Robot Languages in the Eighties

Giuseppina Gini
Dept. of Electronics, Politecnico, Milan, Italy

Maria Gini
Dept. of Computer Science, University of Minnesota, Minneapolis, USA

ABSTRACT

The scenario of general-purpose programming systems is rapidly changing; what are the consequences for robot programming? The programming environments built around ADA, UNIX, and Interlisp are useful for robot programming? After introducing the peculiar aspects of robot programming we will discuss some examples of general-purpose languages applied to robots and languages specifically designed for robotics. Criteria for making a choice between the two approaches should take into account the present state of the art. The need of a strong integration between different components as robots, vision systems, and other automation equipment could support the first approach. The solution of robot and robot users problems has, until now, supported the second approach, as indicated by the choices of European robot manufacturers. Anyway the expression of actions taken by different intelligent agents, as robots can be defined in the future, will require absolutely new linguistic media. We hope that robot programming in the eighties can develop experiences useful for the assessment in the field.

INTRODUCTION

Today robot applications are generally carried out in an integrated industrial setting, where robots and other equipment manipulate and sense parts to repeatedly perform a task. We may see a typical setting as consisting of a conveyor belt which transports parts, usually partially oriented and separated, a vision system getting information about the incoming parts, and a robot making assembly or inspection or serving other machines. Those robots are sometimes programmed on line, guiding them through their task and storing positions and operations into a memory. The drawbacks of this methods have been stressed enough. Errors during the programming require to restart again, teaching of many positions is too long and error prone, the programming time is not productive because all the robot related equipment must be stopped during new task programming, synchronization with other equipment is hard, sensors cannot be used to modify actions during program execution. During recent years we have seen a significant change in the attitude of robot manufacturers, and almost every new robot is today

K. Rathmill et al. (eds.), *Robot Technology and Applications*

sold with an off line programming system. Ten years of experiences have passed and manipulator level languages are today accepted. Usually we use them to give the cartesian reference frames where the manipulator hand should be moved in order to accomplish the task. Those frames or their sequence can be modified by run-time events, as sensor output or external synchronization signals. In some early systems every joint of the manipulator was individually given a value in its own coordinate system. Programming in terms of frames is not yet a good solution because it requires a lot of detailed information be given by the programmer, who is expected to have some skills in mathematics and programming. The writing of robot programs is not so easy as one might think. It is quite difficult to understand and use positions in 3D space, and this is the basis of robot programming now.

An intelligent robot (20) will be given only a task and the spared parts and materials needed to perform it, instead of programs embedding the complete sequence of actions to be taken. An issue to be addressed by more advanced systems is world representation and object modelling. A representation of the world should allow the robot to manipulate parts and to sense the environment. This model should be a copy of the real world. Actually no world model can be complete, so at least it should be detailed and rich enough. Present industrial robots do not have any geometrical world model; their knowledge is encrypted into variables and data structures which have only meaning for their human programmer. An integration of modelling used in design and production is highly desirable. Many systems are today commercially available for CAD/CAM applications. Most of them are purely graphics systems, without much interest for robotics. The models used by CAD systems may be useful in robotics, even though robot relevant information, as the center of gravity, the mass, the grasping point, etc. of the object are not contained in CAD models. The experience made in the RAPT system is that using CAD systems world modelling is still long and tedious. Moreover the use of CAD data bases for defining world models to be used also for sensorial tasks (for example for vision understanding) is not yet acceptable.

Besides these formidable research issues robots are today in use. The practical solution taken has been to limit the intelligence and the understanding of the robot and to reduce programming of robots to usual programming or, in a few cases, to parallel programming. No use of world models is done in any commercial system.

Many industrial issues in robot programming are still open. How to program robot completely off-line, and how to express the integration of different devices, being the most studied. The present status of software and software engineering has here its impact. Having removed from robot software the problem of intelligence has reduced robot software to a sophisticated software engineering problem: software solutions should work there.

ROBOT PROGRAMMING

The most obvious ways to implement robot languages are to adapt a general-purpose programming system or to develop robot specific programming systems. In the second case new languages may be developed from scratch or from existing automation languages, such as APT.

The first solution is appealing because education and training of robot programmers can be reduced in time and the development of robot applications will be reduced to the writing of a few routines. The second solution is appealing because robot programmers can be not computer expert and what they have to learn will be exactly what they need to use. In the case of APT the same NC programmers could be easily converted into robot programmers.

The first solution has as a shortcoming that it heavily relies on what is available for general purpose programming. The computer-user interface is not tailored on the specific needs. The second solution has as a shortcoming that many development efforts are often spent only to provide something already available with minor differences. In the case of APT like languages we may also argue whether robot programming is the same as NC machinery programming.

Since a language is a way to express or to test different solutions, we may say that all the developments so far obtained have demonstrated what can be done today at the manipulator level programming. After that, the choice of using existing languages or developing new ones is a problem of market image and acceptance. We may hardly see the need to develop new languages. The only important difference is in the programming environment. The development of programs off line requires sophisticated programming environments which are not generally available in standard languages as FORTRAN or even PASCAL. The choice to develop a complete programming environment is the only we see still pushing towards the development of new systems. A set of subroutines written in FORTRAN are long to edit, write, debug.

If the language should demonstrate new solutions or in other word should allow task oriented programming then we may still want to maintain manipulator level programming as target language for planning and sensorial activities. The expression of those activities not necessarily should resemble any of the existing programming languages. Graphics, natural language, or sometimes mathematical equations may be all useful in providing ways to express tasks.

Many papers have reviewed existing robot programming languages, between them (7,22). We do not intend to review all the existent robot programming systems. We want only to see what new ideas have emerged from a group of them and whether they are assessed or waiting to be fully explored.

Usually two systems are strictly integrated in a robot programming system. The *user language*, in which application programs are written, and the *run-time system* which executes the code generated by the language translator. This solution is similar, for instance, to the one used in most of the Pascal systems. It may be used as a way to standardize user languages by changing only the run-time system, as done in VAL (27), developed for PUMA robots and then implemented on other Unimation robots. Often the run time system run on micro computers and is written in assembly languages. This trend could change when more and more cheap computer power will make reasonable to write all the software in high level languages.

An issue to be newly addressed is that of defining standard software interfaces between the robot and the user level software. What kind of information we pass to the robot? joint positions, or frames? In which order? How we may ask for a point to point execution? for a continuous path? There are no reasons why a cartesian robot and a polar robot should be programmed in completely different ways. While the run time system is well tailored to the specific hardware in use, the user level language should be problem oriented more than manipulator oriented. If this standard software interface would be provided and accepted by any robot manufacturer we may get any robot language to work for any robot; homogeneity and modularity will also be valuable in industrial settings. We mention here the CAM-I project of standardization in robot software (33). They have individuated five main components of robot software: robot language, robot simulator, robot controller, robot modeler, teaching system. They have put Artificial Intelligence parts in the CAD/CAM environment, and this seems more a decision not to deal now with difficult problems as the ones open in perception and decision making. In our opinion most of them should be solved at the robot level, being the robot the flexible and adaptable entity of the FMS.

We do not intend to give here a complete list of reference terms to compare robot programming languages. What we had in mind in giving the following descriptions is based on the following key words.

1. expression of movements (joint level, hand level, object level);
2. expression of trajectories;
3. use of sensors;
4. class of the language (Pascal, Basic, functional);
5. implementation (Interpreter, compiler, programming system);
6. I/O: ports, functions provided,integration with other equipment;
7. multitasking, synchronization, and parallelism;
8. integration with CAD/CAM for simulation, planning, control.

LANGUAGES AND SOFTWARE ENVIRONMENTS

We may look at general purpose languages as candidates for robot programming. Some experiences have been made of using Pascal for robot applications. See for instance PASRO (4) as a working example of this. Even Pascal however presents some shortcomings as a language for automation. Among them, Pascal doesn't support cooperation, which can be obtained at the

operating system level, moreover, it is poor in file management, and file management would occur very often in integrated manufacturing. Other solutions have been tried, for instance using Concurrent Pascal, a small language developed around Pascal to define and execute concurrent tasks.

Languages for automation did not exist before the introduction of robots. The only exception is APT, the language for NC programming. APT in fact has been chosen as a basis for robot languages in two projects, at least, ROBEX (32) and RAPT (25). Even though, its use in robotics has not yet been demonstrated truly useful.

Software environments for general purpose programming are now available on most computers. The most complete and advanced software environments are today those of UNIX and Interlisp, while the situation of ADA is not yet assessed. The main advantage of a software environment over a simple compiler of a language is that the first provides an unified approach for all the problems encountered in the project, the development, and the test of the program. In the following we will briefly review those three systems, Unix, Interlisp, and ADA.

UNIX - The output redirection and the pipeline mechanism to connect programs are between the most useful characteristics of Unix to make modular programming easy. Different programs can be connected in any meaningful way. Unix operating system is very popular on the scientific personal workstations, and some of them are intended for CAD/CAM applications. Unix can become the standard operating system for CAD/CAM applications, perhaps not for robot programming. The computers used today to move robots usually runs both the run-time system and the user level language. To reduce the hardware costs they usually are stand alone systems, without any standard operating system. Only very sophisticated robots, not today on the market, could justify the high cost of using a sophisticated computer for their own needs.

INTERLISP - The only experience so far reported of using LISP for robot control and programming has been done at the MIT Artificial Intelligence Lab. where Mini (28), an extension to LISP to deal with real time interrupts and I/O, was used to program a robot equipped with a force sensor. LISP has not yet obtained the interest of robot manufacturers. Many reasons for that can be envisaged. Only recently LISP has been made commercially available on mini and microcomputers, it requires a lot of central memory, it is inefficient in mathematical computations and array management. Even though most of those shortcomings are true LISP can be considered now with a great interest for two reasons. The functional stile of programming which is at the basis of LISP (even though Mini is not an example of functional programming; it used a lot of SET instructions!) makes it interesting because languages embedded in LISP are completely extensible so that all the manipulation functions could be modified by the user. We will discuss the functional style of programming in the following. A second good reason for using

LISP is that robot programming tends to use more and more artificial intelligence techniques, and LISP is still considered the main Artificial Intelligence language. Serious Lisp development requires several software components still not available in even the UNIX environment. This reason makes still LISP an intensive memory user, and is the reason of developing LISP machines to make the best use of the computer. Unfortunately those machines are too expensive for any use at the factory level now.

ADA - The main motivation for using ADA (10) in robot programming is that ADA is a structured and complete computer language, and offers some advanced tools as extensibility, modularity, real time capabilities, strong type checking to increase the programs reliability. Moreover it has been designed to be the only language of the eighties and claim has been made that ADA could substitute any language from the assembler level to highest levels. The importance of a complete programming language for robotics applications has been made clear: we want robots to cooperate with other equipment, and this requires task synchronization and coordination of different and sometimes non trivial tasks. Such programming and coordination is not provided by most of the robot programming languages in industrial use today. We do not know examples of robot run time systems written in ADA while we may find examples of other subsystems developed in ADA. Vision is one of them. A long practice in vision programming has been to choose C or Pascal; ADA is a superset of them and its use should solve more problems than it can open. The experience so far developed in Ann Arbor (31) has demonstrated that the use of packages (a way to simplify program encapsulation) and generic packages (a way to implement abstract data types) can simplify the development of complex software making it easier to distribute tasks to different people, to integrate them, to modify the manufacturing cell without complex software modifications. On the other end, the big size of ADA can be a problem. Many features of ADA are hardly useful in industrial automation, but their presence makes ADA compilers big and ADA language difficult to use. In no way people able today to manually guide a robot to program it can be able to write ADA applications. On the other hand, the definition of the professional requirements and education of robot programmers is far to be reached. ADA has many chances to be a reasonable solution for programming robotics cells. The main shortcoming of this philosophy of algorithmic and explicit programming is that it is unsuitable to deal with a complicated cell in which many events may happen, time and sequence constraints can be meet by different solutions. In this case expert systems, as for instance GARI (9) seem a more flexible and understandable way of planning and controlling the cell.

FUNCTIONAL LANGUAGES AND LOGIC PROGRAMMING

After 1980 a lot of literature has stressed the idea that the future of computer languages can be different from the present in some radical way. The evolution of languages as we have so far seen from Algol to Pascal to Ada can be a dead end for computing. Those languages are based

strictly on the Von Neumann architecture of computers, and that architecture is unlike to continue in the future computer generations because it has an unnecessary bottleneck in accessing memory. Languages using assignments access memory one word at a time, and assignments make user languages more suitable to the way computers operate than to the way humans think. The next generation of computers should avoid this bottleneck. Many architectural solutions are available for that, all of them relay on using functional languages. What makes a functional language attractive is its problem-oriented expression, because it expresses functions and doesn't care about memory locations, is its way to be implemented as a very restricted kernel (the function definition and composition operators) and then to grow in every way using user defined functions, it is suitability to run on largely distributed architectures because it does not produce side effects (no global memory is used). We have found an accent on functionality in many robot languages. Mini (28), LAMA-S (12), AML (29) , and Lenny (30) have provided some way to obtain functional capabilities, mainly extensibility.

Another language with full functional capabilities is *Prolog*. It is also the most successful language for logic programming. The Japanese Fifth Generation programs mostly relies on it as the basic language for future computers. Its use for robot programming has not yet been tried. In some application fields near to robots, i.e. CAD, its use has been demonstrated useful. In a comparison (18) between a 3D graphic program written in Pascal and the same program written in Prolog the Prolog implementation was more concise, readable and clear than the Pascal version. It also took less storage and ran faster than the Pascal compiled version. Since Prolog has been used for operating systems as well as for plan generation and has various ways for managing arrays it applicability to robotics waits only to be fully demonstrated. We are guessing that a prolog implementation will be accepted both by Artificial Intelligence oriented users and by mathematical oriented people.

EUROPEAN ROBOT LANGUAGES

The European scene of robot programming is very active. European was the first commercially available language for robot, SIGLA, and European are some of the most advanced projects. In the following we make a short presentation of all of them we have so far found in the literature.

HELP (8,11). It is the language developed by DEA (Italy) for their Pragma A 3000 (Allegro in USA) assembly robot. It allows concurrent programming and structured programming. The syntax is Pascal-like, all the manipulation functions are provided as subroutines. Signal and wait provide synchronization between different tasks. Any kind of sensors can be connected using a rich set of I/O ports operations. The robot is modular, different arms and different degrees of freedom for each arm can be organized. The coordinate system is cartesian, and two rotary axes can be added to every wrist. The application programs are usually provided with the installation

of the robot. Major applications are in the automotive industry, electronic assembly, precision mechanics. It is implemented on DEC LSI 11 computers under the RT-11 operating system.

LAMA-S (12). The language developed by the Spartacus project, a project aimed at developing robots to help handicapped people in many every day life tasks, as serving drinks or food. LAMA-S used APL as implementation language. The user level functions are translated into a low level language, PRIMA, and then executed. Besides move instructions based on the use of frames LAMA-S provides real time primitives and parallel execution of tasks. The language uses two structures to define the execution order: sequence block, to indicate that all the instructions inside are to be executed sequentially, and parallel block, to indicate that all the instructions inside are initiated in parallel (something as cobegin-coend structure). Other standard control structures are provided. The use of APL demonstrated, according to the authors, that APL is a good implementation tool because it allows functional extensibility of the language. On the other hand they do not recommend it for industrial use because of the following shortcomings: it needs an APL machine to run, the APL syntax is not convenient, the syntax analysis is not perfect, it is difficult to implement interactive programming using APL.

LENNY (30). The language under development at the University of Genova (Italy) to be used to describe movements for an emulated anthropomorphic arm, with seven degrees of freedom. It is intended as a language powerful enough to express complex chains of actions and understandable by humans as a way to represent processes and concurrent computations. One of the key issue of Lenny is functionality. No reference can be done to any absolute kinematic quantity. References are always to actual mechanical context. In Lenny the robot reference frame is fixed in the shoulder and commands like up, down, right, etc. refer to that coordinate system. Functionality will enable Lenny to use any new procedure as part of the language. Lenny got its name from an Asimov novel in which accidentally a robot, named Lenny, became able to learn.

LM: Langage Manipulation (21,23). A language developed at the University of Grenoble (France). It is implemented on a Robitron robot (4 degrees of freedom) cooperating with a Barras robot (2 degrees of freedom), a TH8 of Renault, and a Kremlin robot, both with 6 degrees of freedom, and commercially available on the Scemi robot. It is Pascal-like and frame oriented and provides many of the features of AL but coordination and parallel execution of tasks. It is integrated with LM-Geo (24), a system used to infer bodies positions from geometrical relations. LM-Geo produces program declarations and instructions in LM. LM-Geo resembles RAPT but it doesn't use symbolic algebraic calculus to find the frames which satisfies the equations. It analitycally computes the values.

LMAC (19). A system for flexible manufacturing developed at the University of Besancon

(France). It was designed to assure a safe control of different mechanical devices in the automated cell; for that it performs many checks before actually executing code. It offers modularity based on the implementation of abstract data types, it provides generic modules (the types of data belonging to that kind of module can be specified at run time), and object parametrization. External procedures written in any language can be called by LMAC programs. Different tasks representing different real-time processes can be defined and executed. Synchronization is based on Dijkstra guarded commands. Even though its external form resembles Concurrent Pascal it has been completely rewritten in Pascal.

LPR: Langage de Programmation pour Robots (2). A language developed by Renault and the University of Montpelier (France). It is based on defining state graphs and transition conditions. Transition conditions are used also to synchronize actions. All the graphs at the same level are executed in parallel by the supervisor; every 20 ms an action from each of the same level graphs is executed. Up to 24 input/output ports can be used by LPR to provide sensor interface and synchronization with other devices. LPR runs on a VAX 11/780 and produces code for an Intel 8086 microcomputer controlling the robot. It is available on robots produced by Renault and by ACMA Robotique.

MAL: Multipurpose Assembly Language (14,15). The language developed at the Milan Polytechnic to program a two arm cartesian robot evolved from Olivetti SIGMA. It is a Basic-like system which features synchronization and parallel execution of tasks as well as movement instructions and sensor interfaces. Subroutine calls with argument lists are supported. MAL is composed by two parts, a translator from the input language into intermediate code and an interpreter of the intermediate code. The intermediate code is interfaced with a multimicro hierarchical structure, and all the joints are individually driven by different microcomputers. Force sensing is also controlled by a devoted microcomputer. Photo diodes on the fingers are used as binary sensors. Due to the mechanical architecture of Supersigma collisions between the arms are hardware detected.

PASRO: PAScal for RObots (4). It is provided by the German company Biomatik. It is based on the Pascal language which has been added data types and procedures used to perform robot specific tasks. They are stored in a library and callable by any standard Pascal compiler. It is based on the AL experience. The company may provide assistance in order to modify the coordinate transformations and the control interface for a new kind of robot. Procedures are provided to drive the arm point to point, or along a continuous path. The first implementation of PASRO has been tested on a Microrobot.

Portable AL (13). It is an implementation of the AL programming environment done at Karlsruhe University on mini and micro computers. It incorporates AL compiler (13), POINTY (17)

and a debugging system. A dedicated operating system was developed to support I/O and multi-tasking. It runs on a PDP 11/34 and an LSI - 11/2 which control the PUMA 500 robot.

RAPT: Robot APT (1,25). In its actual implementation RAPT is an APT like language used to describe assemblies in terms of geometric relations and to transform them into VAL programs. A RAPT program consists of a description of the parts involved, the robot, and the workstation and an assembly plan. The assembly plan is a list of geometric relations expressing what geometrical relations should held after a step in the assembly has been done. The program is completely independent of the type of the robot used. Sensors are not integrated movements are not checked against collision avoidance. All the bodies are described as having a position (which is a frame) and some features. Features are plane, cylindrical or spherical faces. A reference system is automatically set in every feature. Against and fits are most used relations. Other relations are used to indicate translation or rotation degrees of freedom left. RAPT builds a graph of those relations and tries to reduce it to the minimum graph using a set of rules. From the reduced graph a VAL program is produced. The Computervision CADD3 system has been used to build the models and to give graphics routines.

ROBEX: ROBoter EXapt (32). The off-line programming system developed at Aachen (Germany) as a programming tool for FMS. Its main purposes are to develop APT for FMS and for robot off-line programming, and to be independent of the kind of robot used. Applications are in workpieces handling. APT style of programming is used to describe geometry, while the ROBEX extensions are robot movement instructions, interactions with sensor (now only binary ones), and synchronization with peripherals (machine tools, conveyor belts...). In this APT like system for FMS three languages will be used: EXAPT, for NC part programming, ROBEX, for part handling programs, and NCMES, for measuring programs. The system is portable in two ways: it is implemented in FORTRAN IV and it generates robot independent pseudo-code which is sent to the appropriate robot for further processing and execution. The user interactively or using a graphic interface inputs coordinates and geometry of the world and programs.

SIGLA: SIGma LAnguage (3,26). The language for programming Olivetti SIGMA robots. Now quite obsolete and under replacement, it has been available since 1975. SIGLA is a complete software system which includes: a supervisor, which interprets a job control language, a teaching module which allows teaching-by-guiding features, an execution module, editing and saving of program and data. SIGLA has been in use for years at the Olivetti plant in Crema (Italy). Its applications span from assembly to riveting, drilling, milling. All the system and the application program run in 4K of memory, and this compactness was necessary at the time SIGMA was delivered because memory was still expensive.

SRL: Structured Robot Language (6). The language under development at the University of

Karlsruhe. It is a successor of Portable AL and owns some to Pascal too. Data types as in Pascal are added to AI data types. The declaration part contains also a specification of the system components. Instructions can be executed sequentially, in parallel, or in a cyclic or delayed way. Different motions are available, in particular straight and circular motions. The project of SRL is part of a standardization project. The source SRL code is translated into an intermediate code, IRDATA, which is a machine independent code.

VML: Virtual Machine Language (16). The language developed in cooperation by the Milan Polytechnic and the CNR Ladseb of Padova (Italy). Intended as an intermediate language between Artificial Intelligence systems and robot it receives points in the cartesian space and transforms them into joint space. It manages task definition and synchronization as well. It is part of a hierarchical architecture, in which 3 levels are today implemented.

CONCLUSIONS

It is hard to not get lost in the many different languages for robots available around. We have tried to review them trying to see what experience they have provided and what open problems they have not solved. Many issues has not been addressed and we didn't expect to be complete. Most of the new commercially available languages for North American market have not been included. We wanted to make our overview on the basis of the experiences available in Europe now. While most of the attention is now focused on acquiring manipulator level systems we have tried to discover what other trends and experiences are available to expand robot programming toward more ambitious tasks.

ACKNOWLEDGEMENTS

Thanks are given to J. Bach, A. Haurat, and J. Lefebre who provided unpublished material.

REFERENCES

1. Ambler, A. P. and Popplestone, R. J., *Inferring the positions of bodies from specified spatial relationships*, Artificial Intelligence 6, pp 157-174 (1975).

2. Bach, J., *LPR Description*, unpublished, Renault, France (1983).

3. Banzano, T. and Buronzo, A., *SIGLA - Olivetti robot programming language*, Proc. Programming Methods and Languages for industrial robots, IRIA, France, pp 117-124 (1979).

4. Biomatik, *PASRO - Pascal for robots*, Biomatik Co., Freiburg, West Germany (1983).

5. Blume, C., *A structured way of implementing the high level programming language*

AL on a Mini and Microcomputer configuration, Proc. 11th ISIR, Tokyo, Japan, pp 663-674 (1981).

6. Blume, C. and Jacob, W., *Design of a Structured Robot Language (SRL)*, Proc. Advanced Software in Robotics, Liege, Belgium (1983).

7. Bonner, S. and Shin, K. G., *A comparative Study of Robot Languages*, IEEE Computer, N. 12, pp 82-96 (1982).

8. Camera, A. and Migliardi, G. F., *Integrating parts inspection and functional control during automatic assembly*, Assembly Automation, 1, 2, pp 78-82 (1981).

9. Descotte, T. and Latombe, J. C., *GARI: a problem-solver that plans how to machine mechanical parts*, Proc. 7th IJCAI, Vancouver, Canada, pp 766-772 (1981).

10. DoD, *Reference Manual for the ADA Programming Language*. Proposed Standard Document, Dept. of Defense, USA (1980).

11. Donato, G. and Camera, A., *A high level programming language for a new multiarm assembly robot*, Proc. 1st Int. Conf. on Automated Assembly, pp 67-76 (1980).

12. Falek, D. and Parent, M., *An evolutive language for an intelligent robot*, The industrial robot, 7, 3, pp 168-171 (1980).

13. Finkel, R. et al, *An overview of AL, a programming system for automation*, Proc. 4th IJCAI, Tbilisi, USSR (1975).

14. Gini, G. et al, *A multi-task system for robot programming*, ACM Sigplan Notices, Vol. 14, N 9 (1979).

15. Gini, G. et al, *MAL: a multi-task system for mechanical assembly*, Proc. Programming Methods and Languages for Industrial Robots, IRIA, France (1979).

16. Gini, G. et al, *Distributed robot programming*, Proc. 10th ISIR, Milan, Italy (1980).

17. Gini, G. and Gini, M., *Interactive development of object handling programs*, Computer Languages, Vol. 7, N. 1 (1982).

18. Gonzalez, J. C., Williams, M. H., Aitchison, D. E., *Evaluation of the effectiveness of Prolog for a CAD application*, IEEE CG&A, N 3, pp 67-75, (1984).

19. Haurat, A. and Thomas, M. C., *LMAC: a language generator system for the command of industrial robots*, Proc. 13th ISIR, Chicago, Illinois, pp 12-69 (1983).

20. Kempf, K., *Artificial Intelligence Applications in Robotics - A Tutorial*, IJCAI 83 Tutorial, Karlsruhe, Germany (1983).

21. Latombe, J. C. and Mazer, E., *LM: a high-level programming language for controlling assembly robots*, Proc. 11th ISIR, Tokyo, Japan, pp 683-690 (1981).

22. Lozano-Perez, T., *Robot programming*, Proc. of the IEEE, 17, 7 (1983).

23. Mazer, E., *Geometric programming of assembly robots*, Proc. Advanced Software in Robotics, Liege, Belgium (1983).

24. Miribel, J. F. and Mazer, E., *Manuel d'utilisation du langage LM*, Research report IMAG, University of Grenoble, France (1982).

25. Popplestone, R. J. et al, *An interpreter for a language for describing assemblies*, Artificial Intelligence, Vol 14, pp 79-107 (1980).

26. Salmon, M., *SIGLA - The Olivetti SIGMA robot Programming Language*, Proc. 8th ISIR, Stuttgart, Germany, pp 358-363 (1978).

27. Schimano, B., *VAL: an industrial robot programming and control system*, Proc. Programming Languages and methods for industrial robots, IRIA, France (1979).

28. Silver, D., *The littler robot system*, MIT Artificial Intelligence Lab. Rep. AIM 273 (1973)

29. Taylor, R. H., Summers, P. D., and Meyer, J. M., *AML: A Manufacturing Language*, The International Journal of Robotics Research, 1, 3, pp 19-41 (1982).

30. Verardo, A. and Zaccaria, R., *Lenny Reference Manual (in Italian)*. Internal Report, University of Genova, Genova, Italy (1982).

31. Volz, R. A., Mudge, T. N., Gal, D. A., *Using ADA as a programming language for robot-based manufacturing cells*, RSD-TR-15-83 Dep. E&C Eng., The University of Michigan, Ann Arbor, Michigan (1983).

32. Weck, M. and Eversheim, E., *ROBEX - An off line programming system for industrial robots*, Proc. 11th ISIR, Tokyo, Japan, pp 655-662 (1981).

33. *CAM-I proposes standards in robot software*, The Industrial Robot, pp 252-253 (1982).