M. Vukobratović, B. Karan, N. Kirčanski, D. Stokić

A Software System for Teaching and Commanding The Industrial Robots

In this paper the design features of a software system for programming industrial robots with dynamic control are described. The software is intended for implementation an a microprocessor-based system and should enable the user to control a robot via specialized programming language RL. The paper contains description of the RL language and description of the system structure. Main characteristics and advantages of the system as well as questions concerning its realization on existing microprocessors are also discussed.

Programski sistem za učenje i upravljane industrijskih robota. U radu su opisane projektne karakteristike programskog sistema za programiranje industrijskih robota sa dinamičkim upravljanjem. Programska podrška predvidena je za implementaciju na sistemu zasnovanom na mikroprocesorskoj tehnologiji i treba da omogući korisniku upravljanje robotom preko specijalizovanog programskog jezika RL. Rad sadrži opis jezika RL i opis strukture sistema. Takode su razmotrene glavne karakteristike i prednosti sistema kao i pitanja vezana za njegovu implementaciju na postojećim mikroprocesorima.

Key-words: industrial robots, robot controllers, robot programming languages.
Ključne reči: industrijski roboti, robotski kontroleri, robotski programski jezici.

1. INTRODUCTION

A number of systems for programming the industrial robots is developed in the last decade. Their capability of programming robots in external coordinates and incorporated advantages of general purpose programming languages were important factors for the rapid increase of robotized industry sites.

Most of the existing systems are intended for use with a particular robot; for example, the most popular system VAL [1] can be used only for control of PUMA family robots, the AML [2] for IBM System/1 robots etc. Recently, some systems that can be applied to various types of robots were developed and already announced on the market. However, their adjustment to a particular type of robot is still too complex and tedious job to be efficiently accomplished by a customer. Besides, they usually do not perform compensation of dynamic effects, so that good tracking of fast trajctories cannot be achieved. For these, reasons, the development of a new general purpose controller UCS-1 was commenced in the Mihailo Pupin Institute, Belgrade [3]. This paper describes the main design features of the software support of the controller.

2. DESIGN OBJECTIVES

The system is designed to meet two main advantages over existing controllers: dynamic control ensuring tracking of fast trajectories and easy maintenance and adjustment the system to nonredundant robots of arbitrary type with up to six degrees of freedom.

The specific goals having in mind during the design of the software were:

— reduction of the run time computation required for calculation of quantities related to the robot dynamics; in order to meet this goal, which has the principal importance in implementation of the system on existing microcomputers, the use of analitical models is adopted;

— hardware transportability, i. e. a possibility of adapting the system by the costumer to a particular robot and a particular application without the need of intensive training the user; for this purpose, an interactive procedure for imposing the parameters necessary for the automatic creation of an analitical model and the selection of control algorithm is developed;

— possibility of adapting the robot operation to the robot environment, espetially of sinchronizing the robot with the external hardware; this condition is essential for the applicability of the robot in most factory sites; in order to achieve this goal, a set of routines for processing input and output signals and for controlling the order of operation are designed;

— possibility of implementing the system on a unexpensive computer system and possibility of ope-

rating the system without utilizing mass memories, so that the probability of the system faults in factory conditions is decreased;

— reduction of the human effort necessary for programming the robot task; to this end, a specialized programming language RL is designed; the language supports programming the robot in external coordinates, enables to utilize variables of various types, control structures and user-written subroutines.

— robustness of the system, i. e. protection of the system integrity from an unautorized use and from accidental programming errors.

In the following section, a short descrition of the language RL is presented.

3. THE ROBOT PROGRAMMINING LANGUAGE OVERVIEW

One of main objectives stated in design of many robot programming languages was simplicity of use, even for novice programmers unfamiliar with computers and basic programming concepts. We adopted somewhere different approach: our specific goal was to anable the field engineer, having some expirience in programming, to exploit as much as possible from available robot without necessity to learn a new programming language. For this reason we decided to use a PASCAL-like syntax, supposing that the PASCAL is well-known to a majority of system analysts, control and mechanical engineers which are to be the main designers of automated manufactury sites and the main class of users of programming tools in industrial robotics. Besides, the top-down approach in writing programs that is encouraged in PASCAL and clear structure of PASCAL programs were also reasons for selecting it as a basis for the RL language.

During the specification of the RL language, we started from a view that the language should facilitate all main phases of robot programming and exploitation; among others are:

— definition of the robot task;

— writing a robot program which should describe not only the robot hebaviour during performing the task, but which also should describe procedures for robot teaching, testing, tuning etc.;

— teaching the robot, i. e. memorizing indiviual positions and orientations of the robot effector during performing the task, as well as computation of positions, orientations and dimensions of working objects;

— testing the robot program, including eventual reteaching the positions;

— adjusting the controller parameters in order to meet particular requirements connected to the manipulator mechanical structure and the task to be performed;

— exploitation, which can also include necessity for an occasional reteaching particular positions and orientations and adjusting the controller parameters.

The definition of the robot task can be done either in an ad-hoc manner, either using the robot language itself; we decided to encourage the second one, believing that it can speed up the process of writing robot programs and that it can result in more readable programs and therefore programs that are easier to debug, test, tune and expand.

The program entities corresponding to particular phases stated, espetially to the phases of teaching and performing the task, can be viewed as separate program tasks surrounding a common data base. The data base should contain all data that are to be premanently memorized (end effector trajectory, position and orientation of working objects etc.). Such data base is implicitelly included in all systems for programming the robots; however, we were standing in position that the programming errors checking and the robot programs maintenance can be done much easier if the base were declared explicitelly by the programer and if all the tasks were decsribed in one programming unit which we call package. The general structure of the robot program package is:

```
PACKAGE name;
    BASE declaration-of-the-data-base;
    description-of-the-initialization-part;
TASK name-1;
    description-of-the-task-name-1;
TASK name-2;
    description-of-the-task-name-2;
*
*
*
TASK name-n;
    description-of-the-task-name-n.
```

As it can be seen, the package consists of the data base declaration, description of the initalization part and description of zero or more tasks. Every task can be considered as a user-written extension of system-supplied routines provided for assignment of values to program variables, communication between the user and the system, robot motion, gripper operation, synchronization with external equipment and setting the system parameters.

The language supports Boolean, integer, real, character string, vector and body date types, as well as arrays. Boolean data types can have values from the set {TRUE, FALSE}. The range of values for integer and real data types is implementation--dependent; on the microprocessor Intel 8086 integers can be in the range -32768 to $+32767$, while the real data types can have absolute values in the range $1.0 \, E - 70$ to $1.0 \, E + 70$ including zero, with approximatelly seven significant decimal digits. The character string is a sequence of zero or more (up to 255) printable ASCII characters The vector is defined as an ordered string of three real components (x, y, z) representing its coordinates in the referent

coordinate frame, while the body is defined as an ordered string $(x, y, z, \varphi, \vartheta, \psi)_j$ of six real components representing Euler coordinates of the frame connected to the body with respect to the referent frame.

Data supported by RL can be either constants or variables. Variables are to be referenced via identifiers which can be constructed from an arbitrary string of alphanumeric characters including the underscore character »—« and starting with a letter Like in FASCAL, all identifiers used as variables or named constants must be explicitely declared. For example;

```
CONST  MESSAGE = »SUCCESS«;
       APP = (0.0, 0.0, 20.0, 0.0, 0.0, 0.0);
VAR    TRANSL : VECTOR;
       FAULT : BOOLEAN;
       P, PICK, PLACE, PIN, BLOCK:
       BODY;
       HOLE: ARRAY [1 ... 8] OF BODY;
```

Assignment of a value to a variable is to be achieved using the operator of assignment » : = «. Assignments can be also performed by the operater oy the system via command ACCEPT:

ACCEPT variable

A special command is provided for assignment of values to body-type variables:

LEARN object

Values corresponding to the manipulator tip position and orientation after an intervention of the operater are to be assigned to components of the body-type variable object as the effect of the execution of the LEARN command.

Individual components of vectors and bodies can be referenced using keywords X, Y, Z, PHI, $THETA$, PSI and POS. POS is defined as an ordered string (x, y, z) representing the position of the origin of the frame connected to the body with respect to the referent frame. For example:

```
(* set x-component of TRANSL to 20 mm *)
TRANSL. X : = 20.0;
(* translate BLOCK by TRANSL *)
BLOCK. POS : = BLOCK. POS + TRANSL;
```

A set of built-in library procedures is provided for computation of Boolean, integer and real expressions. The language incorporates Boolean (NOT, AND, OR), relational $(=, <, >, < =, > =, <>)$ and arithmetic operators $(+, -, *, /)$ for scalar operations. Operations with vectors can be achieved using vector addition and subctraction operators $(+, -)$ and the operator for multiplication of vector by scalar $(*)$.

Relationships between bodies can be expressed using the operator »*« and the built-in function INV. The operator »*« moves the referent coordinate frame for the second argument to the position and the orientation represented by the first argument; the function INV returns the position and the orien-

tation of the referent frame with respect to the argument of INV. For an example, if P represents the position and the orientation of the robot gripper grasping the pin, and PICK represents the position and the orientation of the pin bottom, the grasping position and orientation with respect to the pin bottom can be computed as:

PIN := INV (PICK) * P

As in other programming langauges, operator precedence can be overriden using parentheses.

Values of variables, constants and expressions can be displayed on the user terminal by using the command:

DISPLAY expr -1, expr -2, ..., expr $-n$

The most important statement in the language is the statement for specifying the robot motion. The motion statement of the form:

MOVE object TO body-expression

cause the coordinated motion of robot joints until the object connected to the robot effector is aligned with the position and the orientation represented by the body-expression. The object specification can be omitted, in which case the execution of the MOVE statement results in alignement of the robot effector with the body-expression. For example, if the EFFECTOR were the body-type variable representing the robot effector position and orientation with respect to the referent frame, the execution of:

MOVE PIN TO PLACE * BLOCK * HOLE[I]

results in a coordinated motion until the following is satisfied:

EFFECTOR = PLACE * BLOCK * HOLE[I] * * INV(PIN)

Synchronization with the external equipment such as conveyors, feeders, sensory devices etc., is to be achieved using 32 input and 32 output channels, via attached variables. A variable is attached to a channel using the ATTACH statement of the form:

ATTACH identifier: type TO channel number

where the identifier represents the scalar variable to be attached, type can be Boolean or integer, channel represents the channel type (INPUT or OUTPUT) and the number is a positive integer in the range 1 to 32. Such explicit attachment is included in the language in order to provide the user a flexibility of using input and output signals with different meanings. On the other side, it provides the language translator with additional infoimations necessary for error-checking (for example, assignment of values to variables attached to input channels is disabled).

Variables attached to input channels behave as read-only variables and they can be referenced in any place of the program where the occurence of the value of the same type is allowed. Output signals can be generated as a result of specifying assigning values to the variables attached to output channels. Also, explicit waiting for an external event is possible using the WAIT command:

WAII wait-clause

where the wait-clause is any logical expression involving at least one variable attached to an input channel. An example of using attached variables follows:

```
ATTACH LEVEL: INTEGER TO INPUT 2;
    DONE: BOOLEAN TO OUTPUT 10;
*
*
*
WAIT LEVEL < 100;
*
*
*
DONE := TRUE;
```

The specification of the motion command can include a number of control parameters and can also include a testing of variables attached to input channels. The general form of the MOVE command is:

```
NOVE object TO body-expression
    VIA expr — 1, expr — 2, ..., expr — n
    DEPART depart-expr
    APPRO appro — expr
    WITH control — 1, control — 2, ..., control — m
    UNITIL unitil-clause
```

In this form the expressions expr — 1, expr — 2, ..., expr — n return positions and orientations of intermediate points that should be passed by the object without stopping the robot. Expressions depart-expr and appro-expr define the object relative positions and orientations during the motion with respect to its starting and ending positions and orientations where the velocity of the robot tip should stop increasing to or start decreasing from its maximum level. If STARTING were the body--type variable representing the starting position and orientation of the robot tip, the following relations will hold during the execution of the motion command:

```
EFFECTOR = STARTING
EFFECTOR = STARTING * depart-expr *
    * INV (object)
```

```
EFFECTOR = expr — 1 * INV (object)
EFFECTOR = expr — 2 * INV (object)
*
*
*
EFFECTOR = expr — n * INV (object)
EFFECTOR = body — expression * appro —
    — expr * INV (object)
EFFECTOR = body — expression * INV (object)
```

Optional clause WITH enables the user to control the mode of movement. Controls control — 1, control — 2, ..., control — n have the form:

keyword = value

where the keyword specifies a control parameter (SPEED, MAXTIME, PASSMODE, TOLERAN-

CE, EXECMODE etc.) and the value is a keyword, an integer or a floating point value of the parameter. For example:

MOVE PIN TO PLACE WITH PASSMODE =
= Joint, MAXTIME = 3.0;

specifies that the motion is to be performed with linear change of robot point coordinates (default mode, other alternatives are STRIGHT, with linear change of Euler coordinates, and PARABOLIC, with polynomial interpolation) and in time less than or equal 3 seconds.

Optional clause UNTIL specifies a logical expression until-clause involving variables attached to the input channels; the expression is evaluated during the motion and the motion is to be stopped when the value of the until-clause becomes true.

Another way to specify the movement is by specifying displacements in the robot joint coordinates:

```
DRIVE   number — 1 BY displacement — 1
        number — 2 BY displacement — 2
        .
        .
        .
        number — n BY displacement — n
        UNTIL unitl — clause
```

where the integers number — 1, number — 2, ..., ..., number — n are the indices of the robot joints, and the real expressions displacement — 1, displacemnet — 2, ..., displacement — n corresponding displacements; Optional UNTIL clause is the same as in the MOVE command.

Two commands are provided for the gripper opperation:

```
OPEN UNTIL until — clause
CLOSE UNTIL until — clause
```

the optional suffix UNTIL until — clause includes the expression until-clause which has the same meaning as in MOVE and DRIVE commands. The operation of other effectors (as in painting, etc.) can be requested by assigning values to variables attached to corresponding output channels.

Control parameters common to a sequence of MOVE commands can also be set via command:

SET control — 1, control — 2 ..., control — n

where list of controls has the same meaning as in the MOVE command.

The order of execution of commands (i. e. assignment, wait, input/output, motion specification, gripper operation and parameter setting) can be controlled by usual BEGIN ... END, IF ... THEN ... ... ELSE, WHILE ... DO and REPEAT ... UNTIL constructs.

Commands can be groupped in procedure or function subroutines which can be freely used as user--written commands or parts of expressions; their use is restricted to the packages in which they are declared and it is disabled to request execution of subroutines directly from the user terminal. The

scope of variables and named constants are subroutines or tasks in which they are declared. Variables declared in the initialization part of the package, base variables and variables attached to channels are global to the package. During execution of the package it is disabled to change the values of variables by directly entering the RL commands from the user terminal. The only exeption are the variables explicitely declared by the programmer as public; for example:

BASE PICK: PUBLIC BODY;

4. CONTROL SYNTHESIS

The execution of motion commands involves computation of the robot trajectory in the space of joint coordinates (kinematic model), computation of quantities necessary for compensation of the robot dynamics (dynamic model) and computation of output signals for the actuators in the robot joints (control synthesis). In order to reduce the number of floating point operations in the models, an expert program is developed by Vukobratovic and Kircanski [4] for the automatic generation of the models in analytical form. The operation of the program is controlled by the data on mechanism parameters, actuators and tolerances imposed by the user.

The computation of the dynamic model for some types of manipulator structures may be time consuming even with the use of analytical models. For this reason the effects of selecting particular control laws were carefully studied and the following control structure was adopted [3]:

— the local control is synthesized for each robot joint, using the models of particular actuators and neglecting the coupling among the joints;

— the global control necessary for satisfactory tracking of fast trajectories is synthesized as a function of the driving torque.

The driving torques are nonlinear functions of angles, velocities and accelerations of all joints of the manipulator and the computation of the torques using the complete dynamic model is vary complex. However, it has been shown that it is not necessary to use the complete model: as an example, according to investigations by Vukobratovic and Stokic [5], Coriolis and centrifugal forces can be neglected in most cases without loosing the robot performance. Thus, some dynamic effects can be neglected and the computation therefore reduced.

An example of the computational complexity is shown in Figure 1. The figure displays the number of floating point operations that is to be performed for the dynamic control synthesis with various approximative dynamic analytical models and for the following manipulator structures:

CL   — cylindrical, 3 d. o. f. (RTT)
AR   — arthropoid, 3 d. o. f. (RRR)
AN   — anthropomorphic, 3 d. o. f. (RRR)

CL-AN   cylindrical-anthropomorphic, 6 d. o. f. (RTTRRR)
SAR-AN — semiarthropoid-anthropomorphic, 6 d. o. f. (RRTRRR)

The selection of a particular model as well as the selection of period of sampling input data on point angles and velocities is included in the previously mentioned procedure for generation models for controlling the robot motion. In this manner, a satisfactory adjusting the system to the particular application can be achieved.
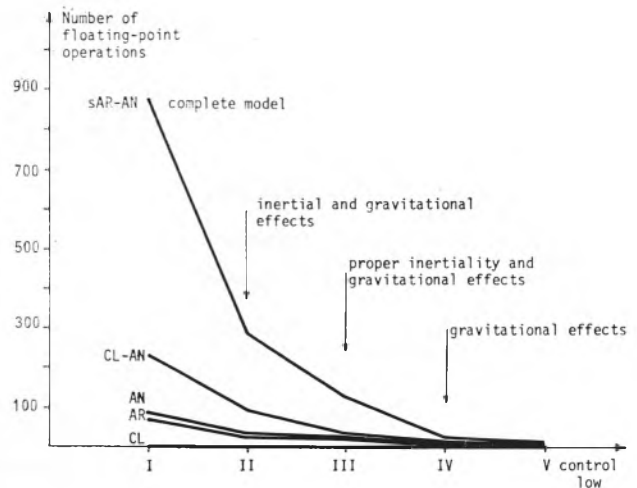

Figure 1. Number of floating point operations in decentralized control structures

5. SOFTWARE ORGANIZATION

The software is designed as a multiprocessing system consisting of a set of interconnected modules (Figure 2.). The central part of the system is denoted as monitor and its main function is the allocation of processor time to individual proccesses such as accepting the user commands and interpreting RL packages; the operation of the monitor is controlled by the signals generated from previously activated processes.

The user of the system can select one of the system programs provided for initial system generation, creating or editing RL packages, cassette drive operation and execution of RL commands via terminal keyboard.

The program for initial generation of the system produces analytical models of the robot kinematics and dynamics, produces models of the robot actuators and calculates the digital servosystem parameters. The program generates the models in machine--readable form and is therefore implementation dependent. It is designed as an interactive program which enables the user to impose mechanical parameters of the robot and actuator parameters and to specify tolerances serving as a basis for producing a code for computation of the digital servosystem
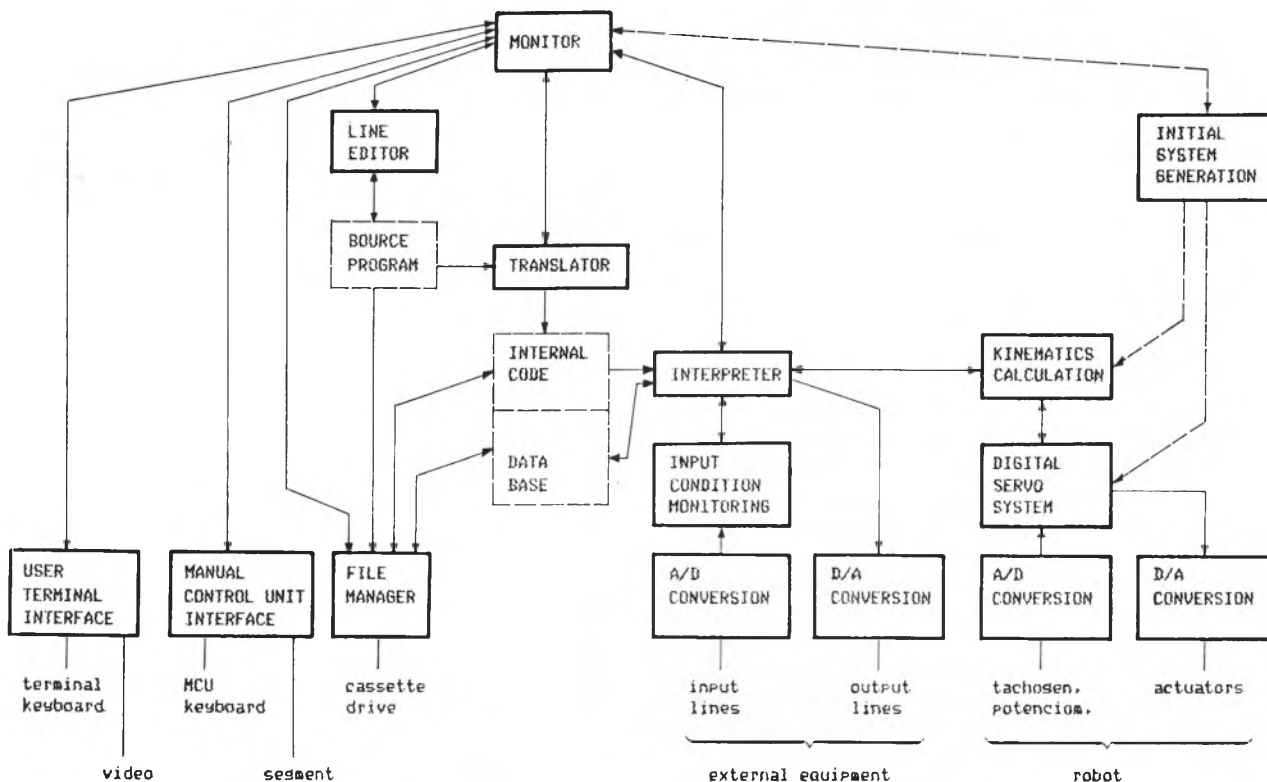
Figure 2. Organization of the software

gains. The parameters and the models can be easily changed and adjusted to particular applications during the exploitation of the robot; however, the program for initial system generation is not necessary during the normal operation of the system.

The creation and editing of RL packages is supported by the specialized line editor which also performs partial syntax error checking during the editing. After completition of editing, the control is automatically transfered to the RL translator. It translates the source RL program into Polish form, produces the symbol table (comprising identifiers denoting tasks entry points, procedures, functions, variables and named constants), and allocates a space for global variables.

The activation of a previously translated package can be requested by the user by simply entering the name of the package; after the initialization part of the package is executed, particular tasks can be activated. The execution of any task can be always stopped by the user and later continued. Also, the execution of any acceptable RL command can be requested from the terminal keyboard; only RL commands not involving identifiers are acceptable before the initialization of the package is done. An alternative way for controlling the system operation is via portable manual control unit: commands from the manual control unit are to be imposed via its functional keys and are a subset of RL commands.

There is no need to use a mass memory during the normal operation of the system. The module denoted as file manager is included in the system in order to enable the user to save and later reload previously written RL packages. The file manager supports formatting cassettes, saving, loading and deleting packages as well as displaying a list of packages saved on a cassette.

The operation of the RL interpreter is controlled by the internal code generated by the translator. The interpreter performs calculation of RL expressions, allocation of core memory to local RL variables, assigns values to the variables and prepares data necessary for realization of the robot motion. The preparation comprises setting the motion control parameters and (for the MOVE command) calculation of Euler coordinates for the robot tip position and orientation with respect to its base that are to be reached in the next execution step. This preparation also includes calculation of constant parts of expressions used in the UNTIL clause of MOVE, DRIVE, OPEN and CLOSE commands (i. e. computation of valuses that do not depend on input signals from external hardware).

The operation of the kinematic module and the module for monitoring signals from the input channeles are under control of the interpreter. The kinematic module realizes on line trajectory synthesis in the space of the robot joint coordinates until

150

the desired point is reached or until the end-of-
-motion condition tested by the module for moni-
toring external signals becomes true. The operation
of the digital servo system is explained in the pre-
ceding section: this module generates signals for the
robot actuator amplifiers on the basis of desired
joint coordinates computed by the kinematic module,
attained angles/positions of the robot joints supplied
from potenciometers, and current velocities in the
joints supplied from tachogenerators.

6. CONCLUSION

The main problem in the design of the system was
to ensure on line computation of trajectories in the
robot joint coordinates and dynamic digital control,
as well as to ensure efficient programming without a
necessity of use an expensive and complex processors.
The solution is quite acceptable from the stand-
point of the equipment complexity and price. It
enables to achieve good performance and to make
the system quite independent from the robot
structure.

REFERENCES

[1] Unimation Inc., **User's Guide to VAL.** Version 12, Danbury, Unimation Inc., 1980.
[2] Taylor R. H., Summers P. D., Meyer J. M., **AML: A Manufacturing Language.** Robotic Ressearch, Vol. 1, No 3, pp. 19—41, 1982.
[3] Vukobratovic M., Kircanski N., Stokic D., Kircanski M., Karan B., **General Purpose Controller for Industrial Manipulators.** Proc. of the Second Yugoslav-Soviet Symposium on Applied Robotics, pp 1—15, Belgrade, Mihailo Pupin Institute, 1984.
[4] Vukobratovic M., Kircanski N., **Computer Assisted Generation of Robot Dynamie Models in Analytical Form,** Acta Applicandae Mathematicae, No 2, pp. 49—70, 1985.
[5] Vukobratovic M., Stokic D., **Control of Manipulation Robots** Berlin, Springer-Verlag, 1982.

AUTHOR'S ADDRESS:

**M. Vukobratović, B. Karan, N. Kirčanski, D. Stokić
Mihailo Pupin Institute
Robotics Department
Volgina 15
11000 Belgrade
Yugoslavia**