10.1515/acsc-2015-0014

Archives of Control Sciences
Volume 25(LXI), 2015
No. 2, *pages 215–225*

Parametric programming of industrial robots

PAWEŁ SZULCZYŃSKI and KRZYSZTOF KOZŁOWSKI

This article proposes the use of parametric design software, commonly used by architects, in order to obtain complex trajectory and program code for industrial robots. The paper describes the drawbacks of existing solutions and proposes a new script to obtain a correct program. The result of the algorithm was verified experimentally.

Key words: robotics, robot programming, CAAD/CAM, parametric design.

1. Introduction

Industrial robots supersede human in difficult, tedious, repetitive and sometimes dangerous activities. They can be found mainly in large factories engaged in mass series production. 50 years have passed since the first use of a robot in the industry to the creation of today's robotized stations. The dynamic development of robots and their control systems has increased theirreliability and reduced their prices. Robots are no longer expensive devices and have become tools used in most factories all over the world, where they deal with welding, painting, assembling, milling, packaging, palletizing and so on. The possibility of programming the robots in any way and their general availability contribute to the need of exploring new areas of their applications [7, 13, 6].

This multifunctionality of industrial robots has also been recognized by architects and designers [1, 2, 4, 11], who have encountered problems with the accomplishment of their projects created in programs such as CAAD (Computer Aided Architectural Design). Modern CAAD systems allow designing various types of models, the shape of which depends on the adopted parameters. This makes it possible to change or manage designed forms easily [5]. However, transferring these changes to the CAM (Computer Aided Manufacturing) remains a problem.

This paper presents the existing test solutions for CAAD systems that are not devoid of drawbacks. Then the authors' module is proposed and the result of its works is verified experimentally.

The Authors are with Poznan University of Technology, ul. Piotrowo 3a, 60-965 Poznan, Poland. E-mails: {firstname.lastname}@put.poznan.pl

Received 12.11.2014. Revised 20.04.2015.

2. Parametric programming

2.1. Robot programming

Programming of industrial robots is generally done in two ways: online and offline. The first method involves teaching by showing, i.e. the operator moves the robot by hand to the required positions and then logs these positions into the memory. An offline programming is done in special programs in which the robot's movements are determined on the basis of specified geometry. This is translated into the language of programming of the robot and then saved in a file. The file prepared in this way must be transferred to the control system of the robot and then executed.

The study was based on KUKA robots Agilus (KR 6 sixx R900). These robots are programmed in KRL (KUKA Robot Language), which in its general structure is similar to Pascal. The main difference is the program code is divided into two files. The first file with the extension *.src contains the structure of the program which includes motion commands, loops, declarations of local variables, arithmetic, etc. The second file contains mainly declarations of points and the associated additional information. The location point is stored in the coordinate axis (AXIS) or Cartesian coordinates (POS), where the orientation is expressed by Euler angles (RPY). For movement between two points PTP commands (fastest movement from point to point) and LIN (movement along the shortest path) are used.

2.2. Parametric programming

One of the most popular tools for parametric programming, used by architects, is Rhinoceros - a free addition to the CAD program Grasshopper. It allows parameterizing graphic design and producing virtually any shape obtained by using all available mathematical tools. This type of design is often used for analyzing sunlight, natural, unforced ventilation or communication in large buildings. It is also employed in modern design to generate natural curved shapes. Parametric project is dependent on certain checkpoints and therefore any changes in design or appearance are not labor-intensive.

2.3. Parametric robot programming

One of the possibilities of combining parametric programming with the programming of KUKA robots is to use the module KUKA|prc [3]. This module enables visualizing the robot model in a graphical environment Rhinoceros (Rhino). The programming itself is done in Grasshopper's add-on where the reference trajectory is generated using the available blocks or can be directly drawn from Rhino. In addition, the module allows defining the size of the robot or tool dimensions and detects whether the robot reaches the preset points.

The result of the module KUKA | prc file is *.src written in KRL. An example of a program that uses the module KUKA | prc is shown in Fig. 1.

Since the resulting program was not always performed by the robot the authors decided to carry out an analysis of the module KUKA | prc and the resulting code. The

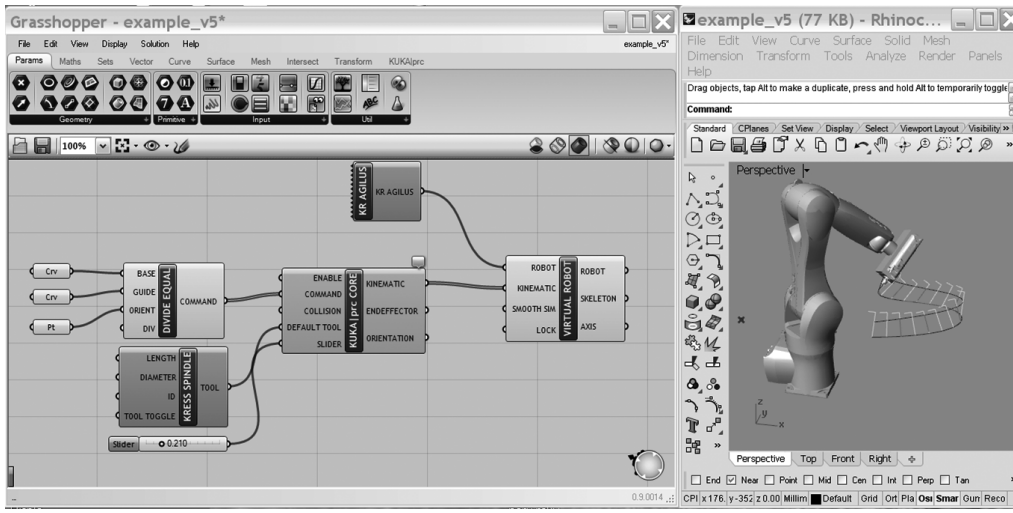


Figure 1: Example view of Grasshopper and Rhinoceros program.

trajectory generated either in Rhino or in the Grasshopper is approximated by straight lines. The number of these lines depends on the parameter indicated by the user. When selecting this parameter one should be aware of the fact that the frequency distribution of the curve has a significant impact on the quality of approximation. With insufficient division one can completely lose the point of the curve shape, e.g. specifying a circle 4 points one will receive a square. In order to smooth the trajectory C_DIS command language KRL was used, thanks to which the robot does not commute to selected points but bypasses them in a given distance. Road approximation consists of two segments of a parabola, which tangentially pass one into another and at the same time communicate with the reference trajectory. A disadvantage of dividing the curve into segments of equal length is that an excess of credit is generated. A long straight is unnecessarily divided into sections. A large number of points contributes to the file size and even though the program can be run on the robot it is not possible to re-edit it on the operator panel.

Some difficulty in transferring the program to the robot is the definition of coordinate systems and tools. It has to be remembered that the definitions in the program need to be identical to those in the robot.

The most serious drawback of the module KUKA|prc is the lack of information about the configuration of the robot. It is known that for 6-axis robots it is usually possible to obtain 4 inverse kinematics solutions. The KUKA robots resolved this ambiguity by adding bits of status and turns. No declaration of these bits results in the fact that the robot in reality moves differently that in simulations and sometimes it is not able to perform this movement.

Because of the above-mentioned disadvantages programming of robot using KUKA|prc module requires the user to have a specific knowledge about programming in KRL. It allows understanding why the program does not work and how to improve it.

3. A new approach to parametric programming robots

3.1. Calculation of information about the orientation of the robot

An important advantage of the module KUKA|prc is that it is free. Unfortunately, this program is not open software so making direct changes in the module is not possible. Therefore, it was decided to extend the code with the necessary features using the tools available in the module. For this purpose the block "Custom Command" was used to manually enter commands by means of which the movement command was added to take account of information about the orientation of the robot, i.e. the variable Status and Turn. The designation of these variables is not a task simple enough to perform it in memory and therefore for this purpose a script was written in built-in script editor Rino VBscript that sets them on the basis of a specified point and configuration information for the robot joints. Variable turn defines 6 bits, each assigned to one of the joints, which takes the value 1 for angles less than zero and the value 0 otherwise. The status is composed of three bits. The first contains information on whether the robot wrist is in front or behind. The second defines the relationship between the arm and the wrist of the robot and the third is associated with position of the fifth axes [10].

3.2. Hop2Kuka - calculation

Despite the numerous advantages of the module KUKA|prc it is time consuming to write a simple program with it. Therefore, a new application (script) was written, Hop2Kuka, thanks to which it is easy and fast to generate code in KRL from the trajectory designed in Rhino or Grasshopper.

Just as it is the case in the module KUKA|prc the curve trajectory of the tool tip is divided into straight sections. Unfortunately, this information is insufficient. To determine the trajectory a robot needs a set of points defining both the position and orientation of the tool tip. Since such information cannot be included in a curve, for the full description of the trajectory of the tip of the robot tool, two additional curves were used. In Fig. 2, three curves denoted by t, x and o are presented. The curve t defines the position of the working point of the tool. Additionally curves x and o are used to determine X_{TOOL} and tool orientation, respectively.

By sharing all the curves equally, for each position of the tip of the robot, a set of three points $P_{ti} = t(i)$, $P_{xi} = x(i)$ and $P_{oi} = o(i)$ is obtained, where $i = 1,2,\ldots,N$ denotes the index of segment while N is the number of total segments. Using the basic operations on vectors $P_{xi}P_{ti}$ and $P_{oi}P_{ti}$ we set a Cartesian coordinate system which specifies the orientation of the tool $(X_{tool}, Y_{tool}, Z_{tool})$. In a KRL environment the target point is defined either as information about the location of each robot axis (type AXIS) or Cartesian orientation stored using Euler angles (type POS). It should be noted that the linear movement can take place only to position POS. Therefore, based on the coordinate system that specifies the orientation of the tool Euler angles were determined by commonly available formulas [8].

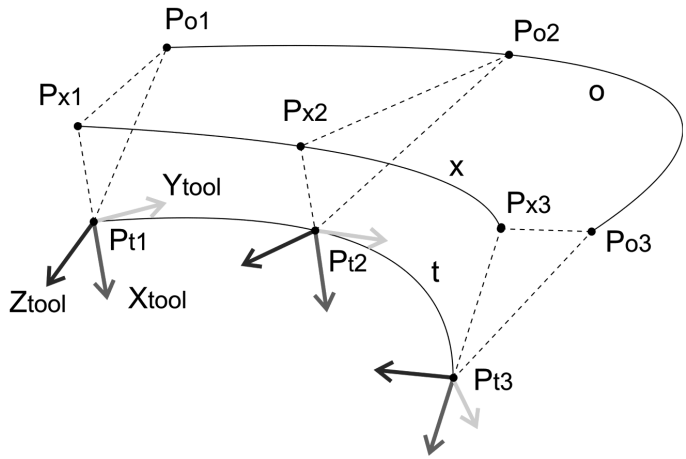


Figure 2: Calculating the position and orientation based on three curves.

In order to determine information about the orientation of the robot the script described in section 3.1 was used. The script was modified since the information about the position of each axis module came from KUKA | prc. Consequently, using the information about the size of the robot (Tab. 1 and Fig. 3) the Denavit-Hartenberg parameters (D-H) of the crawler (Tab. 2) were defined on the basis of which direct and inverse kinematics of the robot Agilus were calculated.

Because the orientation in space of the tool is to change during the path motion, the orientation control mode must be set using the system variable $ORI_TYPE. The value of this variable remains valid until a new value is assigned. In cases where the tool direction is of particular interest recommended option is #VAR. In this mode the orientation of the tool changes continuously from the start position to the end position. However, if the trajectory is near a singularity maintain the programmed velocity may not be possible. In this case, use of option #JOINT is necessary but then the orientation change is not linear. Simple method of avoiding the singular position represents the work [12].

3.3. Hop2Kuka – code generation

KRL syntax is largely similar to Pascal. Basic information about the available commands, variables, and method of writing the program can be found in the technical documentation [10].

Each program *.src in KRL consists of three sections:

- declaration,
- initialization,
- instructions.

Table 6: Parameters of the robot shown in Fig. 3 [9].

A [mm]	B [mm]	C [mm]	D [mm]	E [mm]	F [mm]	G [mm]	H [mm]	I [mm]	J [mm]
1276	1620	901.5	656	245.5	851.5	420	455	400	855

Table 7: Parameters D-H for the robot KR 6 R900 sixx AGILUS.

i	α_i [rad]	d_i [mm]	a_i [mm]	θ_i [rad]
1	$-\pi/2$	$d_1 = I$	$a_1 = 25$	$q_1 = \pi/2$
2	0	0	$a_2 = H$	$q_2 = -\pi/2$
3	$\pi/2$	0	$a_3 = 35$	$q_3 = 0$
4	$-\pi/2$	$d_4 = -(G-1.4)$	0	$q_4 = 0$
5	$\pi/2$	0	0	$q_5 = 0$
6	π	$d_6 = -80$	0	$q_6 = 0$

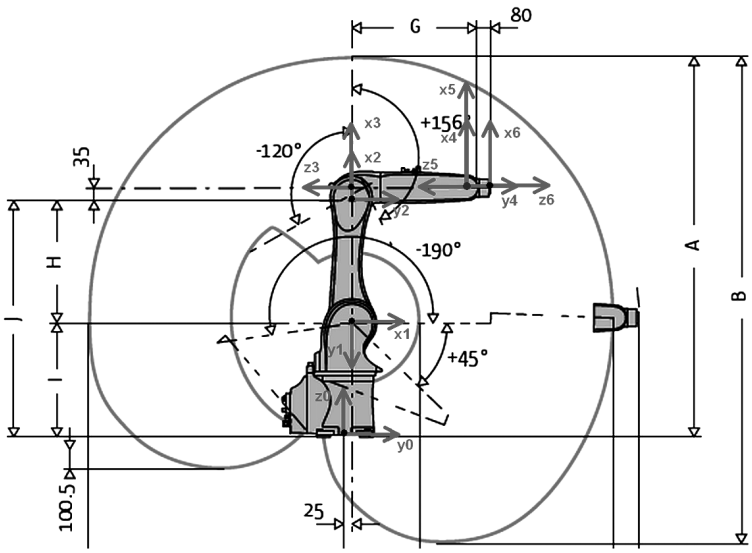


Figure 3: Dimensions of the robot KR 6 R900 sixx AGILUS [9] with marked coordinate systems.

Since all the necessary calculations are performed in the script VBscript, declaration of variables in the KRL file is not necessary, either.

In the initialization section to the default speed, acceleration, performance tools and databases are set through command BAS (# INITMOV, 0). A sequence of commands associated with the security system is also implemented (GLOBAL INTERRUPT Decl 3 WHEN $ STOPMESS == TRUE TO IR_STOPM (); INTERRUPT ON 3), which ensures that the robot returns to the programmed path in case of the robot emergency stop.

The instruction section includes primarily motion commands. In the script only two types of these commands were used: Command PTP providing the fastest movement to the set point and click LIN performing movement along a straight line. The first command is a command direction to the home position (ptp XHOME), which is a global position, adjustable by the user and by default taken to be safe. Then set the base and the robot tool are specified by the user ($ BASE = POS; $ TOOL = POS). In the following steps commuting to selected items is performed (e.g. .: LIN X 980, Y -238, Z 718, A 133, B 66, C 146, S 6, T 50 C_DIS). The program ends with instructions to return to the home position.

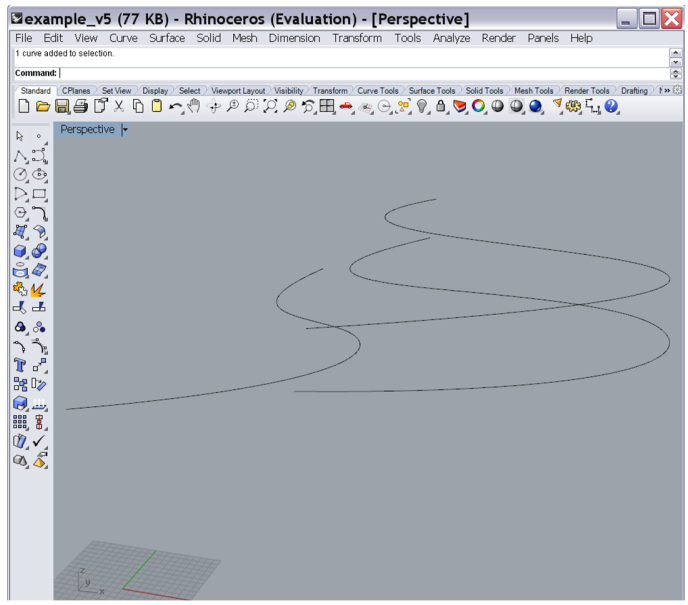
The end result of the program is shown in Fig. 5. The red lines indicate the planned trajectory in Rhino. The reference trajectory is divided into 13 parts (points P1-P13). Black lines connecting individual points represent the trajectory uses LIN command. The blue color indicates the trajectory of the DIS option uses the command LIN. Number of points dividing the original trajectory was chosen so to showcase defects in the applied solution. From Fig. 5 it is clear that the actual trajectories do not overlap with the reference trajectory. But it is possible to determine the maximum error of mapping trajectory. This property comes from the fact that the distribution of selected number of points to get only a concave or convex function (no inflection points) for each segment. Tracking Error is should not exceed the sum of maximum distance between the reference trajectory and the straight line connecting two points and value of variable $APO.CDIS (If C_DIS is specified as the approximate positioning criterion, the distance from the start of the approximation to the corner point corresponds to the value of $APO.CDIS).

4. Results of the program Hop2Kuka

Hop2Kuka written script has already been tested on the robot KR 6 sixx R900 Agilus in the Chair of Control and Systems Engineering.

After starting Rhino the three curves were outlined determining the position of the tool tip, the axis of approach tools and tool orientation relative to the axis of approach as shown in Fig. 4a. Then the script Hop2Kuka was launched. After that the other relevant curves were indicated and a value for the crossover frequency curves and performance of the tool was given. On the basis of the obtained data the trajectory was determined, marked in blue in Fig. 4b. In order to better disclose the resulting trajectory, required curves were divided only into 4 parts. As a result, the program generated two files. The first with the extension *.src contains the program code and the second with the extension *.dat includes information on the location of points.

a)



b)

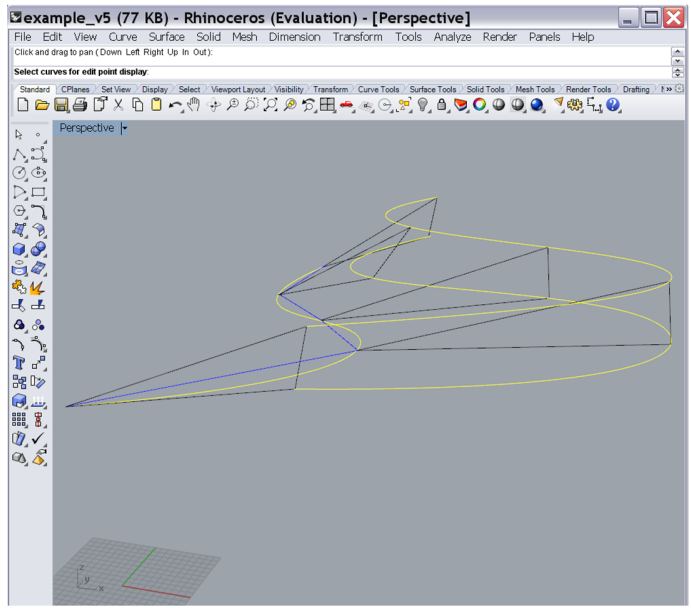


Figure 4: Screenshots of the program Rhino: a) – the curves that define the trajectory of the robot, b) – designated trajectory (blue).

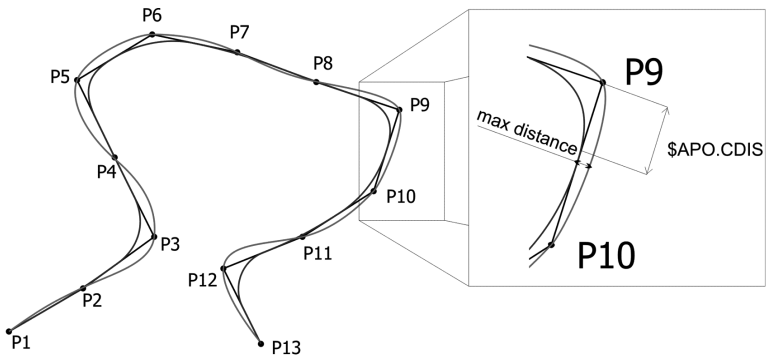


Figure 5: Final result.

The file was transferred to the robot system and successfully executed. Transfer on the trajectory took place in a continuous movement and internal points were omitted as described in Section 2.3.

5. Summary

The article proposes the use of graphics software for parametric design in order to obtain complicated trajectories for industrial robots. The paper describes existing solutions and tested constraints they are subjected to. It also proposes a new script that allows obtaining a correct program for the robot KUKA Agilus easily and fast. The result of the algorithm was verified experimentally.

Further work may be considered to optimize the number of points needed to restore the preset curve. Control of the parameter responsible for the continuous path (CP) also seems possible. Note should also be taken of other motion instructions that are available in the KRL environment, especially command Spline. Generally, such paths can also be generated using approximated LIN and CIRC motions, but Spline has more advantages. The most important disadvantage of approximated LIN and CIRC motions is that the path changes in accordance with the override setting such as velocity or acceleration. For this reason, in many cases it is not possible to calculate the path. In Spline motion the path is defined by means of points that are located on the path and the programmed velocity is maintained. It contributes to the programmed velocity. The problem that remains is how to detect KUKA algorithm to generate Spline curves.

It also seems useful to transfer the code to the .NET and use RhinoCommon SDK and SDK Grasshpper libraries to form a module similar to KUKA|prc. This change allowed inserting it into the Rino environment of the robot model. It seems to be interesting to implement the collision detection algorithm and transition of the vicinity at singular points.

References

[1] A. AIGNER and S. BRELL COKCAN: Surface Structures and Robot Milling. The Impact of Curvilinear Structured Architectural Scale Models on Architectural Design and Production. In: Innovative Design & Construction Technologies. Building complex shapes and beyond, I. Paoletti (Ed.), Milano, 2009, 433-445.

[2] S. AMBROSZKIEWICZ STANISŁAW, A. BORKOWSKI, K. CETNAROWICZ and C. ZIELIŃSKI: Intelligence around us; cooperation of software agents, robots, intelligent units. Academic Publishing House EXIT, Warsaw, 2010. Monographs of the Committee for Control and Robotics of the Polish Academy of Sciences, in Polish.

[3] J. BRAUMANN and S. BRELL-COKCAN: Parametric robot control: integrated CAD/CAM for architectural design. *Proc. 31st Conference, Association for Computer-Aided Design in Architecture*, Banff, (2011), 242-251.

[4] S. BRELL-COKCAN and J. BRAUMANN: Computer Numeric Controlled Manufacturing for Freeform Surfaces in Architecture. In: Emotion in Architecture, D. Kuhlmann, S. Brell-Cokcan, K. Schinegger (Ed.), issued by: Institut f ur Architekturwissenschaften, TU Wien; Luftschacht, Wien, 2011.

[5] C. EASTMAN, P. TEICHOLZ, R. SACKS and K. LISTON: BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors. John Wiley & Sons, Inc., New Jersey, 2008.

[6] J. HEINDL, M. OTTER, H. HIRSCHM ULLER, M. FROMMBERGER, N. SPORER, F. SIEGERT and H. HEINRICH: The robocoaster as simulation platform – experiences from the first authentic Mars flight simulation. *Proc. of the Motion Simulator Conf.*, Braunschweig, Germany, (2005).

[7] W. KACZMAREK and J. PANASIUK: Selected Application of Industrial Robots. *Control Engineering*, **11**(104), (2013), 56-62, in Polish.

[8] K. KOZŁOWSKI, P. DUTKIEWICZ and W. WRÓBLEWSKI: Modeling and Control of Robot Manipulators. Warszawa, PWN, 2003, in Polish.

[9] Data Sheet | KR 6 R900 sixx, http://www.kuka-robotics.com/res /sps/f776ebab-f613-4818-9feb-527612db8dc4_PB0001_KR_AGILUS_en.pdf.

[10] Use And Programming Of Industrial Robots, KUKA Roboter GmbH Zugspitzstraße 140 D-86165, Augsburg Germany.

[11] A. PAYNE: A five-axis robotic motion controller for designers. *Proc. 31st Conference, Association for Computer-Aided Design in Architecture*, Banff, (2011), 162-169.

[12] T. SZKODNY: Avoiding of the kinematic singularities of contemporary industrial robots. Lecture Notes in Computer Science, Springer Int. Publ. Switzerland, **8918** (2014), 183-194.

[13] C. ZIELIŃSKI, W. SZYNKIEWICZ, T. WINIARSKI, M. STANIAK, W. CZAJEWSKI and T. KORNUTA: Rubik's cube as a benchmark validating MRROC++ as an implementation tool for service robot control systems. *Industrial Robot: An Int. Journal*, **34**(5) (2007), 368-375.