MODULAR, DIGITAL SIMULATION OF ELECTRO/HYDRAULIC DRIVES USING CSMP

François E. Cellier, Research Collaborator, Institute for Automatic Control,
Federal Polytechnical Institute (ETH), Gloriastr. 35, CH-8006 Zurich/ZH, Switzerland

Dr. Bernardo A. Ferroni, Dep. of Automatic Control, AGIE Ltd.,
CH-6616 Losone/TI, Switzerland

I. INTRODUCTION

The advantages and disadvantages of different program languages for the digital simulation of continuous dynamic systems have often been discussed during the last years. In particular the question was asked whether simulation problems should be solved by using a user-oriented program language as FORTRAN, ALGOL, PL/I etc. or by use of a problem-oriented program language as CSMP, MIMIC, SLANG etc.. In this discussion there is still one point of great importance which - as for our opinion - was not yet enough taken into consideration -- the modularity of subprograms. It is to prove that the CSMP-language provides the user with a very powerful instrument to obtain complete modularity of subprograms. In the following we will always quote the electro/hydraulic drives as an illustration for the modular, digital simulation using CSMP.

II. FORMULATION OF THE PROBLEM

Let us consider a system consisting of a motor and a gear (fig.1).

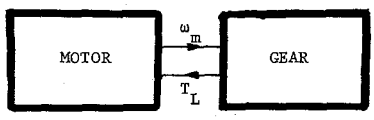

Fig.1 System consisting of a motor and a gear.

To calculate the motor speed (ω_m) knowledge of the load torque (T_L) is certainly necessary in advance. On the other hand this torque itself may depend on the motor speed due to speed-dependent friction. In this case there exists an algebraic loop between the two modules as there are motor and gear. In the same way algebraic loops are often involved between different modules and even in a single module itself. It is possible to break most of such loops by introducing memory functions. Memory functions are functions whose output-vector depends only on past values of the input-vector and of the output-vector but not on the input-vector at the actual time. In such cases the loop can be solved by simply arranging the statements in an accurate manner. Normally the memory function is introduced by use of an explicit integration routine - as done also in CSMP. In the example which was given above the load torque directly influences the acceleration of the motor which is separated from the motor speed by integration. The motor speed depends - due to explicit integration - only on the acceleration at past values of time and can therefore be calculated without primary knowledge of the torque load at the actual time.

A "real" algebraic loop is called a loop of the form

$$\left|\begin{array}{l} y = f(x) \\ x = g(y) \end{array}\right| \qquad (1)$$

which is not separated by an integration or another memory function. Such loops can be solved by different methods:

analytical method: It is often possible to reorganize the equations in a way so that the algebraic loop does not appear any more. The disadvantage of this method is that one loses the physical variable form of the system description which is desirable in simulation programs.

numerical iteration method: In CSMP there exists the functional block IMPL. By use of this feature in the form

```
|Y = IMPL (IC, P, FOFY)|
|X = G(Y)              |
|FOFY = F(X)           |
```
(2)

one full iteration will be carried out for each simulation step. This method is therefore very expensive and should be used only in case of absolut need.

analytical buffer method: The typical solution of the analytical mathematician will be:

```
|XX = INTGRL (0., X)  |
|XXX = DERIV (0., XX)|
|Y = F(XXX)          |
|X = G(Y)            |
```
(3)

This method is not very cheap either. It seems to be very elegant indeed. Numerically the user introduces nevertheless a phase error because by calculation of XX he loses some information concerning the phase (due to explicit integration routine used in CSMP) which he cannot recover through his DERIV-block. If this phase error is acceptable there exists a cheaper method:

numerical buffer method: The set of equations (1) can be written in the form

```
|XX = DELAY (3, 3.*DELT, X)|
|Y = F(XX)                  |
|X = G(Y)                   |
```
(4a)

or

```
|Y = F(X)                   |
|YY = DELAY (3, 3.*DELT, Y)|
|X = G(YY)                  |
```
(4b)

The DELAY-block is of course also a memory function. If both descriptions (4a,4b) are used one after the other and if the results of these simulation runs do not differ to much from each other, this does not prove in the mathematical sense of the word but at least indicates the reliability of the results, because the phase errors will occur contrarily.

Such a "real" algebraic loop will occur e.g. by simulating a lead screw with regard to the allongation and the torsion of the spindle, because rotational and transversional acceleration do influence each other directly. The numerical buffer method has been applied successfully.

A system as given in the example (fig.1) can of course also be simulated by use of a FORTRAN-program (or a program written in any other user-oriented

computer language). The development of a CSMP-program is faster - especially for the beginner, the execution on the contrary is much more expensive. For this reason most of the people we know do not make extensive use of CSMP exept for the solution of smaller problems. Complicated problems normally are solved by the development of a FORTRAN-program - which actually is not too difficult either.

Considering once more the example given above, the arrangement of the statements in a FORTRAN-program must be:

```
-- calculation of the motor speed        (motor)
-- calculation of the load torque        (gear)
-- calculation of the acceleration       (motor)
```

The statements for the motor-module and for the gear-module can therefore not be separated. The requirement of modularity can not be garanteed. On the contrary the CSMP-language provides the user with the SORT-option. Between the statements SORT and NOSORT the statements may be entered in any order and will be put into the accurate sequence by the system-program itself. This option enables the user to define the needed modules as CSMP-macros which can be called afterwards as many times as wanted. The arrangement of the statements will be correct, because the call of the macros will be replaced by the whole macro definition before the sorting algorithm is activated. This is very important in the case where the user does not know in advance which might be the final structure of the system he is laying out. In the given example he may not know whether he will use a spur gear or a worm gear, whether he is going to use a DC-motor or a hydraulic motor. In the following it will be shown how a modular program for the simulation of any given electro/hydraulic drive can be built up.

III. DESCRIPTION

Principally two parts of a drive can be distinguished -- the driving part (motor) on one hand and the driven part (gear) on the other hand. If a system with smaller proportions is considered, DC-motors and sometimes stepping motors can be used on the driving side, spur gears or even friction gears on the driven side. In case of a configuration with larger proportions the use of DC-motors is not indicated because of the relatively high value of their inertia, specially if the drive has to react very fast. For such applications hydraulic drives are used on the driving side, preloaded spur gear boxes or - in case of higher gear reduction - preloaded worm gear boxes are used on the driven side. The preload will be given only in case of positioning action in order to avoid back lash.

To apply CSMP on the problem of modular simulation of electro/hydraulic drives, a symbolic library should be built up which contains a big amount of CSMP-macros and FORTRAN-subroutines modeling the different types of servovalves including their torquemotors, pneumatic drives, quill head, the different kinds of gears with stiff and with elastic axle such as spur gear (normal and helical), worm gear, sliding gear, lead screw, recirculating ball screw etc.. Beside these macros some secundary macros are needed to describe effects such as friction in a proper way.

Our work during the last two years has shown that it is possible to describe all needed modules in a way which garantees full compatibility. It is of course impossible to reproduce all equations, each model in this short review, but as an example the proceeding will be explained by developing the modules for the description of the friction.

Let us consider the dry friction first.

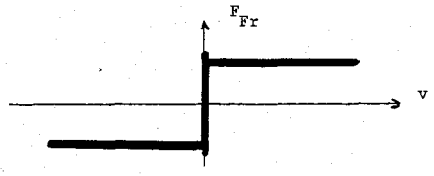

Fig.2 Dry friction in function of the velocity

The block diagram of a system using dry friction would be:

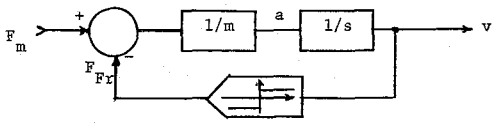

Fig.3 Block diagram of a system with dry friction

It is easy to see that this system is instable if the inequality

$$|F_{Fr}| > |F_m| \qquad (5)$$

is valid. A change of the sign of the velocity (v) will change the sign of the acceleration (a) immediately. This block diagram therefore describes an oscillator with theoretically infinite frequency. By use of CSMP together with a variable step integration method the system-program will try to reduce the integration step until the program stops with the error message "VARIABLE STEP DELT LESS THAN DELMIN". By use of a fixed step integration method an overflow will be produced. In reality equation (5) can only be valid for $v \neq 0$. At $v \equiv 0$ there exists an inequality (6) which reduces F_{Fr} to F_m:

$$|F_{Fr}| \leq |F_m|\Big|_{v\equiv 0} \qquad (6)$$

This inequality should be taken into consideration by developing a simulation program. Beside of this a calculated real variable cannot be checked on zero. Therefore a small hysteresis loop should be introduced. According to these explanations the friction should have a graph as shown in fig.4.

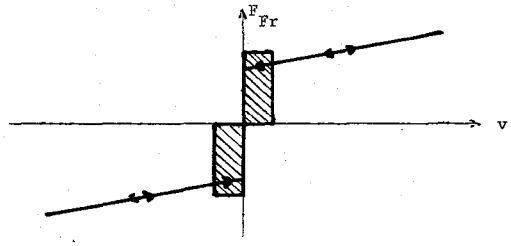

Fig.4 Graph of the friction in function of the velocity

In the hatched area any value can be obtained. The following four routines can be used to simulate the friction in a proper way. The procedural parts of the macros REIB and KRGL are written as FORTRAN-functions to save macro space. REIB calculates the friction without regard to the inequality (6). KRGL corrects the result of REIB by taking (6) into consideration. REIB and KRGL may be used in any other macro to simulate specific friction effects.

```
MACRO  DAEMPF = REIB (THETA, MYHAFT, MYTR, MYVISK, EPS)
H11 = INSW (HSTRSS (-1., 0., EPS, THETA),-MYHAFT,-MYTR)
H12 = INSW (THETA, -MYHAFT, 0.)
H21 = INSW (HSTRSS (1., -EPS, 0., THETA), MYTR, MYHAFT)
H22 = INSW (THETA, 0., MYHAFT)
DAEMPF = VISK (THETA, EPS, MYVISK, H11, H12, H21, H22)
ENDMAC
```

Fig.5a Listing of macro REIB

```
      FUNCTION VISK (THETA,EPS,MYVISK,H11,H12,H21,H22)
      REAL  MYVISK
      R1 = H12 - H11
      R2 = H22 - H21
      R3 = MYVISK*THETA
      IF ((R3.GT.-EPS).AND.(R3.LT.EPS))  R3 = 0.
      VISK = R1 + R2 + R3
      RETURN
      END
```

Fig.5b Listing of function VISK

```
MACRO  MAUS = KRGL (MEIN, MLAST, MREIB, THETA, EPS)
H31 = INSW (HSTRSS (-1., 0., EPS, THETA), -1., 0.)
H32 = INSW (THETA, -1., 0.)
H41 = INSW (HSTRSS (1., -EPS, 0., THETA), 0., 1.)
H42 = INSW (THETA, 0., -1.)
MAUS = VERG (MEIN, MLAST, MREIB, H31, H32, H41, H42)
ENDMAC
```

Fig.5c Listing of macro KRGL

```
      FUNCTION VERG (MEIN,MLAST,MREIB,H31,H32,H41,H42)
      REAL  MEIN, MLAST, MREIB, MAUS, M1, M2
      M1 = H32 - H31
      M2 = H41 + H42
      ENT = M1 + M2
      IF (ENT - 0.5)  1, 1, 2
1     MAUS = MEIN - MLAST - MREIB
      GO TO  5
2     HILF = ABS (MEIN-MLAST) - ABS (MREIB)
      IF (HILF)  3, 3, 4
3     MAUS = 0.
      GO TO  5
4     MAUS = SIGN (HILF, MEIN-MLAST)
5     VERG = MAUS
      RETURN
      END
```

Fig.5d Listing of function VERG

IV. ANALYSIS

The language CSMP in its ordinary implementation is very well equipped for the analysis of systems which once have been described. It was stated before that it would also be possible to analyse a problem by use of FORTRAN or another user-oriented language. But just the complexity of a problem justifies the use of the more expensive problem-oriented language CSMP, if almost the same program can be used different times for the analysis of different systems, as it is possible according to our explanations. Modifying a CSMP-program which takes full advantage of the possibilities of modular programming is much easier and therefore much more reasonable than to modify a FORTRAN-program, because this modification is almost identical with a recommencement of the program development.

Our research has shown that even very complex models of high system order can be analysed with reasonable financial efforts as long as there do not arise too many discontinuities in the variables during the simulation. Such discontinuities may occur due to dry friction or due to switching actions (e.g. modelling of thyristors). The simultaneously occurence of continuous and discontinuous proceedings in a digital simulation program always creates difficulties independent from which language has been used for the simulation. In this case only hybrid computation can help out of the dilemma, because on a hybrid computer parallel and sequential program sections may go on simultaneously. All integration activities can be carried out by the analog part of the hybrid computer.

As an example let us consider a machine-tool consisting of controller, DC-motor (or hydraulic motor with servovalve and torquemotor), preloaded worm gear box, recirculating ball screw, quill head, nonlinear recording of the instantaneous value, feedback filter, comparison of the set point and the filtered instantaneous value. This system has the order 13 and uses 7 times the friction modules (fig.5a-d) which corresponds to 56 INSW-functions. The simulated system therefore contains a big amount of switching actions and is of high grade of nonlinearity. It has been simulated on IBM 370. Four runs have been accomplished with different values of elasticity of the spindle (different diameter, different quill screwing). Compilation and execution phase needed together 7 min 50 sec CPU-time. Fig.9 (at the end of the paper) shows the result of one simulation run.

One interesting point is the dead time of the velocity (XDOT) at the beginning. The elasticity is the mechanical analogon to the electric transmission line. It represents therefore a system with distributed parameters which has to be described by partial derivatives. As known from the line theory it is possible to model a transmission line through an infinite number of concentrated elements as shown in fig.6.

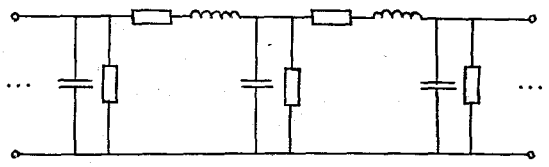

Fig.6 Model of an electrical transmission line

It is often accurate enough to replace this infinite number of components by a very small number using a T-section or a π-section instead. The results from the line theory can of course also be applied to the mechanical problem.

In our simulation we have used one π-network for every elasticity of the system. The result (fig.9) shows on one hand the accuracy of the model description and on the other hand it proves the aptitude of the applied simulation package for the system analysis.

V. SYNTHESIS

Up to 1972 it was almost impossible to practise any synthesis work by using CSMP. The new version CSMP III principally provides the user with the possibility of practising system synthesis. This is possible owing to the new subroutines RERUN, CONTIN and FINISH which enable the user to control the progress of the simulation dynamically. Nevertheless no optimization routine has been integrated into the system until now. The user is therefore forced to write his own optimization routines or - what even is more efficient - to adapt existing FORTRAN optimization routines

(e.g. IBM-Scientific Subroutine Package SSP, the subroutines: FMFP and FMCG) to the possibilities given in CSMP. This application will be explained by the following.

All optimization routines we know are organized in the way, where a subroutine containing the optimization algorithm for the evaluation of a new parameter-vector (PAR) is called with an initial guess for the parameter values (ICPAR). This subroutine itself calls a function (or a subroutine) in which the performance index (PI) has to be calculated in function of the actual parameter-vector (PAR) and which has to be added by the user.

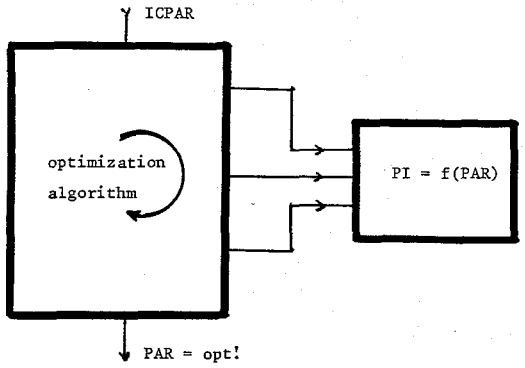

Fig.7 Organization of optimization routines

This "function" need not to be an algebraic assignment, it may as well contain a whole simulation run for a continuous dynamic system. In this case application of CSMP may be desirable. The problem occuring is that the user-written "function" should now be the CSMP-program itself. The organization must therefore be turned over. This reorganization can be obtained very easily as shown in fig.8. The optimization routine OPT may need k calls of the function FUNCT. The introduced integer variable JUMP must be set to 1 in a parameter list. A call of OPT in the TERMINAL-segment of the model will initialize the optimization algorithm.

VI. ACKNOWLEDGEMENT

We would like to express our thanks to AGIE Ltd. for their scientific support. Without their intensive cooperation it would have been impossible for us to realize the research described in this article. Likewise we wish to express our gratitude towards Prof.Dr.M.Mansour, the head of our institute at the Federal Polytechnical Institute, who gave us the opportunity to carry out this interesting research work.

REFERENCES

[1] IBM, Continuous System Modeling Program III (CSMP III) Program Reference Manual, Form SH19-7001-0

[2] Herbert E. Merritt, Hydraulic Control Systems, John Wiley & Sons, Inc. New York/London/Sydney 1967

[3] Y.H.Ku, Analysis and Control of Linear Systems, International Textbook Company, Scranton Pennsylvania, Chapter 11

[4] Gerhard Niemeyer, Systemsimulation, Akademische Verlagsgesellschaft, Frankfurt am Main 1973

1st call of function FUNCT

ith call of function FUNCT

kth call of function FUNCT

```
      SUBROUTINE OPT (FUNCT,...,ICPAR,PAR,...)
      DIMENSION ...
                .
                .
                .
      PI = FUNCT (ICPAR)
                .
                .
                .
      PI = FUNCT (PAR)
                .
                .
                .
      PI = FUNCT (PAR)
                .
                .
                .
      RETURN
      END
```

```
      SUBROUTINE OPT (JUMP,...,ICPAR,PAR,...)
      DIMENSION ...
      GO TO (1,2,3,...,k+1),JUMP
1     CONTINUE
                .
                .
      JUMP = 2
      CALL RERUN
      RETURN
2     CONTINUE
                .
                .
      JUMP = i+1
      CALL RERUN
      RETURN
i+1   CONTINUE
                .
                .
      JUMP = k+1
      CALL RERUN
      RETURN
k+1   CONTINUE
                .
                .
      RETURN
      END
```

Fig.8 Modification of optimization routines

SIMULATION OF MACHINE-TOOLS

	-160.0	'P'=PMOT	160.0			
	-.2400	'V'=XDOT	8.000E-02			
	.0	'X'=X	3.200E-02			
TIME	X			XDOT	PMOT	UR
.0	3.0000E-02	I----------I----------P----------V----------X--I		.0	.0	5.0000
4.0000E-03	3.0001E-02	I PI V X I		2.8238E-06	77.402	-5.4094
8.0000E-03	3.0001E-02	I I V PX I		-3.6079E-05	138.53	-5.2377
1.2000E-02	3.0001E-02	I I V P X I		-2.8801E-04	124.12	-5.0431
1.6000E-02	3.0001E-02	I I V P X I		-8.3745E-05	96.902	-5.0025
2.0000E-02	3.0000E-02	I I PV X I		-1.0979E-04	72.142	-4.9988
2.4000E-02	2.9996E-02	I I P V X I		-2.6401E-03	53.066	-4.9995
2.8000E-02	2.9978E-02	I I P V I X I		-1.2390E-02	39.573	-4.9999
3.2000E-02	2.9853E-02	I I VP I X I		-5.4311E-02	30.510	-5.0000
3.6000E-02	2.9520E-02	I I V I P I X I		-.11332	24.611	-5.0000
4.0000E-02	2.8949E-02	I----------V-I----------I---P----------I---------X----I		-.17066	20.807	-5.0000
4.4000E-02	2.8180E-02	I V I I P I X I		-.20986	18.304	-5.0000
4.8000E-02	2.7307E-02	I V I I P I X I		-.22149	16.558	-5.0000
5.2000E-02	2.6446E-02	I V I I P I X I		-.20516	15.250	-5.0000
5.6000E-02	2.5693E-02	I V I I P I X I		-.16875	14.268	-5.0000
6.0000E-02	2.5105E-02	I V I I P I X I		-.12518	13.508	-5.0000
6.4000E-02	2.4683E-02	I V I P I X I		-8.8122E-02	12.973	-5.0000
6.8000E-02	2.4378E-02	I I P IX I		-6.7835E-02	12.678	-5.0000
7.2000E-02	2.4112E-02	I I P X I		-6.8542E-02	12.582	-5.0000
7.6000E-02	2.3804E-02	I V I P X I		-8.7810E-02	12.654	-5.0000
8.0000E-02	2.3395E-02	I----------I-------V------I-P----------XI----------I		-.11799	12.801	-5.0000
8.4000E-02	2.2859E-02	I V I P X I I		-.14900	12.962	-5.0000
8.8000E-02	2.2214E-02	I V I I P X I I		-.17162	13.069	-5.0000
9.2000E-02	2.1506E-02	I V I I P X I I		-.18006	13.086	-5.0000
9.6000E-02	2.0794E-02	I V I I P X I I		-.17333	13.005	-5.0000
.10000	2.0135E-02	I IV I P X I I		-.15504	12.871	-5.0000
.10400	1.9561E-02	I V I P X I I		-.13181	12.715	-5.0000
.10800	1.9077E-02	I V I P X I I		-.11103	12.585	-5.0000
.11200	1.8661E-02	I V I P X I I		-9.8562E-02	12.506	-5.0000
.11600	1.8273E-02	I V I P X I I		-9.7216E-02	12.511	-5.0000
.12000	1.7870E-02	I----------I---------V----I-P-X----------I----------I		-.10620	12.591	-5.0000
.12400	1.7415E-02	I V I PX I I		-.12177	12.702	-5.0000
.12800	1.6894E-02	I V I P I I		-.13863	12.808	-5.0000
.13200	1.6311E-02	I IV IXP I I		-.15166	12.885	-5.0000
.13600	1.5690E-02	I V XI P I I		-.15742	12.912	-5.0000
.14000	1.5063E-02	I IV X I P I I		-.15499	12.884	-5.0000
.14400	1.4459E-02	I I V X I P I I		-.14595	12.818	-5.0000
.14800	1.3900E-02	I V X I P I I		-.13368	12.738	-5.0000
.15200	1.3389E-02	I V X I P I I		-.12213	12.666	-5.0000
.15600	1.2917E-02	I VX I P I I		-.11465	12.623	-5.0000
.16000	1.2464E-02	I----------I-------XV-----I-P----------I----------I		-.11301	12.620	-5.0000
.16400	1.2006E-02	I XV I P I I		-.11707	12.652	-5.0000
.16800	1.1522E-02	I V I P I I		-.12502	12.708	-5.0000
.17200	1.1004E-02	I V I P I I		-.13411	12.766	-5.0000
.17600	1.0452E-02	I VX I P I I		-.14154	12.812	-5.0000
.18000	9.8767E-03	I V I P I I		-.14528	12.835	-5.0000
.18400	9.2955E-03	I XV I P I I		-.14462	12.823	-5.0000
.18800	8.7248E-03	I X V I P I I		-.14023	12.791	-5.0000
.19200	8.1764E-03	I X V I P I I		-.13380	12.748	-5.0000
.19600	7.6544E-03	I XI V I P I I		-.12744	12.707	-5.0000
.20000	7.1544E-03	I----------X-I------V-------I-P----------I----------I		-.12302	12.682	-5.0000
.20400	6.6661E-03	I X I V I P I I		-.12165	12.675	-5.0000
.20800	6.1770E-03	I X I V I P I I		-.12340	12.690	-5.0000
.21200	5.6760E-03	I X I V P I I		-.12741	-10.686	-4.6969
.21600	5.1576E-03	I X P I V I I		-.13149	-98.462	-3.4263
.22000	4.6305E-03	I P X I V I I		-.13081	-136.47	-2.1062
.22400	4.1247E-03	I P X I V I I		-.12002	-129.86	-.80931
.22800	3.6884E-03	I X P I V I I		-9.5807E-02	-91.876	.36472
.23200	3.3760E-03	I X I P I V I I		-5.8477E-02	-41.241	1.2833
.23600	3.2326E-03	I X I P V I I		-1.2394E-02	.12426	1.8203
.23703	3.2263E-03	I X I P V I		-9.0607E-05	1.9563	1.8854

Fig.9 Simulation run of machine-tool