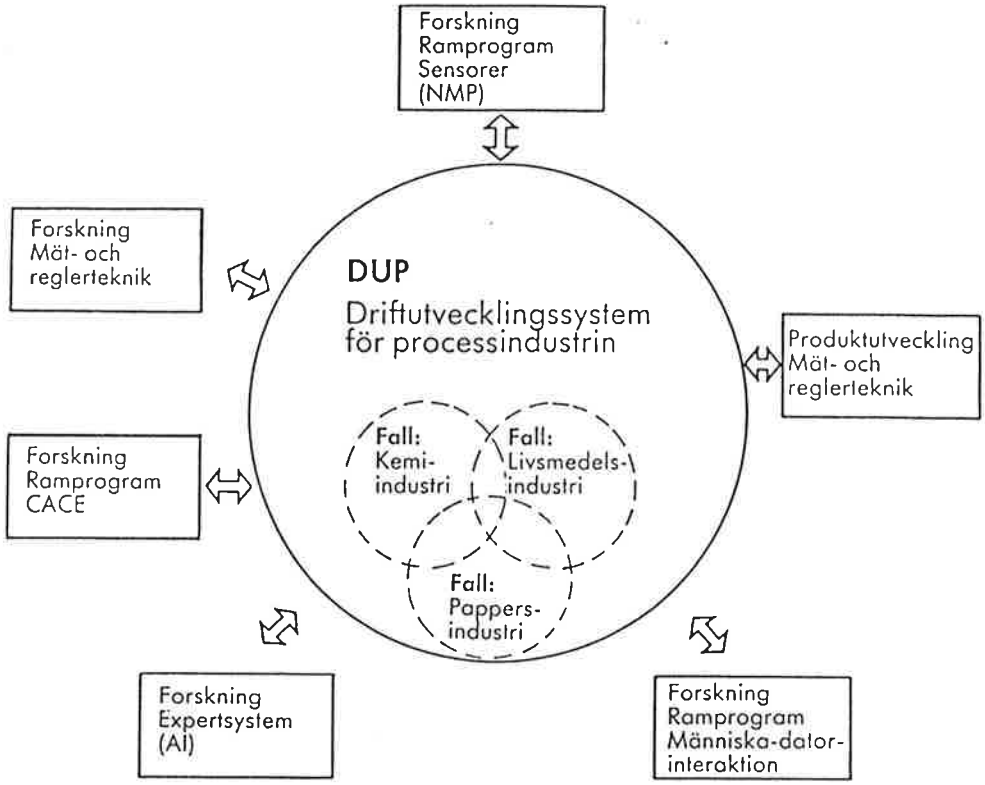
PROGRAMMÅL

Kunskapsspridning från teori till tillämpning

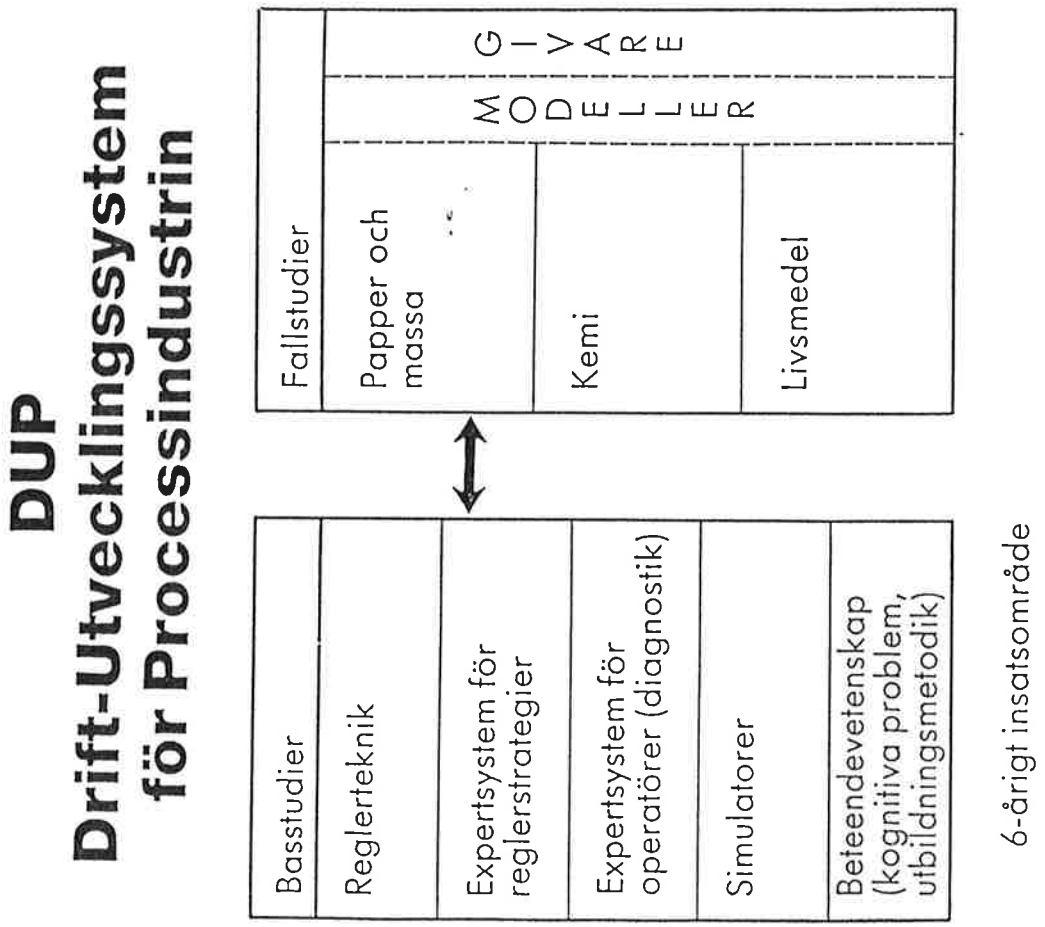
Kunskapsspridning inom och mellan branscher

Tvärteknisk och tvärvetenskaplig samverkan

Samverkan mellan DUP och andra IT-program

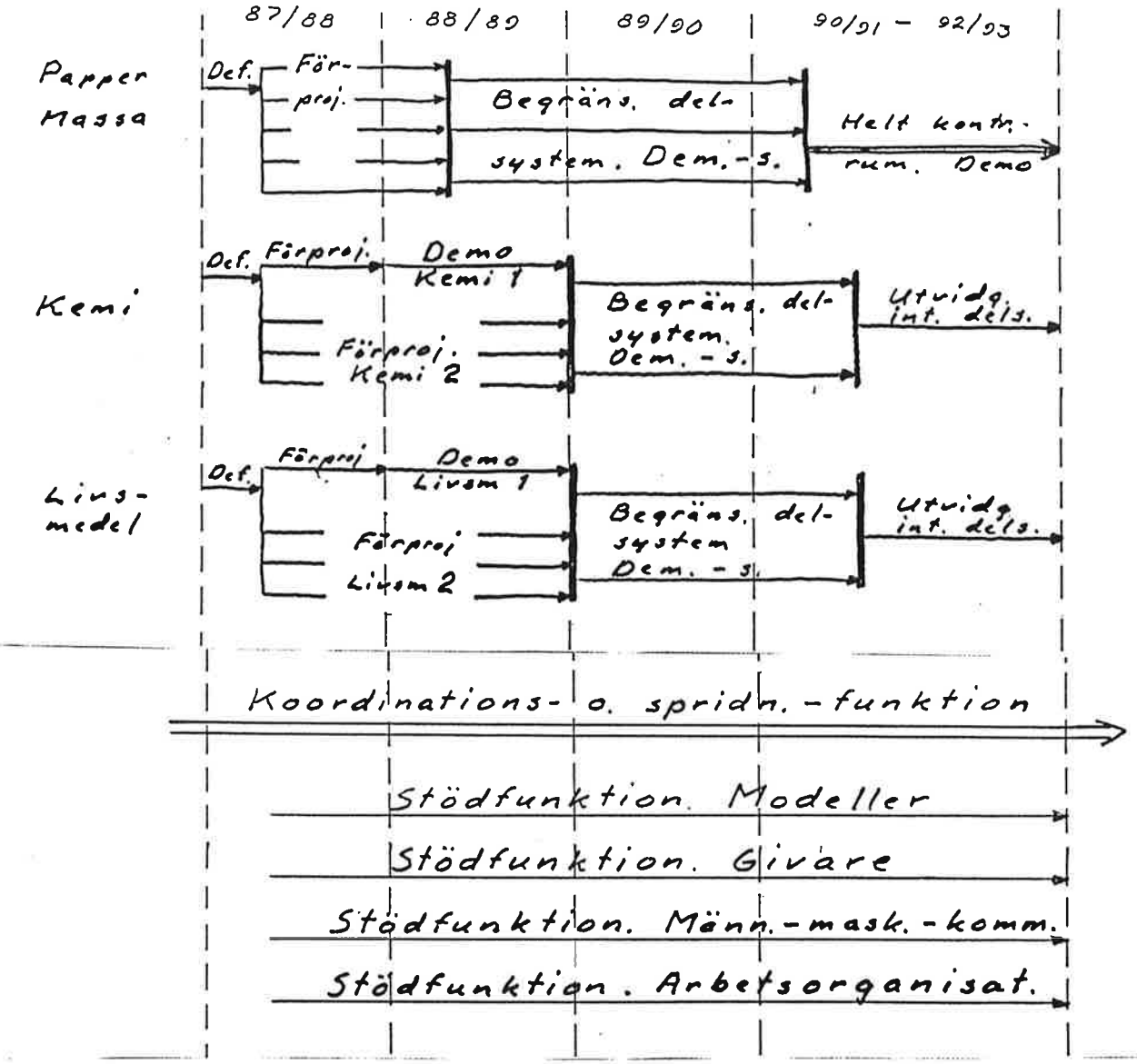


872 16455-17



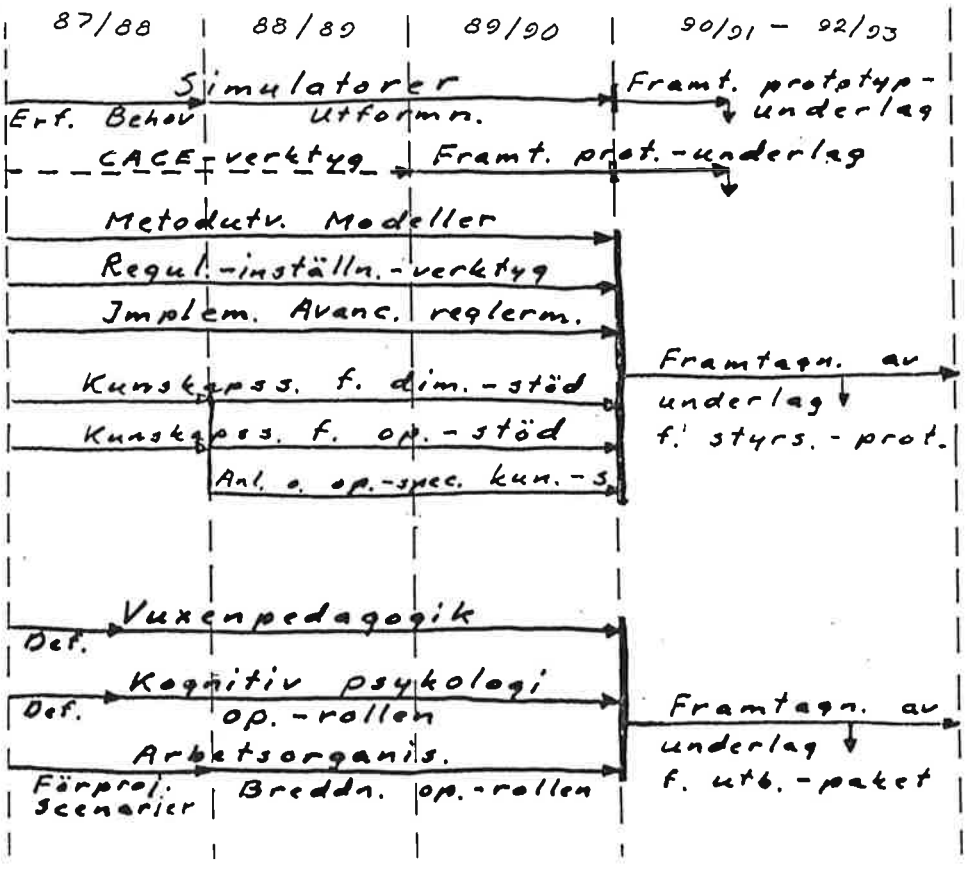
Skiss
A0/870211

Strukturplan DUP



Skiss
A0/870216

Strukturplan DUP



SLUTSATSER

- Grafikbaserade användarsnitt
- Expertsystemsnitt
- Symbolisk formelbehandling och numerik
- Systemrepresentation
- Implementeringsspråk
- Expertreglering

HUR SKALL VI AGERA?

Både bredd och djup:

- Störst insats på spjutspetsprojekt
- Bredd genom internationella kontakter
  Samla in idéer, erfarenheter och programvara
  Våra resultat är bra bytesvara.

SPJUTSPETSPROJEKT

Utnyttja erfarenheterna från pilotprojekten.

Bygga en prototyp:

- illustrera idéer och principer för
  användare
  tillverkare av CACE-programvara
- kan få användarnas synpunkter och idéer
- ger återkoppling på vad man kan göra
- bytesvara

Verktygslåda för simulering:

- viktig komponent i CACE-system
- fokuserar på representativa och relevanta problem
  representation av system
  grafik för definition av system struktur
  grafik för att visa beteende hos ett system
  symbolisk manipulering
- passar bra till DUP
- passar bra in i det internationella samarbetet
- ett område där vi är starka och har tradition

VERKTYGSLÅDA FÖR SIMULERING

Struktureringsmekanismer:

- modelldekomposition – blockdiagram
- multipla representationer
- bibliotek

Beskrivning av beteende på symbolisk form
från vilken CACE-systemet kan generera

- effektiv simuleringskod
- kod för att beräkna stationära punkter
- linjäriserade modeller etc.
- beskrivningar som accepteras av andra paket
- reglerkod

Grafisk presentation av resultat från analyser och simuleringar.

Integrera tredjeparts programvara.

Tillämpningsstudie.

BREDDEN

- Kunskapsbaser för identifiering och syntes av regulatorer
- Metoder och kod för syntes
- SERC
- CEGB
- INRIA
- Maryland
- Berkeley
- Lawrence Livermore National Laboratory

NYTTIGGÖRANDE AV RESULTATEN

Vetenskapliga resultat:

- doktorsavhandlingar
- licentiatavhandlingar
- examensarbeten
- artiklar i internationella tidskrifter
- internationella konferensbidrag
- interna forskningsrapporter

Informationsöverföring till CACE-tillverkare.

Informationsöverföring till industrin:

- utbildade forskare och ingenjörer
- forskarutbildningskurser
- kurser för ingenjörer verksamma i industrin
- studiebesök och samarbetsprojekt
- specialiserade verktyg
  simulatorer och "virtuella processer"

Överföring av prototyper och verktyg till DUP.