Proc. 14th CISM-IFToMM Symp. on Robotics, RoManSy'02, 1–4 July 2002, Udine, Italy, pp.299-306.

Motion generators in MRROC++ based robot controllers

Cezary Zieliński[1]
Institute of Control and Computation Engineering, Warsaw University of Technology, Poland

Abstract. This paper presents the method of generating effector motions in `MRROC++` (Multi-Robot Research-Oriented Controller) based systems. `MRROC++` is a `C++` library of objects and processes that can be assembled into a robot controller executing a user defined task. The presented formal approach to motion generation is general in nature and not only can it be used as a basis for the implementation of any robot programming language or library, but also as a means of classification of motion generators.

1 Introduction

`MRROC++` (e.g.: Zieliński (1997), Zieliński (1999), Zieliński (2001a), Zieliński (2001b)) belongs to a family of software libraries facilitating the implementation of single- and multi-robot system controllers (e.g. `RCCL`: Hayward and Paul (1986), `Multi-RCCL`: Lloyd and Hayward (1993) `KALI`: Hayward and Hayati (1988), Hayward et al. (1989), `RORC`: Zieliński (1995a), Zieliński (1993), `MRROC`: Zieliński (1995a), `PASRO`: Blume and Jakob (1985)). Sometimes those libraries are also called robot programming languages (RPLs), because besides containing components used for constructing a robot controller they also enable the expression of robot tasks. In this case the user's task and the system controller are inseparably bound together to form a single program solving the task at hand. Those libraries consist of components (procedures or objects) that can be inserted into a general framework to create a structure executing the user's task. If the task changes, the program has to be altered, thus either the components have to be modified or reassembled in a different way. The components can be parametrised. In such a case the values of parameters are altered through external means rendering the overall modification of the controller unnecessary (Zieliński (2000)).

This paper presents a formalised approach to the implementation of library components responsible for robot motion generation. No distinction is made as to manipulators, walking machines or other vehicles – all are treated as robotic agents. The paper also prresents the general structure of `MRROC++` based controllers. Although this structure is hierarchical, by switching off the coordinator, independent functioning of robots is possible. This is especially important from the point of view of distributed systems, where no explicit coordination or direct communication between robots is anticipated (e.g. robots perceiving each other and the environment only through their own sensors – as postulated by behavioural robotics, e.g. Brooks (1991), Arkin (1998)).

Regardless of the approach followed (i.e. sense–plan–act or sense–react) the library must provide facilities for effector motion generation. This paper focuses on such specification of motion generators that any needed type of motion can be caused. As formal description based on transition functions is utilised the implementation of postulated specifications is straight forward. This is due to the fact that the concept of function is the foundation of general purpose programming languages – and those are the basis for the considered libraries.

[1] This work was supported by CATID, Warsaw University of Technology.

2 Multi-robot system decomposition

A robot system S is composed of three subsystems:

$$S =< C; E; R >, \tag{1}$$

C – **control subsystem**, (i.e. memory: variables, program etc.),
E – **effectors** (manipulator arms, tools, legs, wheels, etc.),
R – **receptors** — include all the measuring devices gathering information from the environment of the system. Devices for measuring the internal state of the system (e.g. position encoders, resolvers) are not treated here as sensors. They supply data about the state of E.
The state of the system S is denoted as:

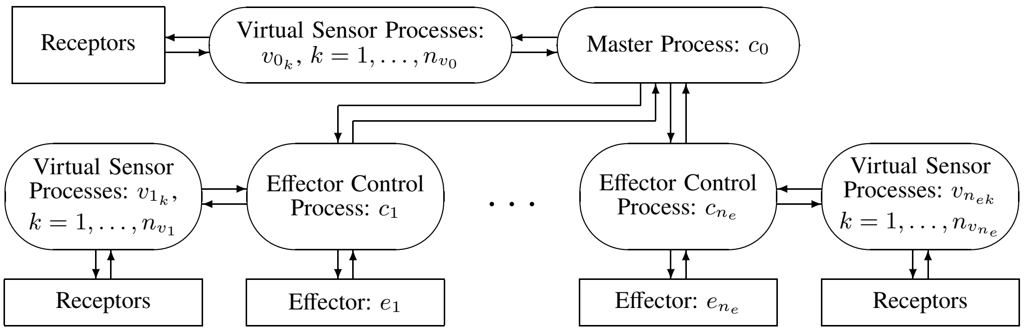
$$s =< c; e; r >, \tag{2}$$

where: c – state of the control subsystem C, e – state of the effectors E, r – state of the real (hardware) sensors R.

Motion instructions of the user's program, in the most general case, take into account the the current states of: the effectors e, the control subsystem c and the current readings r of the receptors, and exert influence over effectors E by enforcing the new desired state e. Usually the motion instructions compute the new desired locations of the effectors and the servos ensure that those locations are attained by the effectors. The computations are performed at servo sampling rate or at a low multiple of that. The state of each component of the system can be expressed in terms of diverse abstract notions. In the case of effectors these can be: joint positions, end-effector locations or grasped object Cartesian-Euler coordinates. Data obtained from real sensors usually cannot be used directly in robot motion control, e.g. to control the arm motion, only the grasping location of the object that is to be picked would be necessary – in the case of a camera a bit-map has to be processed to obtain the grasping location. In some other cases a simple sensor in its own right 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** V. As a result virtual sensor readings v are obtained.

As we are dealing with multi-effector systems, a finer granularity of decomposition is necessary. Thus the system S, as described by (1), is further divided by taking into account that there are n_e effectors $E_j, j = 1, \ldots, n_e$. For this situation a centralised control subsystem can calculate the next effector state for all of the effectors, but a much better and clearer structure is obtained, if a hierarchical distributed control subsystem is considered. In this case the control subsystem C is partitioned into $n_e + 1$ parts, where there is a single coordinator C_0 (in MRROC++ called: Master Process) and n_e effector controllers C_j (in MRROC++ called Effector Control Processes) each responsible for control of the effector E_j, $j = 1, \ldots, n_e$ (fig. 1). If the coordinator C_0 is inactive (dormant) we obtain a purely distributed control system with independent robot (effector) agents. We assume that those agents do not communicate directly – only implicit communication is possible by observing the actions of other entities through the agent's own sensors. This assumption diminishes the complexity of implementation of such a system (detailed discussion is in Zieliński (2001b)). As a system with a dormant coordinator, on the one hand, is equivalent to a system without a coordinator and, on the offer hand, is a special case of a system with a coordinator, we shall focus only on the latter case. Such a structure was assumed by the MRROC/MRROC++ based systems.

Usually virtual sensors are grouped into bundles (MRROC++ does not impose this constraint, but experience does). Each bundle is associated with a separate effector. The actions of effector



Receptors
Virtual Sensor Processes: v_{0_k}, $k = 1, \ldots, n_{v_0}$
Master Process: c_0
Virtual Sensor Processes: v_{1_k}, $k = 1, \ldots, n_{v_1}$
Effector Control Process: c_1
. . .
Effector Control Process: c_{n_e}
Virtual Sensor Processes: $v_{n_{ek}}$ $k = 1, \ldots, n_{v_{n_e}}$
Receptors
Effector: e_1
Effector: e_{n_e}
Receptors

Figure 1. Structure of a MRROC++ based system

E_k rely on n_{v_k} virtual sensors. Moreover the coordinator reads its own n_{v_0} virtual sensors. In consequence the system has n_v virtual sensors: $n_v = \sum_{k=0}^{n_e} n_{v_k}$ (fig. 1). As a result of the above (1) can be transformed into:

$$S = \ < C_0, C_1, \ldots, C_{n_e}; E_1, \ldots, E_{n_e}; V_0, \ldots, V_{n_e} > \tag{3}$$

where each V_k, $k = 0, \ldots, n_e$ represents a bundle of virtual sensors. It is composed of virtual sensors V_{k_l}, $l = 1, \ldots, n_{v_k}$. The reading of each of those sensors is represented by v_{k_l}.

The state of each component of (3) changes at sampling rate (or a low integer multiple of that – depending on the implementation). To note that, discrete time i is introduced, where i is the current time and $i+1$ is the next time instant. Moreover, in each subcontroller C_j we distinguish parts containing the information obtained from subsystems connected to this C_j. Those parts are called **images** of the connected subsystems, because this is how the subcontroller perceives the surrounding entities. The distinction of those parts is important because the information contained in them is used by the motion instructions to compute the desired states of the connected components (i.e. effectors, coordinator, virtual sensor configuration). Thus the state of each C_j can be represented by:

$$c_j^i = \ < c_{c_j}^i,\ e_{c_j}^i,\ e_{c_j}^{i+1},\ V_{c_j}^i,\ V_{c_j}^{i+1},\ a_{0j}^i,\ a_{j0}^{i+1} > \tag{4}$$

where:	$c_{c_j}^i$	– current state of the subcontroller's C_j intrinsic internal variables
	$e_{c_j}^i$	– current state of the effector E_j image
	$e_{c_j}^{i+1}$	– next (desired) state of the effector E_j image (computed in i, utilised in $i+1$)
	$V_{c_j}^i$	– current image of the readings of the virtual sensor bundle
	$V_{c_j}^{i+1}$	– desired configuration of the virtual sensor bundle (commands) (computed in i)
	a_{0j}^i	– current command of the coordinator C_0 for the subcontroller C_j
	a_{j0}^{i+1}	– response of the subcontroller C_j to the coordinator C_0 (computed in i)

Fig.2 presents the internal structure of the subcontroller C_j. In the most general case the subcontroller C_j uses the current state of: its intrinsic internal variables $c_{c_j}^i$, the effector E_j image $e_{c_j}^i$, images of readings of the virtual sensor bundle $V_{c_j}^i$ and the coordinator command a_{0j}^i, to compute: the desired state of the effector E_j (i.e. $e_{c_j}^{i+1}$), the next command for the virtual sensor bundle $V_{c_j}^{i+1}$ and a certain information for the coordinator C_0 (i.e. a_{j0}^{i+1}). Subsequently the images are passed on to the appropriate components of the system for execution or a proper

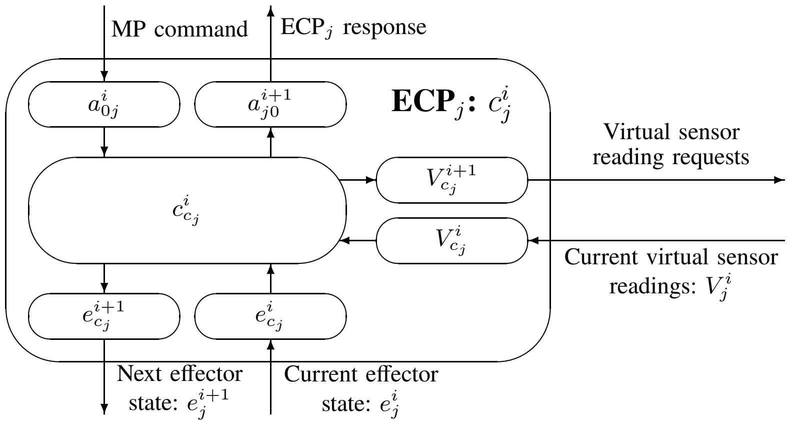

MP command
ECP_j response
a^i_{0j}
a^{i+1}_{j0}
$\mathbf{ECP}_j$: c^i_j
Virtual sensor reading requests
$V^{i+1}_{c_j}$
$c^i_{c_j}$
$V^i_{c_j}$
Current virtual sensor readings: V^i_j
$e^{i+1}_{c_j}$
$e^i_{c_j}$
Next effector state: e^{i+1}_j
Current effector state: e^i_j

Figure 2. Internal structure of subcontroller (Effector Control Process) data

reaction. Thus, $e^{i+1}_{c_j} \rightarrow e^{i+1}_j$ – due to this transfer the effector will attain the desired location (if this is impossible an error condition will occur – we shall deal with this further on). The data a^{i+1}_{j0} and $V^{i+1}_{c_j}$ will be transferred to the coordinator and each virtual sensor in the bundle, respectively. In the next control step new effector state will be obtained: $e^i_j \rightarrow e^i_{c_j}$. In reaction to the data transferred to the coordinator and the virtual sensor bundle new coordinator command a^i_{0j} will be issued and new virtual sensor readings $V^i_{c_j}$ will be acquired. Hence, in each control step the following functions are computed:

$$e^{i+1}_{c_j} = f_{e_j}(c^i_{c_j},\ e^i_{c_j},\ V^i_{c_j},\ a^i_{0j}) \tag{5}$$

$$V^{i+1}_{c_j} = f_{v_j}(c^i_{c_j},\ e^i_{c_j},\ V^i_{c_j},\ a^i_{0j}) \tag{6}$$

$$a^{i+1}_{j0} = f_{a_{j0}}(c^i_{c_j},\ e^i_{c_j},\ V^i_{c_j},\ a^i_{0j}) \tag{7}$$

$$c^{i+1}_{c_j} = f_{c_j}(c^i_{c_j},\ e^i_{c_j},\ V^i_{c_j},\ a^i_{0j}) \tag{8}$$

The evaluation of those functions is the responsibility of the motion generator. Besides the evaluation of (5), (6), (7) and (8) a decision has to be made wether the motion should be continued or terminated. This involves the computation of the **terminal condition** $f_{T_j}(c^i_{c_j}, e^i_{c_j}, V^i_{c_j}, a^i_{0j})$. This is a condition determining when the currently executed motion instruction should be terminated. Its evaluation also rests with the motion generator. All of the above computations and data transfers can be assembled into a flow chart determining their sequence in time (fig. 3). Motion generator of each motion instruction is responsible for the computations contained in the first operational block of this flow diagram and in the part enclosed by the dashed box. Using C++ terms one can say that those two areas constitute two separate methods of an object named motion generator in MRROC++. During the execution of a motion instruction all kinds of unforeseen events can take place. Those we call error conditions. As error conditions are usually dealt with by exception handling, which is asynchronous in nature, the error handling has not been included in fig. 3.

The structure and the general contents of the blocks in the flow chart do not vary with different motions. The only elements that change are the functions: (5), (6), (7) and f_{T_j}. Thus the MRROC++ programmer has to deliver just the code of those functions for each global effector

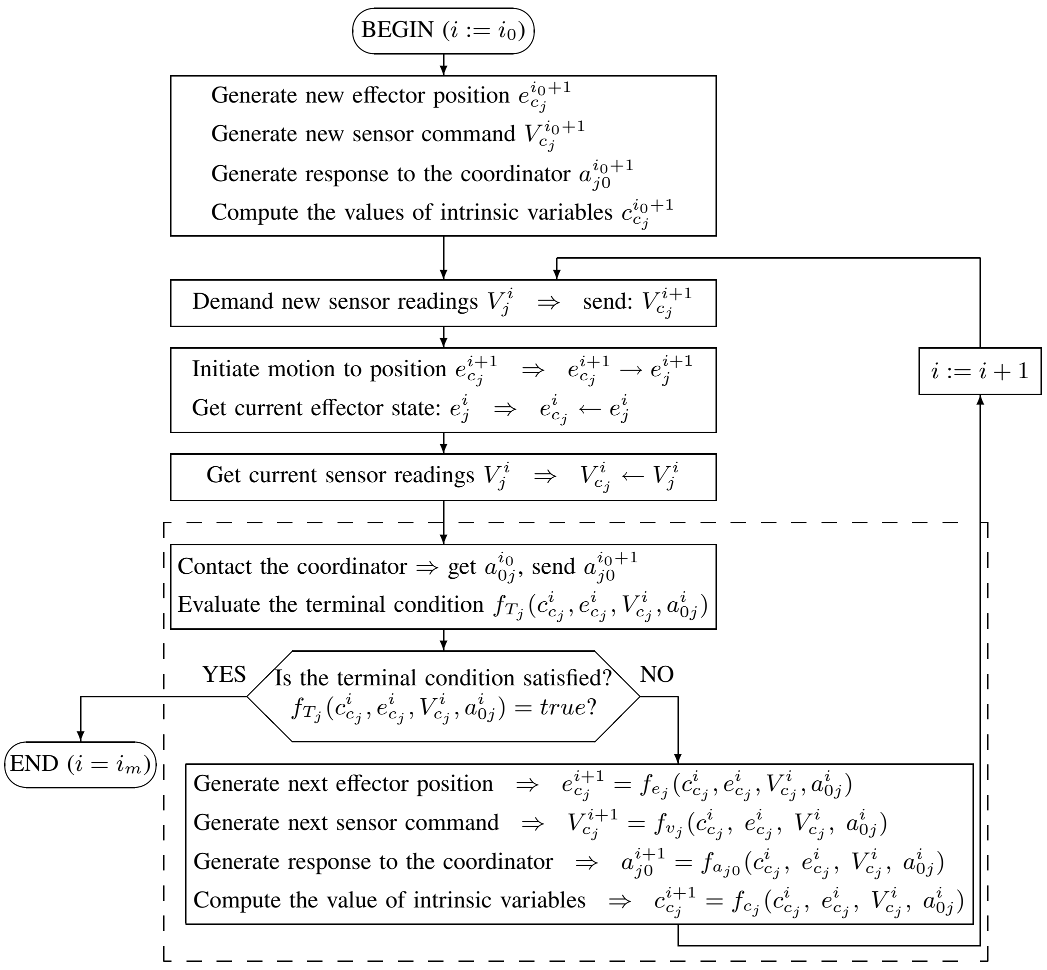

BEGIN ($i := i_0$)
Generate new effector position $e_{c_j}^{i_0+1}$
Generate new sensor command $V_{c_j}^{i_0+1}$
Generate response to the coordinator $a_{j0}^{i_0+1}$
Compute the values of intrinsic variables $c_{c_j}^{i_0+1}$
Demand new sensor readings $V_j^i \Rightarrow$ send: $V_{c_j}^{i+1}$
Initiate motion to position $e_{c_j}^{i+1} \Rightarrow e_{c_j}^{i+1} \rightarrow e_j^{i+1}$
Get current effector state: $e_j^i \Rightarrow e_{c_j}^i \leftarrow e_j^i$
Get current sensor readings $V_j^i \Rightarrow V_{c_j}^i \leftarrow V_j^i$
$i := i + 1$
Contact the coordinator $\Rightarrow$ get $a_{0j}^{i_0}$, send $a_{j0}^{i_0+1}$
Evaluate the terminal condition $f_{T_j}(c_{c_j}^i, e_{c_j}^i, V_{c_j}^i, a_{0j}^i)$
YES
Is the terminal condition satisfied?
$f_{T_j}(c_{c_j}^i, e_{c_j}^i, V_{c_j}^i, a_{0j}^i) = true$?
NO
END ($i = i_m$)
Generate next effector position $\Rightarrow e_{c_j}^{i+1} = f_{e_j}(c_{c_j}^i, e_{c_j}^i, V_{c_j}^i, a_{0j}^i)$
Generate next sensor command $\Rightarrow V_{c_j}^{i+1} = f_{v_j}(c_{c_j}^i, e_{c_j}^i, V_{c_j}^i, a_{0j}^i)$
Generate response to the coordinator $\Rightarrow a_{j0}^{i+1} = f_{a_{j0}}(c_{c_j}^i, e_{c_j}^i, V_{c_j}^i, a_{0j}^i)$
Compute the value of intrinsic variables $\Rightarrow c_{c_j}^{i+1} = f_{c_j}(c_{c_j}^i, e_{c_j}^i, V_{c_j}^i, a_{0j}^i)$

Figure 3. Motion instruction flow chart

motion – a rather simple programming task, considering that this code has to be expressed in C++.

3 MOTION GENERATORS

An interesting point is that motion generators can be classified on the basis of the arguments of the function (5), i.e. $f_{e_j}(c_{c_j}^i,\ e_{c_j}^i,\ V_{c_j}^i,\ a_{0j}^i)$. Functions (7) and (6) have the same argument lists, so it suffices to consider just (5). Function f_{e_j} has four arguments. The argument $c_{c_j}^i$ must always be present, because the intrinsic resources of C_j must be employed in the determination of the next effector state – e.g. they are used as temporary variables in the computations and reflect the current state of C_j. This produces eight possible cases $f_{e_j}^q, q = 0, \ldots, 7$ – all of them valid. When a_{0j}^i is not present in the argument list the effectors are not coordinated – no contact with the coordinator is anticipated. If a_{0j}^i is present, then the effectors are coordinated. Two forms of coordination are possible:

loose – where the coordinator synchronises (in time and space) the effectors sporadically,
tight – where effectors are synchronised in each motion step (e.g. joint transfer of an object).

In the case of loose coordination the coordinator transmits only decision information (an item from a finite set), and in the case of tight coordination numeric information (describing the next location to be attained) is being sent to the subcontrollers. It should be noted that even if the effectors are not coordinated by the coordinator they still can cooperate. In that case an effector perceives the actions of the other effectors with the sensors from its bundle. For the purpose of this paper coordination and cooperation should be distinguished.

When $e^i_{c_j}$ is present in the argument list of $f^q_{e_j}$, the effector state feedback is taken into account during motion generation. An obvious example of that is any form of interpolation between the current arm position and the desired one. To compute the absolute locations of the interpolation nodes the current arm position must be obtained by the subcontroller from lower control level (hardware). As the feedback is required only once per each trajectory section, this is the case of sporadic feedback. More frequent feedback is also possible. Effector state image $e^i_{c_j}$ can be absent from the argument list of $f^q_{e_j}$. For instance, motion generation relative to the current arm location (motion by an offset) does not require such feedback.

A similar situation arises in the case of virtual sensor readings V^i_j. If they are present in the argument list of $f^q_{e_j}$, the motion is generated on the basis of information contained in the sensor readings or this data is used to modify a predefined trajectory present in $c^i_{c_j}$. If V^i_j is missing from the argument list sensorless motion generation takes place.

It should be pointed out that proper design of such a library as `MRROC++` should enable the programmer to create all types of functions: $f^q_{e_j}, q = 0, \ldots, 7$. Otherwise motion capabilities of the system will be unnecessarily limited by the software.

4 Conclusions

`MRROC++` is submerged in `C++`. It utilises the real-time operating system QNX capable of supervising a computer network. Initially `MRROC` (Zieliński (1995b)) was implemented using procedural approach, but currently this has been changed to object-oriented approach, and hence `MRROC++` resulted. The switch of programming approach not only simplified robot task coding, but also proved to be much more effective in the implementation. Polymorphism enables late binding, so procedures could be coded without the specific knowledge of what types of effectors and sensors will be used. Exception handling enabled the separation of the code processing normal system functioning from the code dealing with error situations. Using `C++` instead of `C` functions further simplifies programming, as the former have access to all the data members of objects, while functions either have to rely on global variables or long parameter lists. The programming of such a system consists in assembling out of library objects and procedures a controller dedicated to the execution of the task at hand. The user's program is incorporated into the controller code. The user delivers the code for just a few object classes that are used by motion instructions. The formal approach presented in this paper not only facilitated the structuring of the whole system software, but also helped in distinguishing the few objects that the user has to modify while creating a user's program.

The approach followed in implementing `MRROC++` based controllers has been tested on diverse robots and tasks. `MRROC++` can currently control ASEA type IRb-6 robots (one of them mounted on a track), prototype serial-parallel structure RNT robot (Nazarczuk et al. (1995)), and a prototype fast robot without joint limits – Polycrank (Nazarczuk and Mianowski (1998)). All of those robots require specialised hardware controllers (Zieliński et al. (1998)). Force/torque, ultrasonic, and infrared sensors, CCD cameras and a conveyor belt have been included in the implemented systems. The described approach to programming has been validated on different tasks – both industrial and research.

MRROC++ has been successfully used to build a typical industrial controller for a task consisting in engraving inscriptions in soft materials (e.g. wood) by a robot equipped with a milling machine (Mianowski et al. (2000)). In this case the shape was defined to be a series of B-spline curves.

Cooperative transfer of a rigid body by two robots having 5 d.o.f. each has been demonstrated by using MRROC (Zieliński and Szynkiewicz (1996a)). To automate the tedious process of calibrating the two-robot system another controller was built. For calibration two high precision electronic theodolites were used (Frączek and Buśko (1999a)). The same procedure and software was later used in the case of the RNT robot (Frączek and Buśko (1999b)).

MRROC based software was also used to build a system containing a robot and an ultrasonic matrix overhanging a conveyor. The 3D image obtained through that matrix enabled the detection, localisation and recognition of objects moving on a conveyor. For that purpose neural networks were incorporated into the controller (Pacut et al. (1998), Brudka and Pacut (1998)).

This software can also be applied to create reactive controllers (Zieliński (1994), Zieliński (1995a), Zieliński (1995c)), which have gained much attention (e.g. Arkin (1998), Brooks (1991)). Originally a controller was built for a robot transferring inside a maze a touch probe and later a force sensor. The robot gradually gained information on its surroundings by reacting to collisions with the walls of the maze while trying to attain a global goal of finding a way out of the maze. Reactive control was also used to acquire moving objects from a conveyor. In this case infra-red sensors were the source of information both about velocity and position of the object (Zieliński (1996b)). An interesting aspect of this research was that the same formalism that was presented in this paper can be extended to describe reactive robot systems and that the hierarchical distributed controllers can be used as a platform to implement reactive control.

References

Arkin, R. C. (1998). Behavior-Based Robotics. Cambridge, MA: MIT Press.

Brooks, R. A. (1991). Inteligence Without Representation. *Artificial Intelligence* 47:139–159.

Blume, C., and Jakob, W. (1985). PASRO: Pascal for Robots. Berlin: Springer-Verlag.

Frączek, J., and Buśko, Z. (1999a). Calibration of Multi Robot System Without and Under Load Using Electronic Theodolites. In *Proc. First Workshop on Robot Motion and Control RoMoCo'99*. Kiekrz, Poland. 71-75.

Frączek, J., and Buśko, Z. (1999b). Calibration of Serial and Serial-Parallel Robot Systems Using Electronic Theodolites and Error Correction Procedure. In *Proc. 10th World Congress on the Theory of Machines and Mechanisms*. Oulu, Finland. 972–977.

Hayward, V., and Paul, R. P. (1986). Robot Manipulator Control Under Unix RCCL: A Robot Control C Library. *Int. J. Robotics Research* 5(4):94-111.

Hayward, V., and Hayati, S. (1988). KALI: An Environment for the Programming and Control of Cooperative Manipulators. In *Proc. American Control Conference*. USA. 473-478.

Hayward, V., Daneshmend, L., and Hayati, S. (1989). An Overview of KALI: A System to Program and Control Cooperative Manipulators. In Waldron, K., ed., *Advanced Robotics*. Springer-Verlag. 547–558.

Lloyd, J., and Hayward, V. (1993). Real-Time Trajectory Generation in Multi-RCCL. *Journal of Robotics Systems* 10(3): 369–390.

Mianowski, K., Nazarczuk, K., Wojtyra, M., Szynkiewicz, W., Zieliński, C., and Woźniak, A.: (2000). Application of the RNT Robot to Milling and Polishing. In *Proc. Thirteenth CISM-IFToMM Symposium Theory and Practice of Robots and Manipulators Ro.Man.Sy'13*. Zakopane, Poland. 421–429.

Nazarczuk, K., Mianowski, K., Olędzki, A., and Rzymkowski, C. (1995). Experimental Investigation of the Robot Arm with Serial-Parallel Structure. In *Proc. Ninth World Congress on the Theory of Machines and Mechanisms*. Milan, Italy. 2112-2116.

Nazarczuk, K., and Mianowski, K. (1998). Polycrank – Fast Robot Without Joint Limits. In *Proc. Twelfth CISM-IFToMM Symp. Theory and Practice of Robots and Manipulators Ro.Man.Sy'12*. Paris, France. 317-324.

Pacut, A., Brudka, M., and Jaworski, M. (1998). Neural Processing of Ultrasound Images in Robotic Applications. In *Proc. IEEE Int. Workshop on Emerging Technologies, Intelligent Measurements and Virtual Systems for Instrumentation and Measurements ETIMVIS'98*. St. Paul, USA. 59–66.

Brudka, M., and Pacut, A. (1999). Intelligent Robot Control Using Ultrasonic Measurements. In *Proc. Sixteenth IEEE Instrumentation and Measurement Technology Conference IMTC'99*. Venice, Italy. 727–732.

Zieliński, C. (1993). Flexible Controller for Robots Equipped with Sensors. In *Proc. Ninth Symp. Theory and Practice of Robots and Manipulators, Ro.Man.Sy'92.*, Udine, Italy. 205-214.

Zieliński, C. (1994). Reaction Based Robot Control. *Mechatronics* 4(8): 843–860.

Zieliński, C. (1995). Robot Programming Methods. Warsaw, Publishing House of Warsaw University of Technology.

Zieliński, C. (1995). Control of a Multi-Robot System. In *Proc. Second Int. Symp. Methods and Models in Automation and Robotics MMAR'95*, Międzyzdroje, Poland. 603-608.

Zieliński, C. (1995). Sensorimotor robot control. In *Proc. Seventh IFAC/IFORS/IMACS Symp. Large Scale Systems: Theory and Applications*, London, United Kingdom. 797–802.

Zieliński, C., and Szynkiewicz, W. (1996). Control of Two 5 d.o.f. Robots Manipulating a Rigid Object. In *Proc. IEEE Int. Symp. on Industrial Electronics ISIE'96*, Warsaw, Poland. 979–984.

Zieliński, C. (1996). Control of a Multi-Robot System. In *Proc. Third Int. Symp. Methods and Models in Automation and Robotics MMAR'96*, Międzyzdroje, Poland. 893–898.

Zieliński, C. (1997). Control of a Multi-Robot System. In *Proc. Fourth Int. Symp. Methods and Models in Automation and Robotics MMAR'97*, Międzyzdroje, Poland. 1121–1126.

Zieliński, C., Rydzewski, A., and Szynkiewicz, W. (1998). Multi-Robot System Controllers. In *Proc. Fifth Int. Symp. Methods and Models in Automation and Robotics MMAR'98*, Międzyzdroje, Poland. 795–800.

Zieliński, C. (1999). The MRROC++ System. In *Proc. First Workshop on Robot Motion and Control, RoMoCo'99*, Kiekrz, Poland. 147–152.

Zieliński, C. (2000). Implementation of Control Systems for Autonomous Robots. In *Proc. Sixth International Conference on Control, Automation, Robotics and Vision, ICARCV'2000*, Singapore. On CD-ROM.

Zieliński, C. (2001). 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.

Zieliński, C. (2001). A Quasi-Formal Approach to Structuring Multi-Robot System Controllers. In *Proc. 2nd International Workshop on Robot Motion and Control, RoMoCo'01*, Bukowy Dworek, Poland. 121–128.