6th Int. Conf. on Control, Automation, Robotics and Vision, ICARCV'2000, 5–8 December 2000, Singapore. (on CD-ROM)

Implementation of Control Systems for Autonomous Robots

Cezary Zieliński*

**School of MPE, Nanyang Technological University,
Nanyang Avenue, Singapore 639798.
MCZielinski@ntu.edu.sg**

Abstract

The paper presents diverse control system implementation techniques for autonomous robots. It takes into account that such control systems require programming means for communicating the tasks that have to be executed by the robots and that those devices must have both considerable sensor data gathering capability and technical means for communicating and cooperating with other devices, presumably robots too.

1. Introduction

The encyclopedic definitions of the term "autonomous" usually underscore that a certain entity is independent of a greater whole or that it works without outside control. Speaking of technical devices we do not intend to make them independent to such an extent that they will be utterly outside our control. Our intention is to make them capable of attaining goals formulated by humans, but in a manner that will not require human intervention. This understanding of autonomy requires that, on the one hand, there will be some means of communicating the task to the device that is to carry it out, and, on the other hand, that the device will have adequate means of perception of the environment to execute the task, although the state of the environment might change to a considerable extent. In other words, autonomous devices require adequate sensing and programming capabilities. This paper discusses possible control system structures for autonomous robots, taking into account several factors:

- method of conveying to the robot the task that is to be executed (i.e. programming method),
- method of implementing the control system,
- incorporation of diverse sensors,
- cooperation with other devices (including other autonomous robots).

The discussion of diverse implementation options that the system designer has at his or her disposal is of utmost importance, as misjudgment at the conceptual stage of constructing a controller usually brings about very tedious and costly alterations later on. The paper systematizes and brings together possible robot programming language implementations taking into account both sensor incorporation and different modes of cooperation between the system modules, and thus points out what are the most general control system structures that can be used. A neat system structure fosters easy implementation and simplifies modifications, if they become necessary. Moreover, reusability of code is a by-product, which results in faster implementation of new generations of systems.

The paper deals with diverse autonomous robots, e.g. manipulators, mobile robots, walking, climbing, swimming and flying machines. All of those machines exert their influence on the environment through effectors, e.g.: arms, legs, wheels. Both single and multi-effector robots will be dealt with here.

2. Robot Programming Methods

There are two extreme approaches to robot programming: on-line (i.e. utilizing the robot for creating a program) and off-line (i.e. writing programs without using the robot). In the former the robot is used to teach in the trajectory positions that have to be subsequently reproduced. In the latter the task that is to be executed is expressed in a textual (and sometimes graphical) form. The first one is easy to master and invariant to imprecision of manufacturing of robots, but it makes the robot unproductive during programming and causes considerable problems to sensor incorporation. In the second, sensor incorporation is relatively simple and the robot is not needed during programming, but it is considerably more difficult to learn and – most important – requires calibration. This is due to finite precision of manufacturing, assembly and location of robots. On-line programming relies on high repeatability of robots. Off-line programming suffers from their low accuracy. In the same robot repeatability can be well below a fraction of a millimeter while its accuracy can be of an order of centimeters. Nevertheless the advantages of off-line programming are so compelling that the industry is now introducing this method of programming into their robot controllers, but retains, to a certain extent, the teach-in capability (e.g. `VAL II` [16], `RobotScript` [5]), thus rendering a hybrid method of programming. The teach-in method

*Currently on long term leave from Warsaw University of Technology, Institute of Control and Computation Engineering, ul. Nowowiejska 15/19, 00-665 Warsaw, Poland, C.Zielinski@ia.pw.edu.pl of programming is retained, because calibration is both expensive and difficult, nevertheless there are industrial examples of successful implementation of shop-floor calibration techniques (e.g. [4]). In the research community the off-line method is favored, as it is more flexible, especially regarding sensor integration. Even if the robots are assembled as imprecisely as they are now, calibration would not be needed, if the robots would be equipped with diverse sensors detecting misalignments. The control strategy would have to be intelligent enough to take into account the sensory information to cope with the variations in the expected state of the environment. This strategy aims at increasing the the degree of autonomy of industrial robots, but it is also valid for any other robots. Not only accuracy can be enhanced by using adequate sensors, but also the decision making capabilities of robots can be greatly extended. Before calibration and on-line programming methods can be given up totally in the industry, several conditions must be fulfilled:

- sensor technology must become cheaper,
- efficient motion control strategies utilizing sensory data must be devised,
- improved methods of specifying tasks must be introduced,
- controller implementation methods taking into account the above three conditions have to be defined.

The enumerated conditions are exactly the same as those for producing autonomous robots. Solving the above mentioned problems will benefit both the industrial robots and non-industrial ones (e.g. mobile robots, walking machines). The industrial robots will be capable of dealing with inexactness in positions and non-industrial ones with unstructured environments.

3. Implementation of Programming in Control Systems

Regardless of the method of programming (on-line or off-line) the task that is to be executed by a robot can be treated as data. In the case of on-line programming this data is a sequence of end-effector locations, and in the case of off-line programming this is either a textual description of the task or, rarely, its graphic representation (e.g. [7]). In both cases the controller of the robot has to interpret this data to execute the task. As this data has a certain syntactic form (although in the case of on-line programming it is very simple) we can treat it as a program expressed in a certain programming language.

Generally, two approaches are followed regarding implementation of robot programming languages (RPLs). Either a specialized RPL is defined or a library of modules is created using a general purpose programming language (GPPL). In the former case we shall be speaking of specialized RPLs and in the latter of submerged RPLs. This section intends to look at the consequences that those two approaches have for possible implementation techniques of robot controllers.

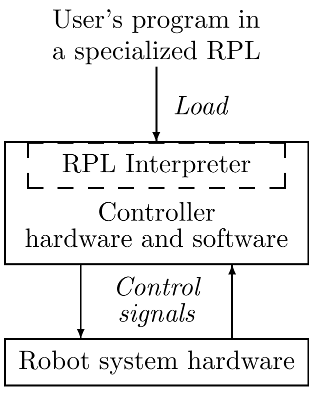

User's program in a specialized RPL
Load
RPL Interpreter
Controller hardware and software
Control signals
Robot system hardware

Fig. 1
Direct interpretation of a specialized RPL user's program

Both specialized RPLs (e.g. `AL` [6], `AML` [8], `VAL II` [16], `SRL` [1], `TORBOL` [9], ROBICON [7]) and submerged RPLs (e.g. `PASRO` [1], `RCCL` [2], `KALI` [3], `RORC` [11], `MRROC` [12], `MRROC++` [14], [15], `RobotScript` [5]) have been implemented in abundance. The latter use diverse GPPL platforms, e.g.: `PASRO` – `Pascal`; `RCCL`, `KALI`, `RORC`, `MRROC` – `C`; `MRROC++` – `C++`; `RobotScript` – `Visual Basic`.

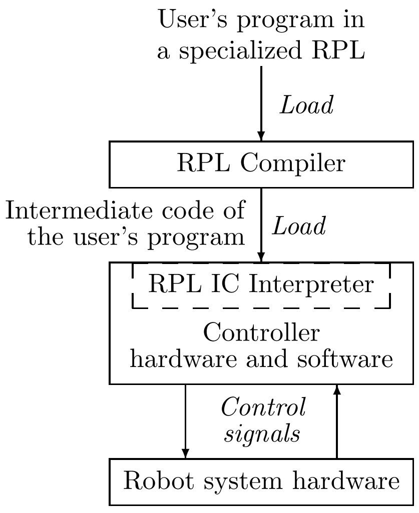

User's program in a specialized RPL
Load
RPL Compiler
Intermediate code of the user's program
Load
RPL IC Interpreter
Controller hardware and software
Control signals
Robot system hardware

Fig. 2
Precompilation followed by interpretation of the intermediate code of a specialized RPL user's program

In the case of specialized RPLs the most obvious solution is presented in fig. 1. The program expressed in a specialized RPL is directly interpreted by the controller, which in turn produces the control signals driving the effector or effectors in the case of a multi-effector system. The drawback of this solution is that in the case of complex RPLs the instruction interpretation process may take considerable time. To remedy this issue specialized RPL can undergo initial compilation to obtain a simpler intermediate code (IC) that in turn is interpreted (fig. 2). The compilation process can be multi-phase utilizing several intermediate RPLs.

Following this line of thought there is a theoretical possibility of compiling an RPL program straight away onto the target controller hardware, which is an embedded computer with adequate input-output hardware (fig. 3). Although this might result in the most efficient code, the effort of creating such a compiler is not worth it. Even a slightest change in the hardware would bring about changes to the compiler.

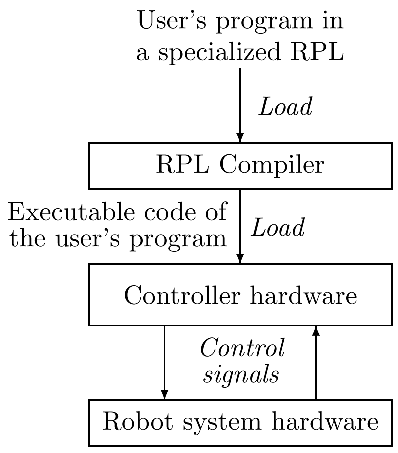
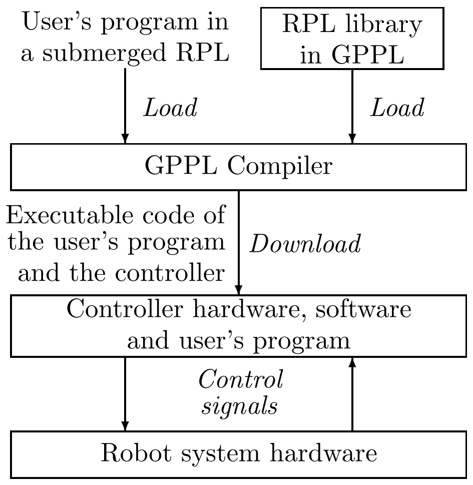


Fig. 3
COMPILATION FOLLOWED BY EXECUTION OF THE EXECUTABLE CODE OF A SPECIALIZED RPL USER'S PROGRAM

The general problem with specialized RPLs is that either the language is tailored to a limited class of tasks, and then any task outside this class cannot be described and in consequence carried out, or the capabilities of the language are very large and so the majority of tasks can be expressed in it, but then, on the one hand, the language and its compiler become very complex and in effect expensive in implementation, and on the other hand, the language becomes difficult to master. Obviously a certain compromise is possible, so this approach is favored by the industry. In industry the robot itself usually performs a very limited class of tasks defined by the end-effectors and sensors that can be incorporated into the system and so the language can be kept within a reasonable complexity. In the academia the hardware configuration of the system can vary considerably and the simplicity of the language is not that much of an issue. In this case, instead of defining a complex RPL, a GPPL is used as a platform and only a library of modules (procedures, functions or objects) is defined which renderes this language an RPL. In this way a standard compiler can be used slashing the development costs and the full potential of GPPLs is at the fingertips of the programmer. Any changes to the hardware configuration or any new control algorithms simply result in the extension of the library. As the library gives extra functionality to the GPPL the new entity is treated as an RPL. Let us look at the implementation possibilities within this paradigm, and see if this excludes the use of specialized RPLs from the university laboratories.



User's program in a submerged RPL
RPL library in GPPL
Load
Load
GPPL Compiler
Executable code of the user's program and the controller
Download
Controller hardware, software and user's program
Control signals
Robot system hardware

Fig. 4
COMPILATION FOLLOWED BY EXECUTION OF THE EXECUTABLE CODE OF A SUBMERGED RPL USER'S PROGRAM AND CONTROLLER

The basic concept of submerged RPL implementation is presented in fig. 4. The user's program, expressed as a sequence of invocations of library modules, is jointly compiled and linked to the library and so an executable code results. This code is directly executed by the embedded computer (or computers) of the controller. What really is created in this case is an amalgamated user's program and the robot control software. The concept introduced in fig. 3 is valid in this case, because a GPPL compiler is used and so no new compiler has to be implemented. Moreover, any addition to the system of new hardware is accommodated through upending code to the library and not through changes to the compiler. In this case programming is really carried out in the GPPL. Obviously the task is expressed by calling adequate modules, and the control software is hidden from the normal user within the library. Only, if new hardware is incorporated into the system, the library must be extended. This solution implicates that even the slightest change to the user's program brings about recompilation of the whole system software.

The remedy to this problem is to input into the working controller parametric data for the user's program. It can be delivered in the form of a specialized RPL, which will be processed by an interpreter contained within the library. The additional language can be extremely simple, e.g. define a syntactic form for the taught-in end-effector locations. To avoid confusion, the RPL derived from the GPPL is termed RPL_1 and the specialized language RPL_2. RPL_1 is used to code the overall structure of the controller and to render it capable of executing a class of tasks. The vastness of the class depends on the RPL_2 interpreter coded in RPL_1 and included in the controller. RPL_2 is used to code the execution of a specific task within the class boundaries. This solution encompasses the one presented in fig. 1. If RPL_2 is subjected to precompilation then this solution also includes the one shown fig. 2.

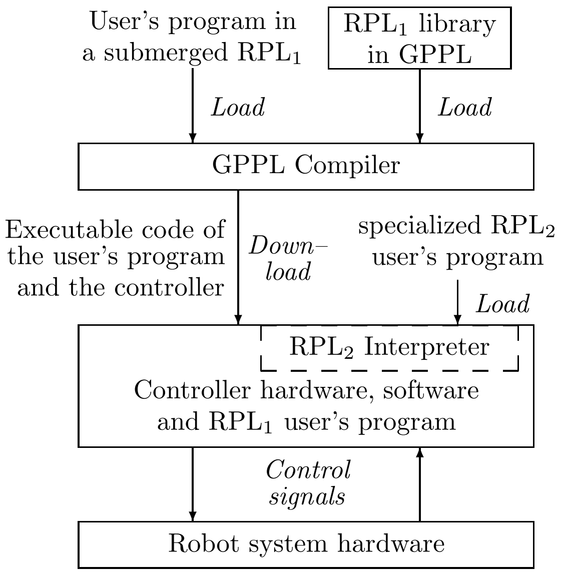

User's program in a submerged RPL1
RPL1 library in GPPL
Load
Load
GPPL Compiler
Executable code of the user's program and the controller
Down–load
specialized RPL2 user's program
Load
RPL2 Interpreter
Controller hardware, software and RPL1 user's program
Control signals
Robot system hardware

Fig. 5
COMPILATION FOLLOWED BY EXECUTION OF THE EXECUTABLE CODE OF A SUBMERGED RPL_1 CONTROLLER AND USER'S PROGRAM WITH A SPECIALIZED RPL_2 INTERPRETER

There still exists another method of merging the specialized and submerged language approaches (fig. 6). In this case a specialized language RPL_3 is developed. It is compiled into an intermediate code in RPL_1, which in turn is subjected to any of the forms of processing presented in fig. 4 or 5. The case shown in fig. 6 is the most general one. However, the use of supplementary RPL_2 in this case is not very probable, but theoretically possible.

It turns out that the structure in fig. 6 is the most general one, as not only does it permit the use of two specialized RPLs, but also suggests a feasible method of implementation of the controller executing programs coded in those RPLs. However, the complexity of this structure might discourage its full implementation, but its substructures can be used then.

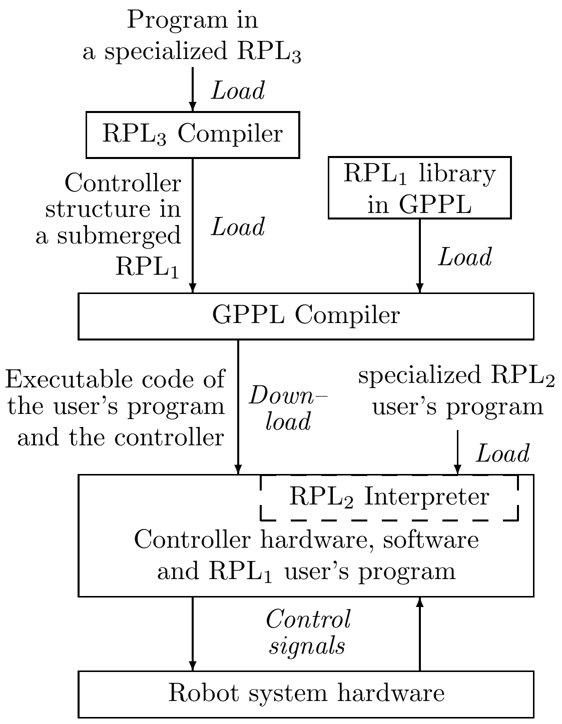

Program in a specialized RPL3
Load
RPL3 Compiler
Controller structure in a submerged RPL1
Load
RPL1 library in GPPL
Load
GPPL Compiler
Executable code of the user's program and the controller
Down–load
specialized RPL2 user's program
Load
RPL2 Interpreter
Controller hardware, software and RPL1 user's program
Control signals
Robot system hardware

Fig. 6
COMPILATION OF RPL_3 PROGRAM RESULTING IN AN RPL_1 PROGRAM DESCRIBING THE CONTROLLER CAPABLE OF INTERPRETING A USER'S PROGRAM EXPRESSED IN RPL_2

4. Structure of the Controller

Once one of the above methods of communicating the task to the controller is selected, the problem of the controller structure has to be tackled. Two problems have to be considered in that respect:

- how the information from sensors will be utilized,
- how will multiple effectors cooperate.

Raw data obtained from hardware sensors usually cannot be used directly to control the system. It has to be transformed. There are cases when only several hardware sensors simultaneously can deliver meaningful data for motion control (e.g. several strain gauges produce a force/torque vector). On the other hand there are complex sensors that deliver data that has to be processed in order to obtain information that can be used in motion control (e.g. CCD camera delivering a bit-map out of which a center of mass of the object must be extracted to locate the best grasping position). Sensor reading transformation is called data aggregation. As a result a virtual sensor reading is obtained. This necessitates the inclusion in the controller of a module performing this function. Modern control software usually utilizes real-time operating systems. In such a case it is reasonable to assign the sensor data aggregation to a process. This will be called a Virtual Sensor Process (VSP).

Three forms of effector cooperation within a single multi-effector robot are distinguished:

- independent operation (effectors are not coordinated),
- loose cooperation (effectors are synchronized from time to time),
- tight cooperation (effectors are coupled).

The same applies to cooperation between robots in a group, so the same control structure can be used both to coordinate effectors within a single robot or the robots among each other. To make the further discussion brief we shall not distinguish between robots and effectors, and the latter term will be used for both (unless ambiguity would result). It is assumed that there are n_e effectors in the system and that each effector uses n_{v_i} virtual sensors, where $i = 1, \ldots, n_e$.

The controller structure for the effectors (or robots) operating independently of each other is presented in fig. 7. It should be pointed out that independent operation means that there is no explicit form of communication between the effectors, i.e. the Effector Control Processes (ECPs) do not exchange data explicitly. Obviously implicit communication is possible. This can be attained by perceiving the actions of the other effectors through the effector's own sensors. In this structure each ECP would be responsible for interpretation of its own RPL (presumably each RPL would be the same, but the programs expressed in this RPL might be different).

Both forms of cooperation, loose and tight, require a coordinator. In the case of the structure shown in fig. 8 it is called the Master Processes (MP). For loose cooperation MP synchronizes the ECPs from time to time, so between those instants each ECP operates on its own. However during tight cooperation the MP drives the ECPs continuously. It can be easily noticed that if MP is kept constantly dormant, the structure off fig. 8 degenerates to the one off fig. 7. That structure has been used in all investigations, because it is generic and includes as a specific case the uncoordinated one. If interpretation of a specialized language (i.e. RPL$_2$) is necessary in the coordinated case then the MP is responsible for it. In the uncoordinated case the MP simply distributes the RPL$_2$ programs to the respective ECPs for local interpretation. If no specialized language is employed the user's program is dispersed between the ECPs and the MP – it is amalgamated with the library software.

5. Examples

The majority of the above described structures has been used for the implementation of various RPLs. For instance, from the point of view of the user, `TORBOL` [9], being a specialized language, used the structure presented in fig. 2. In reality the control system implementation used a library of procedures, but the existence of this library was not disclosed to the user. The intermediate code and the library were expressed in `Pascal`, so the implementation structure resembled the one shown in fig. 6, but RPL$_2$ was not present. `TORBOL` is an object-level RPL, i.e. the task to be executed is expressed in terms of models of real objects (possessing certain attributes) and relations between them (e.g. IN, ON). The run-time system maintained a world model (an attribute graph), so the positions of all the objects could be tracked down. Even if an object was displaced due to being located on top of or affixed to another object that was being moved its position was updated adequately. The language was intended to simplify the description of complex pick-and-place or assembly tasks.

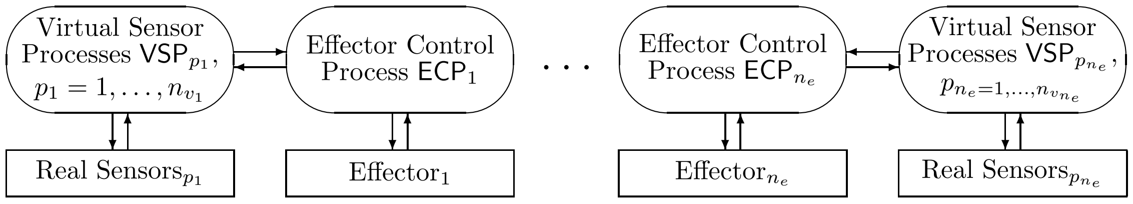
`RORC` [11] was implemented using the structure presented in fig. 4. It could control only a single robot. Diverse controllers were produced subsequently. One of them was used for the investigation of reactive robot control [10], [11]. A manipulator carrying a touch probe had to find a way out of a maze by using only the local knowledge of the environment obtained through sensors.

`MRROC` used a similar implementation technique to `RORC`, but could control multiple robots and cooperating devices. One of the so produced controllers was used for the investigation of cooperative transfer of rigid objects by two robots [13].

`MRROC++` initially used the same structure as `MRROC` (fig. 4), but instead of following a procedural implementation approach the object-oriented one was used. Later the structure presented in fig. 5 was adopted. It controlled a robot equipped with a milling machine. The specialized RPL$_2$ was used to describe tool positions and orientations generated by a Unigraphics CAD system. The trajectories produced by the CAD system had to be pre-processed off-line to create a program in RPL$_2$, which subsequently was loaded into the controller.

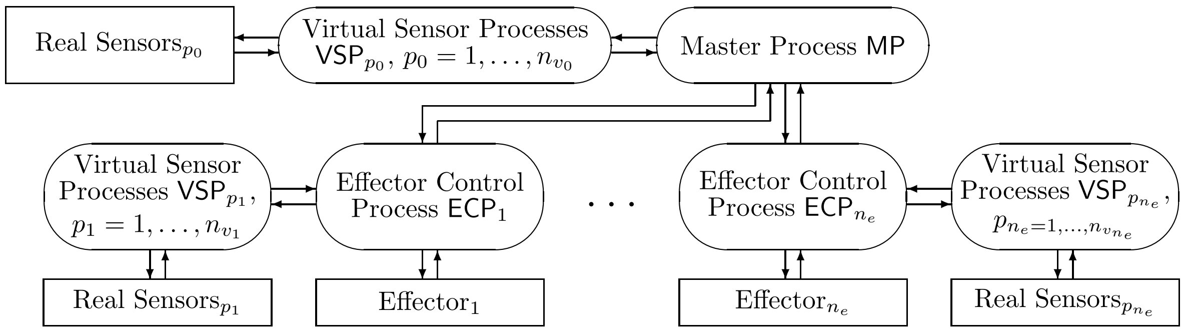
All of the above examples pertain to single or multi-manipulator systems, but the developed structures are not limited only to such systems. For example, a walking machine is a multi-effector system, in which each leg would be controlled by its own ECP and the legs would be coordinated by the MP. The tasks for the machine might be communicated using a specialized RPL$_2$, which would be interpreted by the MP, so the structures of fig. 5 would be appropriate for the implementation of such a system.

Robot soccer has become popular lately. In this case a team of mobile robots, each using local sensors (to detect the ball and other players in their vicinity) and a global sensor (the overhead camera enabling the assessment of the overall situation), cooperates in pushing a golf ball into opposing team's goal. Each mobile robot would be controlled by its own ECP$_i$, $i = 1, \ldots, n_e$, where n_e is the number of teammates. Readings from local sensors would be processed by VSP$_{p_i}$, $p_i = 1, \ldots, n_{v_i}$, where $, n_{v_i}$ is the number of virtual sensors used by the robot



Virtual Sensor Processes VSP_{p_1}, $p_1 = 1, \ldots, n_{v_1}$
Effector Control Process ECP_1
. . .
Effector Control Process ECP_{n_e}
Virtual Sensor Processes $\mathsf{VSP}_{p_{n_e}}$, $p_{n_e=1,\ldots,n_{v_{n_e}}}$
Real Sensors$_{p_1}$
Effector$_1$
Effector$_{n_e}$
Real Sensors$_{p_{n_e}}$

Fig. 7
UNCOORDINATED OPERATION OF EFFECTORS



Real Sensors$_{p_0}$
Virtual Sensor Processes VSP_{p_0}, $p_0 = 1, \ldots, n_{v_0}$
Master Process MP
Virtual Sensor Processes VSP_{p_1}, $p_1 = 1, \ldots, n_{v_1}$
Effector Control Process ECP_1
. . .
Effector Control Process ECP_{n_e}
Virtual Sensor Processes $\mathsf{VSP}_{p_{n_e}}$, $p_{n_e=1,\ldots,n_{v_{n_e}}}$
Real Sensors$_{p_1}$
Effector$_1$
Effector$_{n_e}$
Real Sensors$_{p_{n_e}}$

Fig. 8
COORDINATED OPERATION OF EFFECTORS

i. The global sensor readings would be processed by VSP_{p_0}, and would be delivered to MP. The MP would communicate with the ECPs through a radio link. The structure off fig. 8 would be most appropriate for this setup. If an operator has to deliver any instructions to the system, that would be done by using a certain RPL interpreted by the MP, so in that respect the implementation structure off fig. 5 would be most appropriate.

6. Conclusions

The presented implementation structures have been tested on single and multi-robot systems. The degree of autonomy of the produced systems depended on their: data gathering capability (number and type of sensors used), flexibility of utilizing thus obtained data for the purpose of motion control and decision making process. The submerged RPLs do not limit the possibility of introducing an extra specialized language enabling the description of only a limited class of tasks, thus simplifying the task of programming such a system.

References

[1] Blume C., Jakob W.: *Programming Languages for Industrial Robots*. Springer-Verlag, 1986.

[2] Hayward V., Paul R. P.: *Robot Manipulator Control Under Unix RCCL: A Robot Control C Library*. Int. J. Robotics Research, Vol.5, No.4, Winter 1986. pp.94-111.

[3] Hayward V., Hayati S.: *KALI: An Environment for the Programming and Control of Cooperative Manipulators*. Proc. American Control Conf., 1988. pp.473-478.

[4] Hollingum J.: *Preprogramming complex welds fast*. Industrial Robot, Vol.22, No.3, 1995, pp.35–36.

[5] Lapham J.: *RobotScript: The introduction of a universal robot programming language*. Industrial Robot, Vol.26, No.1, 1999, pp.17–27.

[6] Mujtaba S., Goldman R.: *AL Users' Manual*. Stanford Artificial Intelligence Laboratory, 1979.

[7] Schroeder C., Zuehlke D.: *Integration of Sensor Technology in Visual Robot Programming Systems*. Proc. 30th Int. Symp. on Robotics, Tokyo, October 27-29, 1999. pp.297–304.

[8] Taylor R. H., Summers P. D., Meyer J. M.: *AML: A Manufacturing Language*. The International Journal of Robotics Research, Vol. 1, No. 3, 1982. pp.842–856.

[9] Zieliński C.: *TORBOL: An Object Level Robot Programming Language*. Mechatronics, Vol.1, No.4, 1991. pp.469-485.

[10] Zieliński C.: *Reaction Based Robot Control*. Mechatronics, Vol.4, no.8, 1994. pp.843–860.

[11] Zieliński C.: *Robot Programming Methods*. Publishing House of Warsaw University of Technology, 1995.

[12] Zieliński C.: *Control of a Multi-Robot System*. 2nd Int. Symp. Methods and Models in Automation and Robotics MMAR'95, 30 Aug.–2 Sept. 1995, Międzyzdroje, Poland. pp.603-608.

[13] Zieliński C., Szynkiewicz W.: *Control of Two 5 d.o.f. Robots Manipulating a Rigid Object*, Proc. IEEE Int. Symp. on Industrial Electronics ISIE'96, 17–20 June 1996, Warsaw, Poland. Vol.2, pp.979–984.

[14] Zieliński C.: *Object–Oriented Programming of Multi–Robot Systems*, Proc. 4th Int. Symp. Methods and Models in Automation and Robotics MMAR'97, 26–29 August 1997, Międzyzdroje, Poland, pp.1121–1126.

[15] Zieliński C.: *The MRROC++ System*, 1st Workshop on Robot Motion and Control, RoMoCo'99, 28–29 June, 1999, Kiekrz, Poland. pp.147–152.

[16] —: *User's Guide to VAL II: Programming Manual*. Ver.2.0, Unimation Incorporated, A Westinghouse Company, August 1986.