5th Int. Workshop on Robot Motion and Control, RoMoCo'05, 23–25 June 2005, Dymaczewo, Poland. pp.251–257

Applications of MRROC++ Robot Programming Framework

Cezary Zieliński Wojciech Szynkiewicz Tomasz Winiarski

C.Zielinski@ia.pw.edu.pl W.Szynkiewicz@ia.pw.edu.pl T.Winiarski@ia.pw.edu.pl
Warsaw University of Technology, Inst. of Control and Computation Engineering, Faculty of Electronics and Information Technology, ul. Nowowiejska 15/19, 00-665 Warsaw, POLAND

Abstract

The paper[1] *concentrates on the way that the* MRROC++ *robot programming framework has been applied to produce control systems for robots of different types performing diverse tasks. Moreover, both a brief formal specification and the method of implementation of the* MRROC++ *based systems is presented.*

1 Introduction

Programming frameworks [12] are application generators with the following components:

- library of software modules (building blocks out of which the system is constructed),
- a method for designing new modules that can be appended to the above mentioned library,
- a pattern according to which ready modules can be assembled into a complete system jointly exerting control over it and realizing the task at hand.

MRROC++ [31, 32] is a programming framework facilitating the implementation of single and multi-robot system controllers executing diverse tasks. Its structure was derived by formalizing the description of the general structure of multi-robot controllers [33, 34]. Its implementation was preceded by a version for single robots (RORC [30, 28]), which was later substituted by a non-object oriented version for multi-robot systems (MRROC [30, 29]). MRROC++ is derived from C++, thus it is object oriented.

Work on robot programming frameworks has a long history, but initially they were not distinguished from robot programming libraries. Robot programming frameworks have been especially favored by the research community (e.g. RCCL [9], KALI [8, 7], PASRO [2], RORC [30, 28], MRROC [30, 29], MRROC++ [31, 32], GenoM [5, 1], DCA [20], TCA [23], TDL [22], Generis [16], due to variability and diversity of research tasks. Those two factors render a specialized robot programming language either useless, if with very limited capabilities, or very similar to a general purpose programming language. Currently efforts are being made to produce public domain generic robot control software (e.g. the OROCOS project [4]).

[1] This work was supported by Polish Ministry of Science and Information Technology grant: 4 T11A 003 25.

2 General specification of MRROC++

A system composed of n_e+1 subsystems is considered. The state of such a system is described by:

$$s = \langle s_0, s_1, \ldots, s_{n_e} \rangle \tag{1}$$

where s_0 is the state of the coordinator and s_j, $j = 1, \ldots, n_e$, are the states of the subsystems. To be brief, and because of contextual obviousness, the denotations assigned to the components of the considered system and their state are not distinguished. The state of each subsystem s_j, $j = 1, \ldots, n_e$, is represented by:

$$s_j = \langle c_j, e_j, V_j, T_j \rangle \tag{2}$$

- e_j – state of the **effector** (i.e. a mechanical device capable of influencing the environment),
- c_j – state of the **subsystem controller** (memory: variables, program etc.),
- V_j – bundle of **virtual sensor** readings,
- T_j – information transmitted/received to/from the coordinator.

The **coordinator** s_0 does not posses an effector (a body), but has the ability of gathering the information from the environment, thus has a virtual sensor bundle of its own. The state of the coordinator s_0 is described by:

$$s_0 = \langle c_0, V_0 \rangle \tag{3}$$

A constraint has been imposed on the structure of the system by disallowing direct communication between subsystems. However they can still communicate indirectly, either through the coordinator or by stigmergy [3], i.e. by observing the changes in the environment caused by the other subsystems. This constraint simplifies the implementation of the inter-subsystem communication structure without significantly limiting its functionality.

A bundle of virtual sensor readings assigned to a system s_j contains n_{v_j} individual virtual sensor readings:

$$V_j = \langle v_{j_1}, \ldots, v_{j_{n_{v_j}}} \rangle \tag{4}$$

Each virtual sensor v_{j_k}, $k = 1, \ldots, n_{v_j}$, produces an aggregate reading from one or more hardware sensors – receptors. The data obtained from the receptors usually cannot be used directly in motion control, e.g. control of arm motion requires the grasping location of the object and not the bit-map delivered by a camera. In other cases a simple sensor would not suffice to control the motion (e.g. a single touch sensor), but several such sensors deliver meaningful data. The process of extracting meaningful information for the purpose of motion control is named data aggregation and is performed by virtual sensors. Thus the kth virtual sensor reading obtained by the subsystem s_j is formed as:

$$v_{j_k} = f_{v_{j_k}}(c_j, R_{j_k}) \tag{5}$$

where R_{j_k} is a bundle of receptor readings used for the creation of the kth virtual sensor reading.

$$R_{j_k} = \langle r_{j_{k_1}}, \ldots, r_{j_{k_{n_r}}} \rangle \tag{6}$$

where n_r is the number of receptor readings $r_{j_{k_l}}$, $l = 1 \ldots, n_r$, taken into account in the process of forming the reading of the kth virtual sensor of the subsystem s_j.

There are diverse methods of expressing effector state e_j, e.g. a manipulator can be perceived as:

- a set of actuators (with angular or translational shaft positions),
- a sequence of links (with angular or translational joint positions),
- an end-effector (with a coordinate frame affixed to the end-effector).

The responsibility of the controller c_j of the subsystem s_j is to: gather information about the environment through the associated virtual sensor bundle V_j, obtain the information from the coordinator s_0, monitor the state of its own effector e_j, and to process all of this information to produce: a new state of the effector e_j, influence the future functioning of the virtual sensors V_j, and communicate with the coordinator. As a side effect the internal state of the controller c_j of the subsystem s_j changes. Thus three types of components of the subsystem controller must distinguished:

- input components providing the information about: the state of the effector, virtual sensor readings and the messages obtained from the coordinator (they use a leading subscript x),
- output components exerting influence over: the state of the effector, configuration of virtual sensors and the messages to be transmitted to the coordinator (they use a leading subscript y),
- other resources needed for data processing within the subsystem controller (without a leading subscript).

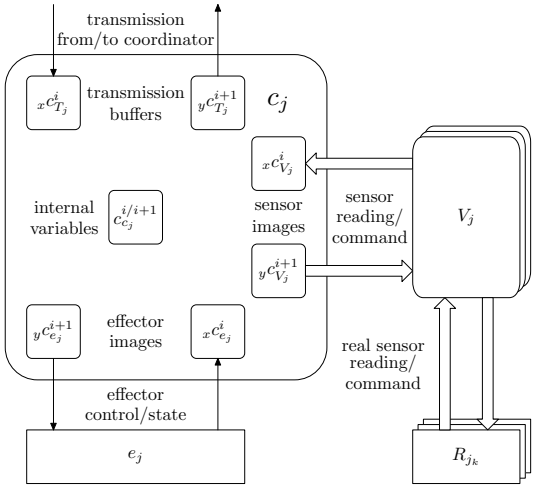


Figure 1: Subsystem s_j, $j = 1, \ldots, n_e$

The following components have been distinguished (fig.2):

- ${}_x c_{e_j}$ – input image of the effector (a perception of the effector by the subsystem controller as produced by processing the input signals transmitted from the effector to the controller, e.g. motor shaft positions, joint angles, end-effector location),
- ${}_x c_{V_j}$ – input images of the virtual sensors (current virtual sensor readings – subsystem controller's perception of the sensors and through them the environment),
- ${}_x c_{T_j}$ – input of the coordinator's commands,
- ${}_y c_{e_j}$ – output image of the effector (this is a perception of the effector by the subsystem controller as needed to produce adequate control signals),

${}_yc_{V_j}$ – output images of the virtual sensors (current configuration and commands controlling the virtual sensors),
${}_yc_{T_j}$ – output of the messages to the coordinator,
c_{c_j} – all the other relevant variables taking part in data processing within the subsystem controller.

From the point of view of the system designer the state of the subsystem controller changes at a servo sampling rate or a low multiple of that. If i denotes the current instant, the next considered instant is denoted by $i+1$. The subsystem controller uses:

$$ {}_xc_j^i = \langle c_{c_j}^i,\ {}_xc_{e_j}^i,\ {}_xc_{V_j}^i,\ {}_xc_{T_j}^i \rangle \tag{7} $$

to produce:

$$ {}_yc_j^{i+1} = \langle c_{c_j}^{i+1},\ {}_yc_{e_j}^{i+1},\ {}_yc_{V_j}^{i+1},\ {}_yc_{T_j}^{i+1} \rangle \tag{8} $$

and hence:

$$ \begin{cases} c_{c_j}^{i+1} &= f_{c_{c_j}}(c_{c_j}^i,\ {}_xc_{e_j}^i,\ {}_xc_{V_j}^i,\ {}_xc_{T_j}^i) \\ {}_yc_{e_j}^{i+1} &= f_{c_{e_j}}(c_{c_j}^i,\ {}_xc_{e_j}^i,\ {}_xc_{V_j}^i,\ {}_xc_{T_j}^i) \\ {}_yc_{V_j}^{i+1} &= f_{c_{V_j}}(c_{c_j}^i,\ {}_xc_{e_j}^i,\ {}_xc_{V_j}^i,\ {}_xc_{T_j}^i) \\ {}_yc_{T_j}^{i+1} &= f_{c_{T_j}}(c_{c_j}^i,\ {}_xc_{e_j}^i,\ {}_xc_{V_j}^i,\ {}_xc_{T_j}^i) \end{cases} \tag{9} $$

or more compactly:

$$ {}_yc_j^{i+1} = f_{c_j}({}_xc_j^i) \tag{10} $$

In the process of producing the output values ${}_yc_j^{i+1}$ the values of inputs ${}_xc_j^i$ and internal variables $c_{c_j}^i$ are used. The internal variables change their values, thus $c_{c_j}^{i+1}$ is created – those will be the values of internal variables at the onset of motion step $i+1$. All of the mentioned quantities are stored in the subsystem controller's memory, thus their values form the total state of the controller.

Delivering just one set of functions (9) for the whole duration of the subsystem's activity would be a formidable task, hence a method of decomposition is proposed. It consists in specifying the subsystem's actions in terms of motion commands, which are implemented as `Move` instructions. The semantics of those instructions is defined in fig.2. Here the single vector function (10) is decomposed into a sequence of n_m vector function triplets: ${}^mf'_{c_j}$, ${}^mf''_{c_j}$ and ${}^mf_{\tau_j}$, $m = 1, \ldots, n_m$. Each triplet is associated with a single `Move` instruction. Within each triplet the function ${}^mf'_{c_j}$ describes the first step of the `Move` instruction, whereas the function ${}^mf''_{c_j}$ describes the behaviour of the subsystem in each next step. The Boolean function ${}^mf_{\tau_j}$ determines the number of steps executed within the `Move` instruction. This number results from testing the instruction termination condition described by ${}^mf_{\tau_j}$.

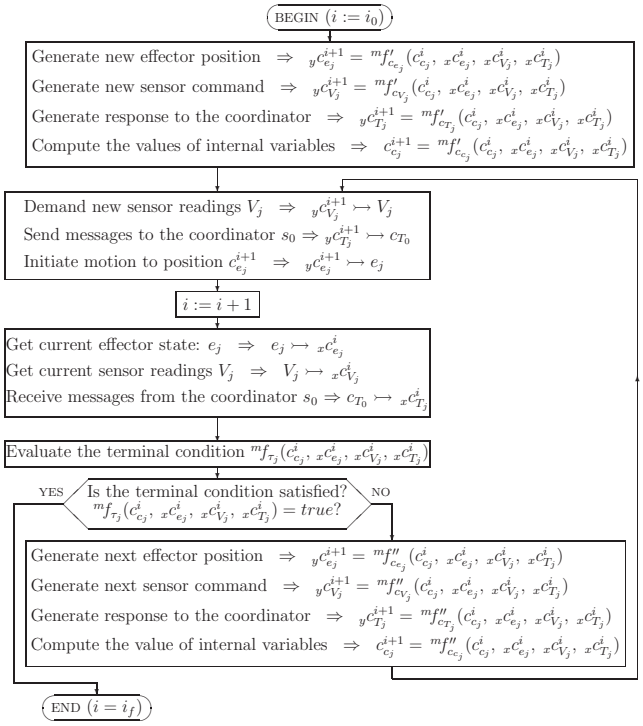


Figure 2: `Move` instruction semantics (where '$\rightarrowtail$' stands for the transfer of data)

3 Implementation

The above general specification of the system can be implemented using diverse software technologies. The most convenient way is to use the facilities (i.e. processes, threads, inter process communication) provided by real-time operating systems (RTOS) and to distinguish the coordinator and the subsystems as processes. Even smaller granularity is of benefit, thus virtual sensors also have been implemented as separate processes. Taking into account that effector control depends on both the task that is to be executed and the hardware that is to be controlled it is reasonable to separate those two functions, hence two processes have been assigned to each of the subsystems. `MRROC++` based control systems are implemented following the pattern depicted in fig.3. `MRROC++` is coded in `C++` using object-oriented programming paradigm. It currently uses QNX ver.6.3 RTOS.

The coordinator s_0 is implemented as the Master Process *MP* supplemented by zero or more Virtual Sensor Process *VSP*. Each subsystem s_j is composed of Effector Control Process *ECP*, Effector Driver Process *EDP* supplemented by zero or more Virtual Sensor Process *VSP*. Both *EDP* and *VSP*s depend on the as-

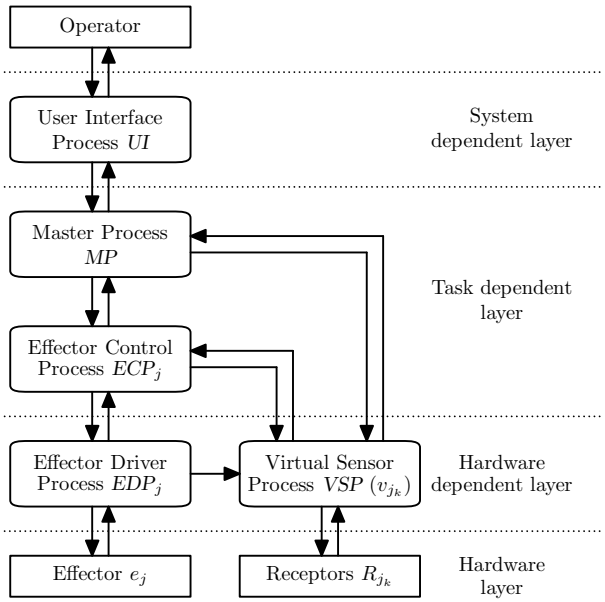


Figure 3: A MRROC++ based system structure

sociated hardware, whereas *MP* and *ECP*s are hardware independent, but depend on the task that is to be executed by the system. This division highly simplifies coding. When the hardware configuration is fixed the programmer with each new task modifies only the *MP* and the *ECP*s. Only when this configuration has to be changed, e.g. by introducing a new type of manipulator, a new *EDP* and *ECP* has to be appended to the system. The code of *ECP* contains the `Move` instructions and the definitions of vector function triplepts: ${}^{m}f'_{c_j}$, ${}^{m}f''_{c_j}$ and ${}^{m}f_{\tau_j}$ (forming a motion generator of the instruction). The code of *MP* contains similar `Move` instructions [34], but they refer to all the coordinated effectors, whereas the `Move` instructions within *ECP*s refer to single effectors. The User Interface Process *UI* depends only on the number of effectors constituting the system. It is responsible for presenting the system status to the operator and enables the operator to start, pause, resume or terminate the execution of the task.

4 Selected applications

MRROC++ has been used to create controllers for several prototype robots executing diverse tasks:

- Serial-parallel robot exhibiting high stiffness and having a large work-space [14, 18, 13, 35], thus well suited to milling (fig.4(a)) and polishing (fig.4(b)) tasks [15],
- Direct-drive robot without joint limits (fig.6(e)) [17, 19], hence applicable to fast transfer of objects,
- Two IRp-6 robots (fig.6(a)) acting as a two-handed manipulation system [37, 24]. In this case the industrial controllers of the robots were substituted by our own hardware [36].

The control programs can be simulated prior to their execution or in parallel to it [6]. Graphic simulation of the robot and its environment (fig.6(d))is run on a separate computer supervised by the Windows operating system. A separate *ECP* and *EDP* process pair is created. The *EDP* instead of issuing motor commands, causing the motions of the real robot, contacts the simulation computer by using the TCP/IP protocol. Those commands are used to animate the robot on the screen.

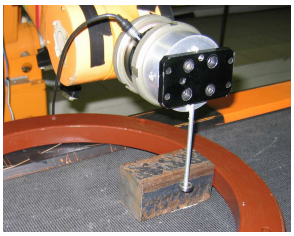
(a) RNT robot milling in wood

(b) RNT robot polishing metal

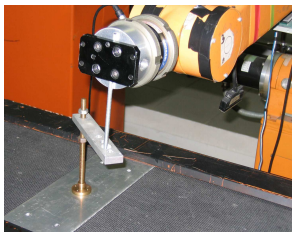
Figure 4: Industrial applications of the RNT robot with MRROC++ controllers

An IRp-6 robot mounted on a track and equipped with a force/torque sensor has been applied to the execution of three benchmark tasks [26]. The first task consisted in following an unknown contour (fig. 5(a)), the second in moving a crank 5(b) and the third in copying drawings (fig.6(b)).

Both the first and the second task use the same control algorithm, thus the *EDP* processes and the *ECP* processes are exactly the same. In both cases the end-effector moves in the horizontal plane. The direction of force command in each macrostep results from the previously measured real direction of force being applied to the surface of an object. In the case of contour following it was the surface an iron block (fig. 5(a)) and in the case of moving the crank it was the internal surface of the hole in the crank, into which the end-effector was inserted (fig. 5(b). In the orthogonal direction the end-effector was position controlled. In



(a) Following an unknown contour



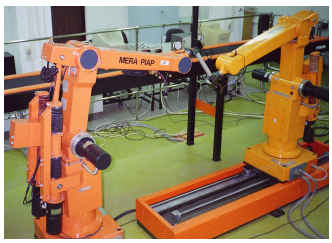
(b) Rotating a crank

Figure 5: An Irp6 robot postion–force MRROC++ controlled applications

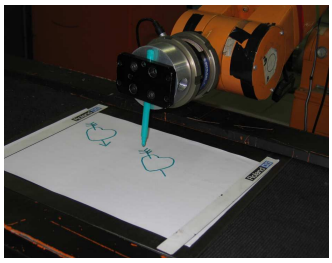
both tasks the force sensor is treated as a proprioceptor, thus its readings are utilized only in the *EDP* and the *ECP*. No *VSP* was necessary.

In drawing copying task the force sensor was used to manually guide the robot holding a pen through the motions producing a drawing. During that phase the robot was fully compliant and positions were recorded at certain intervals of time – the teach-in phase. Besides continuous force control the sensor was also used for detecting vertical jerks signalling the necessity of lifting the pen off or lowering it onto the surface of the paper to transfer it to another area of the drawing. Once the drawing was complete the pen held by the robot was displaced to a new location and the mode of operation was switched to partially compliant – reproduction phase. In that phase the robot was position controlled in the plane of the drawing and it was force controlled in the axis normal to the drawing. In that way the pen could adjust itself to the unevenness of the surface on which the drawing paper rested. In this task the force-torque sensor played a dual role, so besides being used as a proprioceptor and thus being utilized directly by the *EDP*, an extra *VSP* detecting jerks had to be added.

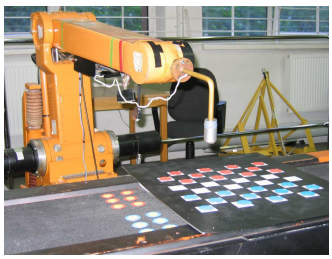
For the purpose of position–force controlled tasks, a special control algorithm was implemented in MRROC++. This algorithm is compatible with the TFF (Task Frame Formalism) specification. Hence the *ECP* process sends the macrostep command to the *EDP* process containing: `local_reference_frame` - related to either: the tool frame (relative mode) or the global frame (absolute mode), `pos_xyz_rot_xyz`, `force_xyz_torque_xyz` and `selection_vector` are vectors with six components each; the `selection_vector` determines along or about which axes of the `local_reference_frame` position/orientation is controlled and along/about which force/torque is controlled; the `pos_xyz_rot_xyz` and `force_xyz_torque_xyz` determine the set values of translation/rotation and force/torque along/about the selected axes; the translation/rotation components in the directions selected as force/torque controlled are irrelevant and vice versa, thus out of the twelve components of the two vectors (`pos_xyz_rot_xyz` and `force_xyz_torque_xyz`) only six are relevant.



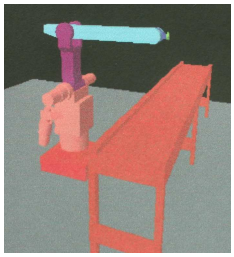
(a) The two IRp-6 robot system



(b) Drawing copying robot



(c) Draughts playing robot



(d) Simulation of an IRp-6 robot and a conveyor

(e) The Polycrank robot

Figure 6: Utilization of MRROC++ in simulation and control of diverse robot systems

Moreover, MRROC++ was used to construct a robotic system playing draughts (fig.6(c)) against a human player [25]. The overall experimental setup consisted of an IRp-6 robot with a Mitsubishi 400E CCD color camera with a lens having a focal length of $f = 3.9 - 54.9\ mm$. The images delivered by the camera were transmitted to a frame grabber at a frame rate of 25 Hz. The images contained blue or red colored pieces on white background (the checkered board was black and white). The pieces were distinguished from the environment based upon their color value. The color information was used to simplify

image analysis, i.e., object identification and extraction. One of the major problems was the effect of an ever-changing illumination, as the change of illumination changes the perceived colors. To reduce this effect the Hue-Saturation-Value (HSV) color model was used and the median non-linear filter was applied. The *VSP* on the basis of an image obtained from the camera produced a data structure representing the current state of the board. The *ECP* used this data structure to deduce the best move in terms of a certain heuristic. The α–β pruning algorithm [21] was applied to a tree of possible moves, five game-moves ahead. The best move was transformed into trajectories for the robot, represented as a sequence of `Move` instructions, and those were executed. An electromagnetic gripper was used for picking colored iron pieces. The transformation between the end-effector frame and the camera frame was only roughly known. The calibration process was performed by an operator before the game session. The force sensor was used to manually guide the end-effector to the two opposite corners of the board. Positions of the corners were recorded and used to calculate the transformation between adequate frames.

5 Conclusions

The formal description of robotic systems proved instrumental in the specification of the `MRROC++` robot programming framework. Subsequently the creation of robot controllers executing diverse tasks by using `MRROC++` turned out fairly simple. Work is being continued on ever more complex systems, which in due time will be a basis for the creation of a fully fledged service robot. Currently the abilities of hearing [10] and speaking [11] are being added. The robot will be able to comprehend Polish language commands and respond to them by a synthesized voice. Moreover, two-handed manipulation using visual and force feedback is integrated with the system.

`MRROC++` based controller is a real-time application, currently running under control of the QNX Neutrino ver.6.3 RTOS. A common problem with real-time application development is that each OS is usually equipped with its own proprietary API. However, in `MRROC++` environment POSIX-standard API is being used, and OS-specific extensions are being avoided. Therefore, `MRROC++` source code can be readily ported from QNX to another RTOS with POSIX API.

`MRROC++` controller program consists of several concurrently running multi-threaded processes. Due to modular structure together with message passing IPC those processes can be easily distributed over the local network. Then, appropriate amount of computing power can be made available at each node to meet the needs of each task performed by robotic system.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab M., and Ingrand F. An architecture for autonomy. *Int. J. of Robotics Research*, 17(4):315–337, 1998.

[2] C. Blume and W. Jakob. *PASRO: Pascal for Robots*. Springer-Verlag, Berlin, 1985.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York, Oxford, 1999.

[4] H. Bruyninckx. Orocos – open robot control software. http://www.orocos.org/, 2002.

[5] S. Fleury and M. Herrb. Genom user's guide. Report, LAAS, Toulouse, December 2001.

[6] G. Górski. Visualization of a working robot. B.Eng. Thesis, Faculty of Electronics and Information Technology, Warsaw University of Technology (in Polish), 2004.

[7] V. Hayward, L. Daneshmend, and S. Hayati. An overview of KALI: A system to program and control cooperative manipulators. In K. Waldron, editor, *Advanced Robotics*, pages 547–558. Springer-Verlag, Berlin, 1989.

[8] V. Hayward and S. Hayati. KALI: An environment for the programming and control of cooperative manipulators. In *Proc. American Control Conference*, pages 473–478. 1988.

[9] V. Hayward and R. P. Paul. Robot manipulator control under unix RCCL: A robot control C library. *Int. J. Robotics Research*, 5(4):94–111, Winter 1986.

[10] W. Kasprzak and A. Okazaki. Applying independent component analysis for speech feature detection. In *11th Int. Workshop on Systems, Signals and Image Processing IWSSIP04, Poznań, Poland*, pages 323–326. 13-15 September 2004.

[11] S. Kula, P. Dymarski, A. Janicki, C. Jobin, and P. Boula de Mareuil. Prosody control in a diphone-based speech synthesis system for polish. In *Prosody 2000 Workshop: Speech Recognition and Synthesis, Cracow*, pages 135–142. 2000.

[12] M. E. Markiewicz and C. J. P. Lucena. Object oriented framework development. *ACM Crossroads*, 7(4), 2001.

[13] K. Mianowski. Parallel and serial-parallel robots for the use of technological applications. In *Proceedings of the Parallel Kinematic Machines PKM'99, November, Milano, Italy*, pages 39–46. 1999.

[14] K. Mianowski and K. Nazarczuk. Parallel drive of manipulator arm. In *Proceedings of the 8th CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators Ro.Man.Sy 8, Cracow, Poland, 2–6 July*, pages 143–150. 1990.

[15] K. Mianowski, K. Nazarczuk, M. Wojtyra, and S. Ziętarski. Application of the unigraphics system for milling and polishing with the use of rnt robot. In *Proceedings of the Workshop for the users of UNIGRAPHICS system, Frankfurt, November*, pages 98–104. 1999.

[16] E. R. Morales. Generis: The ec-jrc generalised software control system for industrial robots. *Industrial Robot*, 26(1):26–32, 1999.

[17] K. Nazarczuk and K. Mianowski. Polycrank – fast robot without joint limits. In *Proceedings of the 12-th CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators Ro.Man.Sy'12, Vienna, 6–9 June*, pages 317–324. Springer-Verlag, 1995.

[18] K. Nazarczuk, K. Mianowski, A. Olędzki A., and C. Rzymkowski. Experimental investigation of the robot arm with serial-parallel structure. In *Proceedings of the 9-th World Congress on the Theory of Machines and Mechanisms, Milan, Italy*, pages 2112–2116. 1995.

[19] K. Nazarczuk, K. Mianowski, and S. Łuszczak. Development of the design of polycrank manipulator without joint limits. In *Proceedings of the 13-th CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators Ro.Man.Sy 13, Zakopane, Poland, 3–6 July*, pages 285–292. 2000.

[20] L. Petersson, D. Austin, and H. Christensen. Dca: A distributed control architecture for robotics. In *Proc. Int. Conference on Intelligent Robots and Systems IROS'01*. 2001.

[21] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, N.J., 1995.

[22] R. Simmons and D. Apfelbaum. A task description languagefor robot control. In *International Conference on Itelligent Robots and Systems IROS'98. Victoria, Canada*. October 1998.

[23] R. Simmons, R. Goodwin, C. Fedor J., and Basista. Task control architecture: Programmer's guide to version 8.0. Carnegie Mellon University, School of Computer Science, Robotics Institute, May 1997.

[24] W. Szynkiewicz. Motion planning for multi-robot systems with closed kinematic chains. In *Proceedings of the 9th IEEE International Conference on Methods and Models in Automation and Robotics MMAR'2003, Międzyzdroje*, pages 779–786. August 25-28 2003.

[25] P. Topór. Utilization of visual information in robot control on an example of the game of checkers. B.Eng. Thesis, Faculty of Electronics and Information Technology, Warsaw University of Technology (in Polish), 2004.

[26] T. Winiarski and C. Zieliński. Test-bed for the investigation of position–force robot control algorithms. In *8-th National Conference on Robotics, Polanica Zdrój*. June, 23–25 2004 (in Polish).

[27] C. Zieliński. TORBOL: An object level robot programming language. *Mechatronics*, 1(4):469–485, 1991.

[28] C. Zieliński. Flexible controller for robots equipped with sensors. In *9th Symp. Theory and Practice of Robots and Manipulators, Ro.Man.Sy'92, Udine, Italy, Lect. Notes: Control and Information Sciences 187*, pages 205–214. Springer-Verlag, Berlin, September 1–4, 1992 1993.

[29] C. Zieliński. Control of a multi-robot system. In *2nd Int. Symp. Methods and Models in Automation and Robotics MMAR'95, Międzyzdroje, Poland*, pages 603–608. 30 Aug.–2 Sept. 1995.

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

[31] C. Zieliński. Object–oriented programming of multi–robot systems. In *Proc. 4th Int. Symp. Methods and Models in Automation and Robotics MMAR'97, Międzyzdroje, Poland*, pages 1121–1126. August 26–29 1997.

[32] C. Zieliński. The MRROC++ system. In *1st Workshop on Robot Motion and Control, RoMoCo'99, Kiekrz, Poland*, pages 147–152. June 28–29 1999.

[33] C. Zieliński. Programming and control of multi-robot systems. In *6th International Conference on Control, Automation, Robotics and Vision, ICARCV'2000, Singapore*, pages on CD–ROM. December 5–8 2000.

[34] C. Zieliński. By how much should a general purpose programming language be extended to become a multi-robot system programming language? *Advanced Robotics*, 15(1):71–95, 2001.

[35] C. Zieliński, K. Mianowski, K. Nazarczuk, and W. Szynkiewicz. A prototype robot for polishing and milling large objects. *Industrial Robot*, 30(1):67–76, January 2003.

[36] C. Zieliński, A. Rydzewski, and W. Szynkiewicz. Multi-robot system controllers. In *Proc. of the 5th International Symposium on Methods and Models in Automation and Robotics MMAR'98, Międzyzdroje, Poland*, volume 3, pages 795–800. August 25–29 1998.

[37] C. Zieliński and W. Szynkiewicz. Control of two 5 d.o.f. robots manipulating a rigid object. In *IEEE Int. Symp. on Industrial Electronics ISIE'96, Warsaw, Poland*, volume 2, pages 979–984. June 17–20 1996.