A MULTI-TASK SYSTEM FOR ROBOT PROGRAMMING.

by Giuseppina Gini, Maria Gini, Renato Gini, Dario Giuse.
Istituto di Elettrotecnica ed Elettronica
Politecnico, Milan, Italy

ABSTRACT

In this paper we discuss issues of design for software systems for computer controlled manipulators.

The aim of the paper is to present the experience obtained in designing and implementing MAL, a stand-alone software system for controlling and programming a two-arms manipulator.

That system supplies the user a simple multi-task programming language, with a BASIC-like external form, and operating instructions for compiling, executing, editing and saving the program.

We will emphasize how management of multiprocess capabilities, synchronization of different devices, error handling and other desirable features can be inserted in a simple system, implemented on a small minicomputer, suitable for industrial applications.

I - INTRODUCTION

Since computer controlled manipulators were introduced as general purpose mechanisms for industrial automation, the methodology of controlling and programming them for specific tasks has seen a great deal of development.

Programmable automation consists of a set of multidegree-of-freedom manipulators and sensors under computer control; these systems can be programmed to perform a task and can be applied to new jobs by reprogramming them. Although programmability is an important aspect of these systems, the development of software for controlling and programming them has just started.

The only programming method in common industrial use today is called "teach mode"; in that case, no text of the program, expressed in any programming language, is obtained, but only a sequence of absolute manipulator positions is memorized.

The research in programming languages has widely demonstrated the importance of developing software systems for robot programming [1][7][8][9]. Flexibility, generality, ease in reprogramming, documentability, are only the most important advantages produced by the introduction of a software system, while only few shortcomings are added. It is, of course, harder to express by a formal language the human experience about how the manipulator hardware can be used to accomplish a task.

Previous researches in Machine Intelligence have been primarly directed to general and very-high-level systems, while

little attention has been paid to computational costs, programming difficulties, real applications. Now the decreasing in the costs of computer components and the large introduction of microcomputers make it possible a new era in programmable industrial manipulation.

The design of complete and simple programming systems for automation represents an important research area, in which some real-time software problems can be approached together with some software-engineering requirements. In this direction MAL (Multipurpose Assembly Language) has been designed and implemented at Milan Polytechnic.

MAL is a complete software system for controlling and programming the Supersigma robot, a two-arms robot mainly designed for assembly operations. MAL was designed with emphasis on portability to different computers and to different robots, ease in programming and multi-task organization. It is a stand-alone system, supporting interactive editing, compilation and execution of programs.

Besides, the programmability of the Supersigma robot is extended to different levels. The control structure, constituted by a set of microcomputers, allows programmability at a level usually fixed by the hardware [2] . The connection with a general purpose computer allows developing higher-level software aimed at managing error recovery and decision taking [4] .

II - THE PHILOSOPHY OF ROBOT PROGRAMMING

For a robot system, programmability means the system is able to learn how to perform a task from its human teacher. Nevertheless the direction of developing programming languages for describing assembly tasks has not been widely investigated.

In industrial applications robots are commonly programmed by guiding the mechanical device through the sequence of operations required to perform the assembly process. A joystick or a button box is used to insert in the control memory the positions that must be remenbered. The position sequence may be played back, to cause the arm to accomplish the task. This method, commonly called teach-mode, is a non-textual approach. It does not require to write a program and does not require to associate abstract symbols with manipulator movements.

The execution of the task is obtained playing back a fixed sequence of movements, and the impossibility of expressing conditional actions makes it impossible to use force sensors or to introduce any adaptation, while the lack of a text produces the impossibility of maintaining, documenting and modifying the program.

We will develop textual languages for robot programming because a text can be read by users, can be saved in an understandable form and can fit different situations.

Textual languages have the big advantage of introducing variables, control structures and interface with people. Variables allow to obtain information during the execution and to modify the behavior of the program according to the results of tests. Control structures allow branching and conditional activity according to the sensor output. Interface with people allows editing, modifying and documenting programs, and supplies facilities in programming.

The textual approach to robot programming introduces in robotics the philosophy and the experience of software system design. In fact, new languages for robot programming are necessary because general purpose languages are generally not adequate.

Two directions in robot software design, as described in [8], may be noted. The first of them, called explicit-programming, requires explicit instructions for every action the robot must take. The second, called world-modeling, tries to make the robot responsible for taking some decisions according to its knowledge. While explicit-programming systems may be inserted by now in factory uses, world-modeling systems are in the research stage. They tend to require a lot of computer power, but they are able to accept very abstract and high-level instructions [1] [3] [5] [6] .

Because of some practical reasons, as our limited computer facilities, and because of the aim of our project at developing systems suitable for industrial uses, we decided to develop an explicit-programming system.

In the next section we will illustrate MAL, an explicit-programming system that introduces new features in the standard of the present manipulator languages. Because MAL is operational al Milan Polytechnic, we will illustrate the project lines, the characteristics of implementation and the way of using it.

III - DESIGN CRITERIA OF MAL

Our research in multi-task systems for robot programming was motivated by the need of developing the programming system for the Supersigma robot. Supersigma is a cartesian axes manipulator, with two arms. Each arm has three degrees of freedom plus the hand opening. Its mechanical structure is the same as Sigma robot of Olivetti Co., while its electronic control has been completely redesigned and implemented with a set of microcomputers [2] , each one devoted to the control of one motor, and using a minicomputer as CPU. The structure of the system is illustrated in Fig. 1.

The lack of an adequate control and programming system for our robot convinced us to design and implement a complete software system supporting the writing, the verification and the execution of programs. This system is called MAL (Multipurpose Assembly Language).

Moreover, the lack of similar systems adequate for industrial applications constituted an important incentive to develop a software system for general use in computer controlled manipulation.

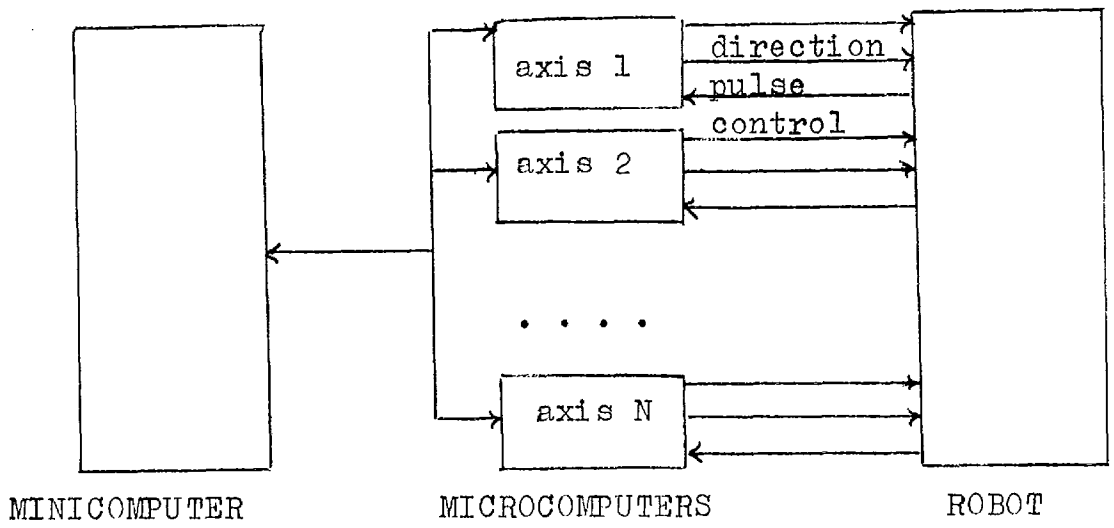

Fig. 1

We will now illustrate the major design issues we bore in mind and we inserted into MAL.

In our opinion, the language used to describe assembly tasks should be unspecialized, in the sense that it should not be a single-problem oriented language. The external form should be as similar as possible to the form of some widely known language, to increase the readability of the program and to reduce the training time. MAL has a BASIC-like external form; the choice of BASIC is motivated by the wide use of this language in industrial applications, the ease in editing and modifying the program, the readability with respect to other languages used for industrial control.

A language for robot programming should minimize the number of steps between the initial writing of the program and the final testing. This organization is important in real applications because it reduces the time necessary to program the robot for a new task and increases the reliability of the system. With MAL the turn-around time after a revision is very short, and the program has not to be restarted after a change. This last feature is called "hot editing" [9]. It may be very easily obtained in an interpreter; in our system it is obtained by partially compiling the program into an internal form which can be fastly interpreted.

The parallel execution of different tasks is another fundamental feature of a robot programming system. In the case of a robot with two or more arms that aspect is particularly important since it makes it possible to write a different task for each arm. Parallel programming is useful when different control activities can be executed by a robot co-operating with external devices. MAL allows the independent programming of different tasks and provides semaphores for synchronization.

Other design issues for a robot programming system are more related to implementation aspects.

The system should run on a small computer because costs are

a fundamental aspect in industrial applications. The minicomputer we employed has 24K memory words of 16 bits; MAL requires less than 20K words, although it is written in FORTRAN and not optimized.

Last, the system should be portable. Portability for a control language has to be defined not only in relation to the computer but also to the manipulator. MAL is implemented in FORTRAN IV (except for a small interface to the robot which is written in assembler). Moreover MAL is implemented in such a way that the change of the controlled robot would not require a complete rewriting of the system. For instance, the conversion from a cartesian robot to a polar one should require changes only to a given module; likewise, the system should be able to control more than one robot, possibly working in co-operation, at the same time.

IV - PROGRAMMING BY MAL

MAL is made up by two different parts, one devoted to the compilation of the input language into an internal form, the other devoted to the execution.

The compilation part gives the user facilities to create, update and maintain the source program. The execution part has the responsibility of executing the sequence of operations described in the user program. The debugging of the program is easy, and the user can modify the text and immediately check it.

To develop a program the user has to express the assembly task as a sequence of elementary operations. If the task requires some parallel activities, the programmer writes the different parts as they are independent and then synchronizes them. MAL compiler translates the program into an easily interpretable object code. The user may, at any given moment, list his program, modify it and save it on a mass storage.

The system is line-oriented; after an instruction has been typed in, the compiler checks for syntactic errors and eventually gives the appropriate error message. When the program is complete the user may ask the execution of it.

The program may be partially executed, by stopping it in correspondance with an instruction or by typing a command from the teletype. The execution can be restarted at any point of the program, because the values of the variables and the physical positions of the arms have not been changed. That feature, called hot editing, is considered one of the most important requirements for manipulator software.

Let us now insist on independent programming of different activities to be executed in parallel; that seems to be a fundamental aspect of robot programming. Parallel programming is useful for multi-arm robots and when different control activities can be carried out by a single-arm robot together with other mechanisms (as moving belts). Besides parallel activities are use-

ful when different robots interact.

To execute parallel tasks, we ought to isolate every logically independent activity in order to minimize the interactions between them. Every activity is separately programmed as independent. The second step is to insert into the program the instructions which manage synchronization and information flow between tasks. These instructions are tests on semaphore variables.

We will now show a simple MAL program for repetitively moving an object from a position to another through an intermediate position. Let us suppose the left arm moves the object to the position X = MP, and the right arm moves it to the final position X = FP, as illustrated in Fig. 2.

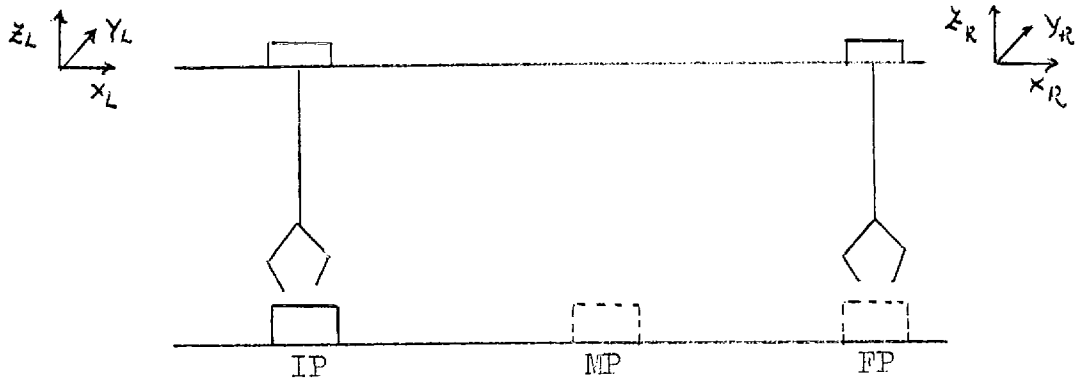

Fig. 2

In the programs controlling each arm we insert synchronization instructions according to the following conditions:

i) no arm can reach MP when the other arm is in it;

ii) the right arm can go and grasp the object in MP only after the left arm has ungrasped it in MP.

The first condition is handled by means of the semaphore variable FREEAREA, which is set to YES only when the area upon MP is free. The second condition is managed by OBJECTREADY, whose value is YES when the object is placed in MP.

The complete MAL program is:

```
5     SET IP=1Ø, MP=15Ø.5, FP=7ØØ, TABLE=15.7, UP=TABLE+4Ø
6     SET LEFTHAND = 8
7 SET YES = Ø, NO = 1, OBJECTREADY = NO
8 TASK 2,15Ø
9 "--- TASK NO. 1 - LEFT ARM ---
1Ø    MOVE W ZL = UP
15 SET FREEAREA = YES
2Ø    MOVE W XL = IP
3Ø    MOVE W ZL = TABLE
4Ø    ACT LEFTHAND                    "close left hand
5Ø    MOVE W ZL = UP
55 WAIT FREEAREA                      "may I access my area?
```

```
 57 SET FREEAREA = NO                "yes. It's my own now!
 6Ø    MOVE W XL = MP
 7Ø    MOVE W ZL = TABLE
 8Ø    DEACT LEFTHAND                "open left hand
 85 SET OBJECTREADY = YES            "object is in MP
 9Ø    GO TO 1Ø                      "repeat the sequence
1Ø6    SET RIGHTHAND = 9
1Ø9 "--- TASK NO. 2 - RIGHT ARM ---
11Ø    MOVE W ZR = UP
115 SET FREEAREA = YES
12Ø    MOVE W XR = FP
13Ø    MOVE W ZR = TABLE
14Ø    DEACT RIGHTHAND               "open right hand
15Ø    MOVE W ZR = UP
155 WAIT OBJECTREADY, FREEAREA       "waiting for an object in MP
157 SET FREEAREA = NO
16Ø    MOVE W XR = MP
17Ø    MOVE W ZR = TABLE
18Ø    ACT RIGHTHAND                 "close right hand
185 SET OBJECTREADY = NO             "no object in MP
19Ø    GO TO 11Ø
```

Only a few notes about that program. The values assigned are in millimeters. TABLE is the height of the working plane, UP the minimum hand elevation to avoid collisions, LEFTHAND and RIGHTHAND represent the hands. The W after MOVE means that the next instruction will be executed only after the accomplishment of the movement.

The programs for the two arms are the same; for making it clearer we incremented by 1ØØ the line numbers of the second task with respect to the first. The indentation is used to show the instructions required for cooperation. The right arm starts operating only when both conditions of line 155 are met (object present and area free). The two tasks are executed at the same time, because all the actions not referring to the common area are executed by each task if they were the only present in the system.

It would be possible to program the same job using only one arm, but in this case the advantages of a two-arms manipulator would be lost.

In a real operation we will modify every time the position X or Y to reach objects in their actual position. Here we suppose IP and FP moving belts.

V - CONCLUSIONS

We presented modern trends in robot programming and we discussed about our experience in designing and implementing a language for assembly, MAL.

Our project is still in evolution, because we intend to in-

vestigate how to extend the interactivity of the system and how to make it really universal. For that reason we are now starting defining a new abstract machine, whose instructions will be interpreted at the execution time. That machine should be universal (the machine language for every robot) and should improve the portability of the system to different computers. A very-high-level language for robot programming will be then easily implemented by translating the external language into the abstract machine language, and in that case we will obtain an universal system.

REFERENCES

1. Binford et al. "Exploratory study of computer integrated assembly systems", Progress Report 4, Stanford Artificial Intelligence Laboratory Memo AIM-285.4, Stanford, Ca, August 1977.

2. Cassinis, R., Mezzalira, L. "A multimicroprocessor system for the control of an industrial robot", Proc. 7th ISIR, Tokio, Japan, 1977.

3. Finkel, R. et al. "An overview of AL, a programming system for automation", Proc. 4th IJCAI, Tbilisi, USSR, 1975.

4. Gini G., et al. "Emergency recovery in intelligent robots" Proc. 5th ISIR, Chicago, Illinois, 1975.

5. Lieberman, L.I., Wesley, M.A. "AUTOPASS: an automatic programming system for computer controlled mechanical assembly", IBM Journal of Research and Development, July 1977.

6. Lozano-Perez,T., Winston, P.H. "LAMA: a language for automatic mechanical assembly", Proc. 5th IJCAI, Boston, Mass, 1977.

7. Nevins, J. et al. "Exploratory research in industrial modular assembly", Draper Lab., 5th Report, Cambridge, Mass, September 1977.

8. Park, W. "Minicomputer software organization for control of industrial robots", Proc. JACC, San Francisco, Ca, 1977.

9. Rosen, C. et al "Machine intelligence research applied to industrial automation", 7th Report, SRI International, Ca, August 1977.