Karan B., Timotijević M.,
Djurović M., Devedžić V., Josifović N.
"Mihajlo Pupin" Institute, Beograd, Yugoslavia

A SYSTEM FOR PROGRAMMING THE UMS-7 INDUSTRIAL ROBOT

ABSTRACT - The paper gives an overview of some characteristic features of the software support of the controller for the UMS-7 industrial robot. The software is intended for textual robot programming and is implemented on a microprocessor based system.

1. INTRODUCTION

The UMS-7 is a general purpose robot system which includes the UMS-7 arthropoid manipulator and the UMS-7 controller. The manipulator [1] is powered by d.c. motors and is primarily devoted for relatively noncomplex manipulation tasks, such as spot welding and simple materials handling. The manipulator controller [2, 3] is a microprocessor based system which in its minimal configuration consists of three processor boards in parallel operation, global memory, an A/D board with up to 32 twelve bit analog input lines, a D/A board with up to 8 twelve bit analog output lines, peripheral parallel interfaces enabling handling up to 256 single bit digital signals connecting the controller to external equipment (other machines, sensory system etc.) and a commanding panel. The number of analog and digital lines is expandable. The processor boards are built arround Intel 8086 microprocessors with airthmetic coprocessors and include local memories, serial and parallel interfaces, real time clocks and watch - dog timers.

The software support of the controller is strictly hierarchically organized and conceptually consists of three layers: commanding level, base level and the system nucleus.

The commanding level acts as an interface between the base level and a user of the system. Its modules enable the user to enter commands for manual guiding the manipulator, to request a direct execution of a particular robot command, to create, modify, debug and execute robot programs, to adjust controller parameters etc. Commands recognized at this level can be divided into two main groups: local

commands (such as commands for computation of geometrical transforms, commands for redirecting the flow of execution etc.) and commands controlling the operation of modules at the base level (such as manipulator motion commands, commands incorporating access to external signals etc.). The main modules at the commanding level are the dialogue processor (DP) incorporating a specialized line editor and supplying the user with a tutorial support during operation of the system, the robot language parser (PAR) and the robot language interpreter (INT).

The base level consists of three modules: the commanding panel interface (CPI), the external signal processing module (IOP) and the manipulator control module (MC). The task of the module CPI is to enable an asynchronous access to the commanding panel from the modules at the commanding level. Its procedures perform reading and elementary processing of signals sent from the commanding keyboard, displaying messages at the terminal screen and generating diagnostic signals at the commanding panel. The module IOP performs monitoring input and generating output digital and analog signals, enabling in this manner synchronization of the manipulator operation with external hardware. The module MC is the central part of the system; it includes three submodules: manipulator control interface (MCI), a kinematic module serving as an arm resolver (KIN) and digital servosystem (DSS). All manipulator motion commands are directed from the commanding level to the module MCI; its responsibility is to fill uncompleted motion specifications with default values, to check feasibility and allowance of requested motion and to generate descriptors of manipulator trajectory segments, as well as to synchronize the operation of module MC with the operation of modules at the commanding level. The module KIN can essentially operate in one of the two modes: velocity controlled motion (implemented in manual guiding) and position controlled motion. In both cases, the module KIN accepts trajectory segment descriptors as well as other special commands from modules at the commanding level (for example, commands for unconditional stopping the motion in progress) and performs transformation of external (Cartesian) coordinates into manipulator joint coordinates. The module DSS accomplishes tracking of calculated joint coordinates taking into account feedback information from potentiometers, shaft encoders and tachogenerators located in the manipulator joints. This module incorporates feed-forward compensation calculated on the basis of manipulator dynamics and digital local servoregulators with proportional, integral and derivative control actions. The last two modules (i.e., KIN and DSS) are numerically the most complex parts of the system and occupy most of its computational resources. In fact, a distribution of the computation is performed in the controller so that distinct microprocessors are devoted to calculation of kinematics and dynamics, including digital servoregulators ("kinematics processor", "dynamics processor"), while the rest

of the software runs on the third processor ("communication processor").

The system nucleus has a responsibility to assure a distribution of processor time between modules at the commanding and the base level and to provide basic synchronization primitives for communication between the modules.

More detailed description of the system organization as well as description of algorithms implemented in particular modules is beyond the scope of this paper. In following sections we will rather concentrate on description of some characteristic features facilitated by modules at the commanding level.

2. ROBOT PROGRAMMING LANGUAGE OVERVIEW

The system is designed to enable textual robot programming resulting in a robot program, which is itself a sequence of individual statements written in a specialized language and representing corresponding robot commands.

The robot commands can include operations on logical, byte, integer, real and point data types.

Logical data can take values from the set {0, 1}, byte data can take integer values in the range 0 to 255 and integer data in the range -32768 to +32767. Real data can take absolute values in the range 1.0 E-38 to 1.0 E+38, with 6 significant decimal digits.

Point data serve for more convenient handling of manipulator positions. The position can be memorized by the system either in external (i.e. world) or joint coordinates. The position in external coordinates is defined through Cartesian coordinates and Euler's angles of the manipulator tip. On the other hand, the position in joint coordinates is defined through six joint angles of the UMS-7 manipulator. Coordinates are expressed in milimeters or degrees depending on their nature. All linear coordinates are stored with precision of 0.1 mm, while the angle coordinates are stored with precision of 0.1 degree.

Mapping from external to joint coordinates is not unique for the UMS-7 manipulator. For this reason point data representing positions in external coordinates can optionally include three additional indicators enabling a selection of proper solution in joint coordinates. The indicators are referred to as joint orientation selectors and can take values from the set {0, 1}. Setting the appropriate selector corresponds to the selection of desired orientation of the UMS-7 arm, elbow or wrist joint.

Data appear in robot programs as constants or variables. Scalar constants can be written as in a majority of other programming languages. Point constants

are formed by following the syntax:

#<type of coordinates>: <values of coordinates>: <optional selectors>#

as in robot language statements:

```
mov #world: 1600, 1000, 78, 90, 70, 60: 0, 1, 1#
mov #joint: 30, -60, 180, 0, -10, 0#
```

Variables can be referred symbolically in a program through associated identifiers which start with a letter and may be followed by up to eight letters, digits and underscore characters. The use of identifiers is restricted to combinations that do not match reserved words as "mov", "call" etc. Variables with associated identifiers must be explicitly declared before their use, as in statements:

```
real x
point p
```

For user convinience another form of referring the variables is supported. It is ilustrated in the statements:

```
point@1 := #world : 0, 0, 0, 0, 0, 0#
real@20 := x - 2.0
```

Where the names of variables contain appropriate data types and indices (integer values in the range 1 to 1023) which uniquely define the variables. The use of this form does not require explicit declaration of a variable and decrease necessary storage several times compared with the case of variables with associated identifiers, but on the other side makes the robot program less readable.

All variables must be given a value before they are used; otherwise, the system reports an error.

The system enables computation of scalar expressions with arithmetic operators "+", "-", "*", "/" and with optional use of parenthesis. Automatic conversion of data types into expressions is performed where possible. Scalar expressions can be compared using relational operators ">", "<", ">=", "<=", "=" and "<>". The result of the comparison is a logical datum with value 1 if the relation is valid and 0 otherwise.

Expressions with point data are also supported. For these data a binary

operator "+" meaning the composition of space relationships represented by operands and an unary operator "-" meaning the inversion of the operand are defined. If <p1> and <p2> are point data, then the construction <p1> - <p2> has the same meaning as <p1> + (-<p2>). The use of point expressions is limited to the case when all operands are defined in external coordinates. Comparison of point data is also allowed with relational operators "=" and "<>".

All executive robot commands can be roughly divided into five groups: assignment commands, print command, commands for redirecting the flow of execution, motion control commands and commands enabling the handling of external conditions.

Besides the usual operator ":=" of assignment, the system supports two special commands, namely "here" and "jhere", enabling assignment of values corresponding to a current manipulator position and orientation to point variables. The first command is used when external coordinates are to be memorized, and the second one when joint coordinates are preferred.

The system allows conditional execution of commands through the construction:

if <expression> then <command>

<command> is executed only if the value of <expression> is different from 0.

Every executive statement can be optionally preceded by a label which is formed by a concatenation of an identifier with the colon character. Command "goto <identifier>" allows unconditional branching to the statement labelled with <identifier>. Commands "call <identifier>" and "ret" allow the use of subroutines, making the program more readable and efficient.

Program execution can be broaken via commands "break" and "abort". The difference between the two is that the first allows to later continue execution from the break point, while the second does not. Command "restart <identifier> " break the execution and reinitiates it from the specified label.

Commands enabling manipulator motion control and handling of external signals are described in the next two sections.

3. MOTION CONTROL

The language incorporates a set of commands for specifying desired motion of the manipulator. All motion commands have the syntax:

<motion type keyword> <point expression>

Where the motion type can be "mov", "jmov", "movi" or "jmovi". The first

one denotes a motion with a linear change of external coordinates. The second corresponds to a motion with a linear change of joint coordinates. In both cases motion is performed to a destination point defined by <point expression>. Types "movi" and "jmovi" in contrast to "mov" and "jmov" interprete point expression as an incremental movement relative to the manipulator current position.

There is a number of commands that specify more closely the desired parameters of the manipulator motion. Their common format is:

<motion parameter keyword> <argument>

Commands "arm", "elbow" and "wrist" have as arguments desired joint orientation selectors after execution of the next motion command. The command "time" has as an argument desired time of movement, the command "speed" have as an argument desired relative speed of movement during the execution of the next motion command, etc.

4. EXTERNAL SIGNAL PROCESSING

The robot controller is connected to peripheral equipment via digital and analog physical lines. Six input digital lines, six analog input lines and six output analog lines are connected respectively to shaft encoders, potentiometers and actuators in the UMS-7 joints. All other lines are under explicit control of the user.

All digital signals enter the system as single bit values (switch signals). In order to achieve more efficient way to handle digital signals and to enable the user to examine more than one digital signal at a time, they are clustered in groups of 8 or 16 bits.

The language has no special commands for reading or creating signals. Both analog and digital signals as well as groups of digital signals are accessed in the UMS-7 robot language in the same manner as variables, via system defined or user-defined identifiers. References to analog signals can appear in programs wherever integer variables can. The user can access digital signals and groups of digital signals in the same manner as logical, byte or integer variables, depending on their range of values. Obviously, input signals behave as read-only variables.

Values of all signals are periodically refreshed by the system in a dedicated memory called external signal area. Default values of output signals that should be sent to all output ports in the case of system failure are saved in another memory area denoted as save area. The user can examine and change the default values by using prefix "&" with references to hardware output signals.

Besides the already described synchronous access to external signals, the user can declare external conditions that are to be permanently monitored during execution of a robot program; the declaration should follow the form:

cond <condition identifier>: <condition>

Where <condition> is a simple relational expression relating the value of an input signal to the value of some constant, variable or another input signal. The actual monitoring of the condition starts with the appearance of the statement:

when <condition priority> <frequency> <type> <condition identifier>

do <command>

At this moment the robot language interpreter RLI puts the condition as well as its associated priority (integer value in the range 1 to 15), selected monitoring frequency and selected type of monitoring on the list of monitored conditions. The process of monitoring, i.e. scanning the list of monitored conditions is performed by the module IOP asynchronously to the operation of the interpreter. When a condition becomes satisfied, module IOP tries to interrupt the RLI operation forcing it to process the interrupt, i.e. to interprete the <command>. The interrupt is acknowledged if the interpreter does not process some other interrupt of higher priority. If the interrupt is not acknowledged, the satisfied condition is put by the module IOP on the waiting list. At this point it should be noted that the <command> may be any executive robot command, including the command "restart". Also, if the command is a call to a subroutine, the interpreter is in the state of processing the interrupt until the corresponding return from the subroutine is executed.

The monitoring frequency can be selected using keywords "fast", "med" and "slow", i.e. conditions can be scanned every elementary time interval, every 10th interval or every 100th interval respectively. The type of monitoring is selected with keywords "edge" and "level". If the first one is selected, monitoring stops immediately after the condition becomes satisfied. If the user specifies the type "level", module IOP does not automatically remove the condition from the list of monitored conditions.

The user can temporarily disable and later enable selected interrupts using statements:

disable <priority>

enable <priority>

Finally, conditions can be definitely removed from the list of scanned conditions using statements:

ignore <condition identifier>

ignore all

Some characteristic system data are denoted as system signals and are processed the same way as actual external signals. Examples of such data are the position of the manipulator tip (input signal), the default speed of motion (output signal) etc. Software timers are handled in the same manner: the system supports eight software timers which can be activated by sending signals to appropriate output software lines and examined by reading signals from corresponding input software lines.

5. USER INTERFACE

The interface between the user and the system is achieved through the dialogue processor DP which supports multi level menu-driven dialogue. The dialogue is performed via video terminal (the current version of the system requires VT100 Digital or other compatible terminal).

On each level of the dialogue a menu of allowed commands is displayed on the last line of the screen. An allowed command can be entered by pressing the corresponding key at the terminal keyboard (usually the key corresponding to the first letter of the command). From any level to the next higher level the user can jump after pressing the key "espace".

It is worth noting that a lot of additional features are incorporated in the dialogue processor in order to facilitate the process of programming as much as possible. For example, manual guiding motion can be performed in one of the three modes: with a linear change of the UMS-7 joint coordinates, with a linear change of external coordinates of the manipulator tip defined with respect to the manipulator base and with a linear change of external coordinates defined with respect to the coordinate system connected to the manipulator gripper. The last one is especially suitable for fine tuning the gripper position in the vicinity of a manipulated object. A special form of motion specification statements without arguments can be entered during the process of editing the program: the system adds the current manipulator position as the destination point by default. Debugger incorporated in the system enable to set break points in the program, to execute the program step by step, to display and change values of program variables, to change the speed of the robot motion etc.

6. CONCLUDING REMARKS

The described system is a first step toward a more powerful robot programming system which is under development in the "Mihajlo Pupin" Institute [3, 4]. It is programmed entirely in high level languages (PL/M and FORTRAN 77) and possesses a strong modular structure facilitating its further expansion. Its future versions should provide better hardware transportability (i.e. a possibility of adapting the system to a particular manipulator and a particular application by the customer) and more efficient robot programming by implementing a more general robot program structure as described in [4].

7. REFERENCES

[1] Hristić D., Vukobratović M., "State and Perspectives in the Production and Application of Industrial Robots in Yugoslavia", Proc. III Soviet-Yugoslav Symposium on Applied Robotics, Moscow, 1986.

[2] Vukobratović M., Karan B., Kirćanski M., Kirćanski N., "Concept of a Multiprocessor Robot Controller",(in Serbocroatian), Proc. XXX ETAN, Herceg Novi 1986.

[3] Vukobratović M., Kirćanski N., Stokić D., Kirćanski M., Karan B., "General Purpose Controller for Industrial Manipulators", Proc. II Soviet-Yugoslav Symposium on Applied Robotics, Arandjelovac 1984.

[4] Karan B., "A Software System for Teaching and Commanding the Industrial Robots, Proc. I IFAC Symposium on Robot Control, Barcelona 1985.