A SOFTWARE SYSTEM FOR TEACHING AND COMMANDING THE INDUSTRIAL ROBOTS

B. Karan

Robotics Department, Mihailo Pupin Institute, Belgrade, Volgina 15, Yugoslavia

Abstract. In this paper the design features of a system for programming the industrial robots with dynamic control are described. During specification of the system special attention was given to achieving hardware transportability, simplicity of communication between the user and the system, possibility of active user participation in generation of control algorithms and possibility of implementing the software on a computer with relatively modest characteristics. The software is intended for implementation on a microprocessor-based system and should enable the user to control a robot via specialized programming language RL which incorporates structures for defining the positions and orientations of working points, motion specification and synchronization of the robot with its environment. The paper contains descriptions of the RL language and of the system structure. The main characteristics and advantages of the system as well as questions concerning its realization on existing microprocessors are also discussed.

Keywords. Robots; controllers; programming languages; control engineering computer applications.

INTRODUCTION

A number of systems for programming the industrial robots was developed in the last decade. Their capability of programming the robots in external coordinates and the 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 (Unimation Inc., 1980) can be used only for control of PUMA family robots, the AML (Taylor et al., 1982) for IBM System/1 robots etc. Recently, some systems that can be applied to various types of robots have been 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 trajectories cannot be achieved. For these reasons, the development of a new general purpose controller UCS-1 was commenced in the Mihailo Pupin Institute, Belgrade (Vukobratović et al., 1984). This paper describes the main design features of the software support of the controller.

DESIGN OBJECTIVES

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

The specific goals kept in mind during software design were:

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

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

- possibility of adapting the robot operation to the robot environment, especially of synchronizing 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 is designed;

- possibility of implementing the system on an inexpensive computer system and possibility of operating the system without utilizing mass memories, so that the probability of 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 the use of variables of various types, control structures and user-written subroutines;

- robustness of the system, i.e. protection of the system integrity against unauthorized use and accidental programming errors.

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

AN OVERVIEW OF ROBOT PROGRAMMING LANGUAGE

One of the 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 a somewhat different approach: our specific goal was to enable the field engineer, having some experience in programming, to exploint as much as possible the available robot without a 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 who are to be the main designers of automated manufacture 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 the clear structure of Pascal programs were also the 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, they are:

- definition of the robot task;

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

- teaching the robot, i.e. memorizing individual 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 of 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 a necessity for occasionally 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, or 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, especially 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 permanently memorized (end effector trajectory, position and orientation of working objects etc.). Such data base is implicitly included in all systems for programming the robots; however, we were of the opinion that the programming errors checking and the robot programs maintenance could be done much easier if the base were declared explicitly by the programmer and if all the tasks were described in one programming unit which we call a 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 can be seen, the package consists of the data base declaration, description of the initialization 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 data 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; for example, 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 1.0E-70 to 1.0E+70 including zero, with approximately 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,ψ,θ,ϕ) 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. As in Pascal, all identifiers used as variables or named constants must be explicitly declared. For example:

```
CONST MESSAGE = 'SUCCESS';
      APP = (0.0, 0.0, 20.0, 0.0, 0.0, 0.0);

VAR TRANSL: VECTOR;
    FAULT: BOOLEAN;
    PICK, PLACE, PIN, BLOCK: BODY;
    HOLE: ARRAY [1 .. 8]OF BODY;
```

Assignment of a value to a variable is achieved using the operator of assignment ":=". Assignments also can be performed by the operater of 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 operater's intervention 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 subtraction operators (+, -) and the operator for multiplication of a vector by a scalar (*).

Releationships 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 orientation represented by the first argument; the function INV returns the position and orientation of the referent frame with respect to the argument of INV. For example, if P represents the position and orientation of the robot gripper grasping the pin, and PICK represents the position and 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 languages, operator precedence can be overrided using parentheses.

The 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
```

causes a 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, and in that case the execution of the MOVE statement results in alignment of the robot effector with the body_expression. For example, if the EFF is 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:

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

Synchronization with the external equipment such as conveyors, feeders, sensory devices, etc., is 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 give the user flexibility in using input and output signals with different meanings. On the other hand, it provides the language translator with additional information 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 at any place of the program where the occurrence of the value of the same type is allowed. Output signals can be generated as a result of assigning values to the variables attached to output channels. Also, explicit waiting for an external event is possible using the WAIT command:

```
WAIT 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 testing of variables attached to input channels. The general form of the MOVE command is:

```
MOVE object TO body_expression
            VIA expr_1, expr_2, ..., expr_n
            DEPART depart_expr
            APPRO appro_expr
            WITH ctrl_1, ctrl_2, ..., ctrl_m
            UNTIL until_clause
```

In this form the expressions expr_1, expr_2, ..., expr_n return the positions and orientations of the intermediate points that should be passed by the object without stoping 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 is 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:

```
EFF = STARTING
EFF = STARTING*depart_expr*INV(object)
EFF = expr_1*INV(object)
EFF = expr_2*INV(object)
.
.
.
EFF = expr_n*INV(object)
EFF = body_expression*appro_expr*INV(object)
EFF = body_expression*INV(object)
```

Optional clause WITH enables the user to control the mode of movement. Controls ctrl_1, ctrl_2, ..., ctrl_n have the form:

```
keyword = value
```

where the keyword specifies a control parameter (SPEED, MAXTIME, PASSMODE, TOLERANCE, 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 joint coordinates (default mode, other alternatives are STRAIGHT, with linear change of Euler coordinates, and PARABOLIC, with polynomial interpolation) and in time less than or equal to 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 until_clause
```

where the integers number_1, number_2, ..., number_n are the indices of the robot joints, and the real expressions displacement_1, displacement_2, ..., displacement_n the corresponding displacements; Optional UNTIL clause is the same as in the MOVE command.

Two commands are provided for the gripper operation:

```
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 the corresponding output channels.

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

```
SET ctrl_1, ctrl_2, ..., ctrl_n
```

where the 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 grouped into procedure of 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 exception are the variables explicitly declared by the programmer as public; for example:

```
BASE PICK: PUBLIC BODY;
```

enables the operater to freely change the value of the variable PICK.

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 (Vukobratović and Kirćanski, 1985) 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 (Vukobratović et al., 1984):

- 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 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 very complex. However, it has been shown that it is not necessary to use the complete model: as an example, according to investigations by Vukobratović and Stokić (1982), Coriolis and centrifugal forces can be neglected in most cases without losing the robot performance. Thus, some dynamic effects can be neglected and the computation therefore reduced.

An example of the computational complexity is shown in Fig. 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 the period of sampling input data on joint angles and velocities is included in the previously mentioned procedure for the generation of models for controlling the robot motion. In this manner, a satisfactory adjustment of the system to the particular application can be achieved.

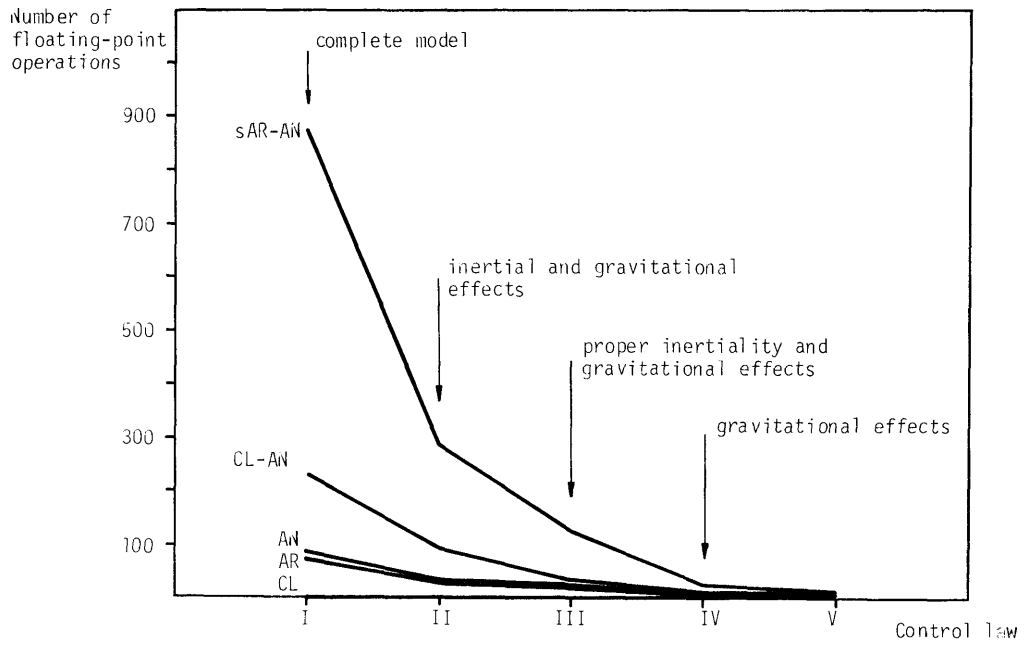

Fig. 1. Number of floating-point operations in decentralized control structures

SOFTWARE ORGANIZATION

The software is designed as a multiprocessing system consisting of a set of interconnected modules (Fig. 2.). The central part of the system is denoted as monitor and its main functions are allocation of processor time to individual proccesses such as process for accepting the user commands or process for interpreting RL packages, and synchronization between active 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 the analytical models of the robot kinematics and dynamics, produces the 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 gains. The parameters and the models can easily be 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 are supported by the specialized line editor which also performs partial syntax error checking during the editing. After completition of editing, the control is automatically transferred to the RL translator. It translates the source RL program into the 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 always be 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 a 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 values 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 channels are under control of the interpreter. The kinematic module realizes on line trajectory syn-

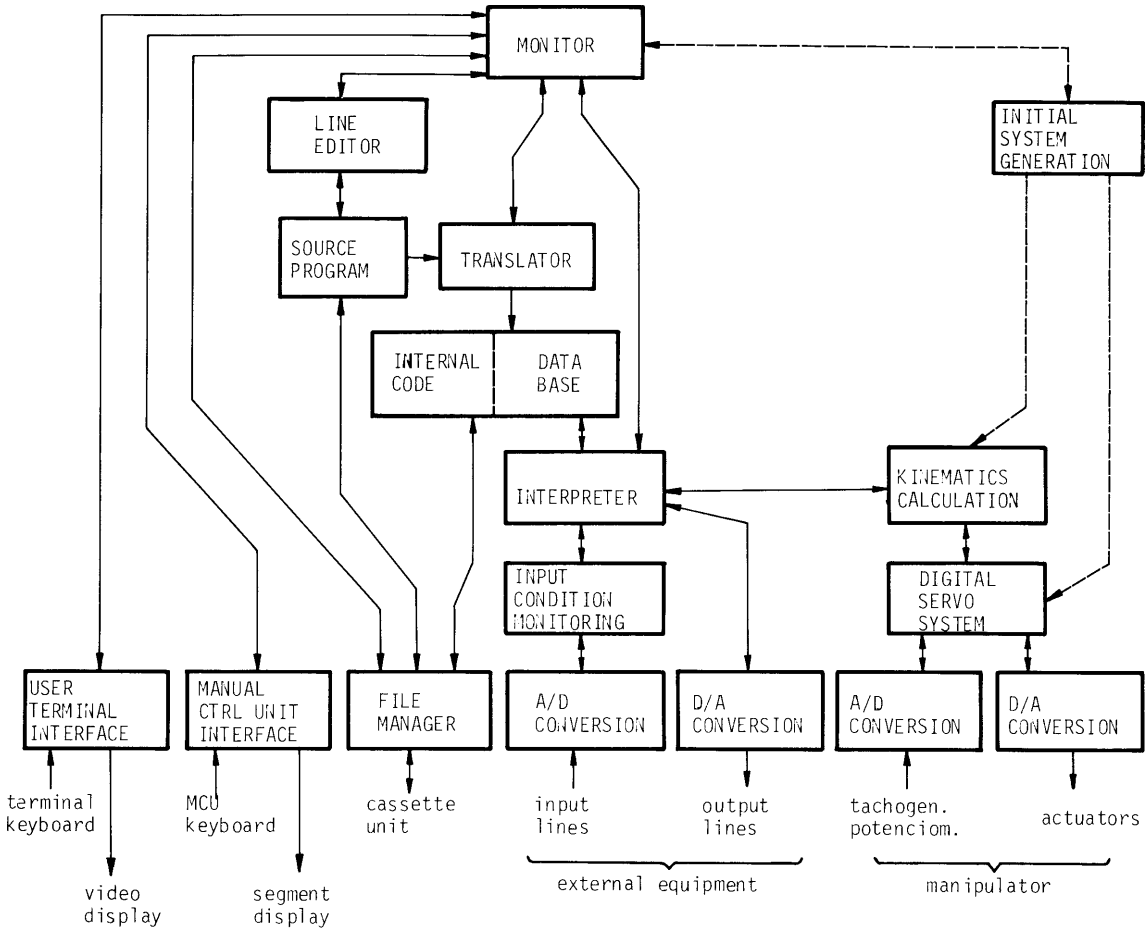

Fig. 2. Organization of the Software

thesis in the space of the robot joint coordinates until the desired point is reached or until the end-of-motion condition tested by the module for monitoring external signals becomes true. The operation of the digital servo system is explained in the preceding 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 potentiometers, and current velocities in the joints supplied from tachogenerators.

CONCLUSION

The main problems in the design of the system were, on one hand, to ensure on line computation of trajectories in the robot joint coordinates and dynamic digital control, and, on the other, to ensure efficient programming of robots with no need for using expensive and complex processors. We find the presented solution quite acceptable from the standpoint of equipment complexity and price. It enables a good performance to be achieved and makes the system quite independent from the robot structure.

REFERENCES

Taylor, R.H., P.D. Summers, and J.M. Meyer (1982). AML: A Manufacturing Language. Robotic Research, 3, 19-41.

Unimation Inc. (1980). User's Guide to VAL, Version 12. Unimation Inc., Danbury, CT.

Vukobratović, M., and D. Stokić (1982). Control of Manipulation Robots. Springer-Werlag, Berlin.

Vukobratović, M., N. Kirćanski, D. Stokić, M. Kirćanski, and B. Karan (1984). General Purpose Controller for Industrial Manipulators. Proc. of the Second Yugoslav-Soviet Symposium on Applied Robotics, Belgrade.

Vukobratović, M., and N. Kirćanski (1985). Computer Assisted Generation of Robot Dynamic Models in Analytical Form. Acta Applicandae Mathematicae, 2, 49-70.