BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 58, No. 1, 2010
DOI: 10.2478/v10175-010-0002-x

General specification of multi-robot control system structures

C. ZIELIŃSKI* and T. WINIARSKI

Institute of Control and Computation Engineering, Warsaw University of Technology, 15/17 Nowowiejska St., 00–665 Warsaw, Poland

Abstract. The paper deals with structuring robot control systems. The control system is decomposed into distinct agents. An agent, in general, is responsible for control of its effector, perception of the environment for the purpose of its effector control, and inter-agent communication. The behaviour of the agent is governed by its set of transition functions. The control system consists of two tiers – the upper tier is defined by the flow of information between the agents and the lower tier is defined by formal specification of each agent's behaviour (influence on the environment, gathering sensor readings, production and consumption of the information for/from the other agents). The paper presents one of the examples of utilization of this approach. The example concerns the multi-robot drawing copying system.

Key words: robot control systems, specification of controllers, multi-robot systems.

1. Introduction

Robot control systems are usually very complex, and multi-robot system controllers are even more so. The discussion of the structures of those controllers and the way they operate requires an adequate formal language based on mathematics. There have been some attempts to formalize the subject in general (e.g., [1–3]), however, the majority of the work has concentrated on software engineering approaches to robotics [4–6], especially with the focus on robot programming frameworks [7] (e.g.: RCCL [8], KALI [9, 10], PASRO [11, 12], RORC [13, 14], MRROC [13, 15], MRROC++ [16, 17], G^{en}oM [18, 19], DCA [20], TCA [21], TDL [22], Generis [23], OROCOS [24, 25], CoolBOT [3, 26], ORCA [27, 28], Player [29–31]).

The contemporary robot system controllers are predominantly computer based. Computers are programmed using programming languages, hence they are treated as automatons accepting programs coded in those programming languages. The description of controller structure necessitates the expression of operations that the controller performs, i.e., requires the definition of semantics of those operations. For the discussion to be precise the semantics of those operations must be stated formally. It should be noted that the majority of control systems has their structure defined only on the basis of the designer's experience. However, rational choice requires the evaluation of many conflicting criteria. Some of the usual questions that the designer must answer are:

- Into how many subsystems should the system be decomposed? (small number of subsystems makes them complex, while a large number imposes heavy communication requirements and makes the synchronization of the operation of subsystems more difficult),
- What should be the individual role of each subsystem?
- What information must be provided for each of the subsystems and thus what information must be exchanged between those subsystems?
- How to ensure future extensibility of the system?

To answer this type of questions the designer must have at his or her disposal a tool for formulating and evaluating decisions. Such a tool is a language for formal expression of: structure of the system, its decomposition into subsystems, definition of functions of each subsystem, communication between subsystems, evaluation of latencies introduced by each component etc. This paper provides a proposal of such a formal language. In its first part the language is described, while in its second part this language is utilized to specify the structure and operation of a drawing reproducing system controller. Unfortunately, for the lack of space, the full discussion of the considered design possibilities cannot be presented, so only the final outcome of this discussion is revealed.

The problem of describing how software based systems function is not new. Description of programming language semantics has its long history [32]. The many specific methods of defining semantics in principle can be categorized into three groups: operational semantics, denotational semantics, axiomatic (logic) semantics.

Operational semantics requires the definition of an abstract machine that accepts the instructions of the considered language. Usually this machine is capable of executing very elementary operations, that are well defined mathematically. Complex operations are defined in terms of elementary ones. Denotational semantics assigns mathematical objects (denotations) to expressions of the defined language. Those denotations describe what those expressions mean. The expressions of the original language are translated into the language of

*e-mail: C.Zielinski@ia.pw.edu.pl

denotations. Axiomatic semantics defines logical expressions that define the meaning of an expression of the language. One useful form of this approach defines the initial conditions for the execution of an instruction. If those conditions are fulfilled, as the result of the execution of this instruction terminal conditions are ascertained. The logical formula connects the initial and terminal conditions.

The denotational approach is translation based – its foundation is mathematical transformation. In producing a tool for the specification of robot control systems one should take into account the achievements of computer science, nevertheless, the fundamental difference between computers and robots should be kept in mind. The model of a computer is well defined and basically deterministic, while robots interact with the environment, which is modeled only approximately and cannot be treated as fully deterministic. Moreover, computers, as their name suggests, are principally used for computations, while robots are used for transforming the environment or reacting to events occurring in it. All this makes the denotational approach less attractive to our purpose. However, both the operational and axiomatic approach can be utilized to a certain extent. The choice of one of the two depends on the goal of the specification. If the aim is the definition of the control system structure and operation, operational approach is more attractive. If the goal is the definition of services provided by the control system to the user, then axiomatic approach is more relevant, as it rids itself from unnecessary details. Nevertheless, one should take into account that the provided services must be implemented, thus the axiomatic approach still has to be redefined in terms of operations of the controller, i.e., in terms of operational semantics. Hence the approach presented in this paper is inspired by operational semantics, which is more fundamental, although more detailed than the axiomatic approach. Although this paper advocates the operational approach, because it is more relevant to the purpose of structuring robot control systems (what is at the focus of this discussion), it does not reject the usefulness of the axiomatic approach favored by those who are interested in the services provided by the system (e.g., SOA architectures [33]).

The discussion of the structures of robot control systems will be based on the concept of agent. The agents having physical bodies (e.g., robots) will be termed embodied agents. Both the operation of a single agent and the interactions between agents is of interest to us. The discussion is based on the general formalism presented in [34, 35].

Initially the necessary concepts are introduced, and subsequently an example of application of those concepts to the specification of a two-robot control system is presented.

2. An embodied agent

A multi-robot system composed of n_a agents a_j, $j = 0, \ldots, n_a - 1$, is considered. The internal structure of each agent a_j is presented in Fig. 1. Four distinct entities are distinguished:

- e_j – effector, i.e., a device responsible for influencing the environment (its state is obtained by reading proprioceptors), including its control hardware,
- R_j – receptors, i.e., devices gathering the information about the state of the environment (external to the agent) – subsequently processed to produce virtual sensor readings V_j (usually this information is gathered by exteroceptors, however in some cases proprioceptors can be used to detect indirectly the changes occurring in the environment, so both kinds of receptors can be a source of data for aggregation by the virtual sensors),
- T_j – transmission links, which are responsible for direct interchange of data between the considered agent a_j and the other agents,
- c_j – control subsystem – enforces a certain behaviour of the agent a_j.

In this paper the symbols representing system components and their state are not differentiated, because they pertain to the same entity and context makes this differentiation obvious, whilst significantly reducing the number of symbols used.

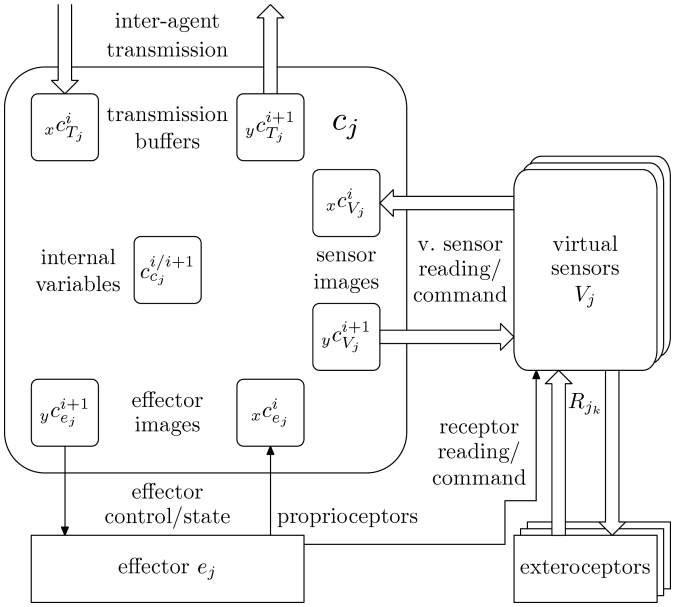

Fig. 1. General structure of an embodied agent

The data obtained from the exteroceptors usually cannot be used directly in motion control, e.g., control of a manipulator requires the goal location and not the bit-map delivered by a camera. In other cases a simple sensor will not suffice to control the motion (e.g., a single proximity sensor), but several such sensors deliver meaningful data about the surrounding obstacles. 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 agent a_j is formed as:

$$v_{j_k} = f_{v_{j_k}}(c_j, v_{c_{j_k}}, R_{j_k}). \tag{1}$$

As the exteroceptors may have to be prompted or configured, c_j is one of the arguments of the aggregating function (1).

Moreover, a virtual sensor sometimes has its internal memory $v_{c_{j_k}}$ – this is equivalent to sensoric memory in animals. Its contents is formed by an auxiliary function:

$$v_{c_{j_k}} = f_{vc_{j_k}}(c_j, v_{c_{j_k}}, R_{j_k}). \tag{2}$$

Obviously the $v_{c_{j_k}}$ being the argument of the function $f_{vc_{j_k}}$ is different from the $v_{c_{j_k}}$ being the computed value of this function (the one on the left hand side of the equals sign). The former is the contents of the sensoric memory before the computation of the value of this function and the latter after the computations have been completed. Further on in the paper such distinction will be made obvious by adding a superscript representing a time stamp.

A bundle of receptors R_{j_k}, used for the creation of the kth virtual sensor reading, consists of n_r individual receptor readings:

$$R_{j_k} = \langle r_{j_{k_1}}, \ldots, r_{j_{k_{n_r}}} \rangle, \tag{3}$$

where $r_{j_{k_l}}$, $l = 1 \ldots, n_r$, are the individual receptors taken into account in the process of forming the reading of the kth virtual sensor of the agent a_j.

The virtual sensor bundle 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 receptors, as described by (1) and (3). Each agent a_j forms and uses its own bundle v_j of virtual sensors.

The first three of the four entities listed above as components of an agent a_j (i.e., e_j, V_j, T_j) are represented in its control subsystem as images. Those images (data structures) contain parameters of the models of those components. The programmer perceives those components through those data structures, thus their names – images. The input images contain the information produced by the component for the control subsystem (denoted by a leading subscript x) and the output images contain the information produced by the controller for the component to utilise (subscript y). Diverse images (views, models) of the physical devices can be envisaged, thus creating different ontologies (An ontology in computer science is a formal representation a domain of concepts and the relationships between them – the images represent those concepts.). The control subsystem c_j of the agent a_j besides the above mentioned three entities contains its own internal data structures, thus the following components exist within it:

- ${}_xc_{e_j}$ – input image of the effector (a set of data conforming to the assumed input model of the effector in the control subsystem – it is produced by processing the input signals transmitted from the effector proprioceptors to the control subsystem, e.g., motor shaft positions, joint angles, end-effector location – they form diverse ontologies),
- ${}_xc_{V_j}$ – input images of the virtual sensors (current virtual sensor readings – control subsystem's perception of the sensors and through them of the environment),
- ${}_xc_{T_j}$ – input of the inter-agent transmission (information obtained from other agents),
- ${}_yc_{e_j}$ – output image of the effector (a set of data conforming to the assumed output model of the effector in the control subsystem – e.g., PWM ratios supplied to the motor drivers; thus the input and output models of the effector need not be the same – and usually are not),
- ${}_yc_{V_j}$ – output images of the virtual sensors (current configuration and commands controlling the virtual sensors),
- ${}_yc_{T_j}$ – output of the inter-agent transmission (information transmitted to the other agents),
- c_{c_j} – all of the other relevant variables taking part in data processing within the agent's control subsystem.

3. General structure of images

The state of the internal data structures c_{c_j} is represented by a structure containing n_{ccj} variables:

$$c_{c_j} = \langle c_{c_{j[1]}}, \ldots, c_{c_{j[n_{ccj}]}} \rangle. \tag{5}$$

Analogically input effector image ${}_xc_{e_j}$ consists of n_{exj} variables:

$${}_xc_{e_j} = \langle {}_xc_{e_{j[1]}}, \ldots, {}_xc_{e_{j[n_{exj}]}} \rangle. \tag{6}$$

The input virtual sensor image ${}_xc_{V_j}$ contains n_{Vxj} individual sensor readings:

$${}_xc_{V_j} = \langle {}_xc_{V_{j_1}}, \ldots, {}_xc_{V_{j_{n_{Vxj}}}} \rangle, \tag{7}$$

where each of those readings has the following structure:

$${}_xc_{V_{j_k}} = \langle {}_xc_{V_{j_k[1]}}, \ldots, {}_xc_{V_{j_k[n_{Vxjk}]}} \rangle. \tag{8}$$

Each input transmission buffer ${}_xc_{T_{jj'}}$ consists of $n_{Txjj'}$ variables:

$${}_xc_{T_{jj'}} = \langle {}_xc_{T_{jj'[1]}}, \ldots, {}_xc_{T_{jj'[n_{Txjj'}]}} \rangle. \tag{9}$$

The transmitters c_{T_j} of agent a_j have received a more detailed description denoting both the owner of the transmission buffer (the first right subscript after T – this is the original subscript used by the one subscript version) and the source/destination of the information (the trailing right subscript), e.g. $c_{T_{jj'}}$ is composed of the transmission buffer of agent a_j receiving information from agent $a_{j'}$: ${}_xc_{T_{jj'}}$, or sending information to $a_{j'}$: ${}_yc_{T_{jj'}}$. The agent a_j contains as many such input transmission buffers as there are direct connections with other agents $a_{j'}$.

Generally each input transmission image of agent a_j corresponds to the output transmission image of agent $a_{j'}$ and vice versa:

$${}_xc_{T_{jj'}} - {}_yc_{T_{j'j}}, \quad {}_yc_{T_{jj'}} - {}_xc_{T_{j'j}}. \tag{10}$$

The output effector image ${}_yc_{e_j}$ consists of n_{eyj} variables:

$$ {}_yc_{e_j} = \langle {}_yc_{e_{j[1]}}, \ldots, {}_yc_{e_{j[n_{eyj}]}} \rangle. \tag{11} $$

The output virtual sensor image ${}_yc_{V_j}$ contains n_{Vyj} individual sensor commands:

$$ {}_yc_{V_j} = \langle {}_yc_{V_{j_1}}, \ldots, {}_yc_{V_{j_{n_{Vyj}}}} \rangle, \tag{12} $$

where each of those commands has the following structure:

$$ {}_yc_{V_{j_k}} = \langle {}_yc_{V_{j_k[1]}}, \ldots, {}_yc_{V_{j_k[n_{Vyjk}]}} \rangle. \tag{13} $$

Output transmission buffer ${}_yc_{T_{jj'}}$ consists of $n_{T_{yjj'}}$ variables:

$$ {}_yc_{T_{jj'}} = \langle {}_yc_{T_{jj'[1]}}, \ldots, {}_yc_{T_{jj'[n_{T_{yjj'}}]}} \rangle. \tag{14} $$

Each input transmission image of agent a_j corresponds to the output transmission image of agent $a_{j'}$ and vice versa. Some types of agents do not have all of the images enumerated by (5)–(14). In general the lack of an image is equivalent to the respective number n being equal to 0.

4. Transition functions

The operation of the control system of an agent can be expressed by specifying the relationship between the input and output images. This relationship is defined in terms of transition functions. From the point of view of the system designer the state of the control subsystem 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 $i+1$. This will be called a motion macrostep. The control subsystem uses:

$$ {}_xc_j^i = \langle c_{c_j}^i, {}_xc_{e_j}^i, {}_xc_{V_j}^i, {}_xc_{T_j}^i \rangle, \tag{15} $$

to produce:

$$ {}_yc_j^{i+1} = \langle c_{c_j}^{i+1}, {}_yc_{e_j}^{i+1}, {}_yc_{V_j}^{i+1}, {}_yc_{T_j}^{i+1} \rangle. \tag{16} $$

For that purpose it uses transition functions:

$$ \begin{cases} c_{c_j}^{i+1} &= f_{c_{c_j}}(c_{c_j}^i, {}_xc_{e_j}^i, {}_xc_{V_j}^i, {}_xc_{T_j}^i) \\ {}_yc_{e_j}^{i+1} &= f_{c_{e_j}}(c_{c_j}^i, {}_xc_{e_j}^i, {}_xc_{V_j}^i, {}_xc_{T_j}^i) \\ {}_yc_{V_j}^{i+1} &= f_{c_{V_j}}(c_{c_j}^i, {}_xc_{e_j}^i, {}_xc_{V_j}^i, {}_xc_{T_j}^i) \\ {}_yc_{T_j}^{i+1} &= f_{c_{T_j}}(c_{c_j}^i, {}_xc_{e_j}^i, {}_xc_{V_j}^i, {}_xc_{T_j}^i) \end{cases}. \tag{17} $$

This can be written down more compactly as:

$$ {}_yc_j^{i+1} = f_{c_j}({}_xc_j^i). \tag{18} $$

Formula (18) is a prescription for evolving the state of the system, thus it has to be treated as a program of the agent's behaviour. For any agent exhibiting useful behaviours this function would be very complex, because it describes the actions of the system throughout its existence. The complexity of this function renders impractical the representation of the program of agent's actions as a single function. Function (18) has to be decomposed to make the specification of the agent's program of actions comprehensible and uncomplicated. However, this implies that there will be many partial functions that need to be selected and composed to produce the program of the agent's actions. Both selection and composition must be defined formally. Usually selection is based on predicates and composition is based on concatenation or superposition [34]. Hence, instead of a single transition function f_{c_j}, n_f partial transition functions are defined:

$$ {}_yc_j^{i+1} = {}^mf_{c_j}({}_xc_j^i), \qquad m = 1, \ldots, n_f. \tag{19} $$

Variability of agents is due to the diversity of those partial transition functions and their different compositions. An in-depth discussion of the possible decompositions is presented in [34].

Each such function governs the operation of the agent for some time. Usually this time is not defined explicitly. There are some external to the agent factors that necessitate the switch of the partial transition function. Such events are detected by a Boolean valued function (a predicate) called the terminal condition. Thus each partial transition function ${}^mf_{c_j}$ is decomposed into two sub-functions: ${}^mf_{\tau_j}$ and ${}^mf'_{c_j}$. The former expresses the terminal condition – its fulfilment stops the repetition of computations of the latter function, i.e., the function responsible for the behaviour of the system within each period $i \to i+1$. This is the foundation of the general motion instruction, which governs the activities of an agent for the duration of the validity of transition function ${}^mf'_{c_j}$.

5. Motion instruction

A motion instruction of each embodied agent a_j requires the input of all arguments ${}_xc_j^i$, testing of the terminal condition ${}^mf_{\tau_j}$, and if it is not true, the computation of the next desired values ${}_yc_j^{i+1}$, which in turn have to be dispatched to the appropriate components of the system. Its general form is as follows:

```
loop
 // Check the terminal condition
 if ᵐf_τj( ₓc_j^i ) = false then
  // Compute the next control subsystem state
   y c_j^{i+1} := ᵐf'_cj( ₓc_j^i );
  // Transmit the results
  y c_ej^{i+1} ⇀ e_j;   y c_Vj^{i+1} ⇀ V_j;   y c_Tjj'^{i+1} ⇀ₓ c_Tj'j;
  // Wait for the next iteration
  i := i + 1;
  // Determine the current state of the agent
  e_j ⇀ ₓc_ej^i;   V_j ⇀ ₓc_Vj^i;   y c_Tj'j ⇀ ₓc_Tjj'^i;
 endif
endloop
```

(20)

where $\rightharpoonup$ represents transfer of data. The motion instruction starts with the test of the terminal condition, so it is assumed that prior to the initiation of the current motion instruction all the necessary data has been read-in by the control susbsystem. Hence the motion instruction terminates with this data being read-in. At system initiation this data is also input.

6. Elementary behaviours

Code (20) defines the agent's partial behaviour. Due to enormous multiplicity of possible transition functions (18) (i.e., ${}^{m}f'_{c_{e_j}}$, ${}^{m}f_{\tau_j}$ pairs) there is no limit to the definition of those behaviours. Thus the programmer has to be supported with some guidance to facilitate the creation of useful systems. The main purpose of functions ${}^{m}f'_{c_j}$ is to induce motion of the effectors. The state of the effectors is measured by proprioceptors. Thus elementary behaviours used for the creation of partial behaviours (20) are based on propropceptive input (e.g., in the case of a manipulator its position and the generalized force it exerts on the environment). Fortunately all elementary behaviours of a manipulator fall into three general categories. Those categories have been singled out by inspecting possible behaviours of the effector in very diverse tasks. The experience gained trough the creation of both industrial and service robot controllers executing considerably differing tasks implemented by using the MRROC++ robot programming framework [36], which was specified in terms of the concepts introduced in this paper, showed that the following general behaviours are necessary:

- unconstrained motion with the assumption that no contact with obstacles will be encountered – here pure position control suffices,
- contact with the environment – here pure force control is used,
- intermediate or transitional behaviour – here initially unconstrained motion is expected to result in eventual contact, or vice versa – for this purpose some form of parallel position–force control has to be utilized (e.g., stiffness [37], damping [38] or impedance control [39]).

It should be noted that sometimes simultaneously one form of those behaviours is expected to occur in one spatial direction, whereas another form has to be realized in another.

The three enumerated elementary behaviours are used as building blocks for constructing more elaborate functions f'_{c_j}, which take into account the data obtained from virtual sensors and other agents, as presented by (20). The functions ${}^{m}f'_{c_j}$ produce values that are the arguments of elementary behaviours executed in the process of transmitting the results (execution of the $\rightarrowtail$ operator in code (20)).

7. Effector driver

The output effector image stores the data necessary for the computation of the control law governing the behaviour of the effector, i.e., the manipulator in this specific case. The agent's control system forms commands for the Effector Driver. Each transmission of the output image to its respective component of the agent defines the behaviour of that component during the next macrostep, so it also defines the behaviour of the manipulator by delivering the parameters to the control law implemented in the effector driver. Each macrostep is divided into steps internally by the effector driver. The operation of this driver within each step $\iota \rightarrow \iota + 1$ is described by the following control law, which is formulated for each direction of motion separately, analogically to the Task Frame Formalism [40] or Operational Space concept [41]:

$$ {}^{E^{\iota}}\dot{r}^{\iota+1}_{(A)c[l]} = \frac{\left(\mathfrak{B}_{[l]}\left({}^{E^{\iota}}\mathcal{F}^{i+1}_{d[l]} - {}^{E^{\iota}}\mathcal{F}^{\iota}_{a[l]}\right) + {}^{E^{\iota}}_{E}\dot{r}^{i+1}_{(A)d[l]}\right)\Delta t}{\Delta t + \mathfrak{B}_{[l]}\, I_{[l]}} + $$
$$ + \frac{\mathfrak{B}_{[l]}\, I_{[l]}\ {}^{E^{\iota}}\dot{r}^{\iota}_{(A)c[l]}}{\Delta t + \mathfrak{B}_{[l]}\, I_{[l]}}, \tag{21} $$

where E – the frame affixed to the end-effector (tool), E^{ι} – the superscript denotes the fact that a certain value is expressed with respect to a frame with an orientation of the frame E at instant ι, (A) – the subscript indicates that a certain quantity is expressed in XYZ Cartesian coordinates supplemented by an angle and axis representation of orientation, ${}^{E^{\iota}}\dot{r}^{\iota+1}_{(A)c}$ – the computed generalized velocity of the end-effector in relation to the world coordinate frame for the next step $(\iota + 1)$ expressed with respect to E^{ι}, ${}^{E^{\iota}}_{E}\dot{r}^{i+1}_{(A)d}$ – the desired generalized velocity of the end-effector (set for the whole of the macrostep) in relation to the world coordinate frame for the next step $(\iota + 1)$ expressed with respect to E^{ι}, ${}^{E^{\iota}}\mathcal{F}^{i+1}_{d}$ – desired general force for time instant $\iota + 1$ expressed with respect to E^{ι}, ${}^{E^{\iota}}\mathcal{F}^{\iota}_{a}$ – measured general force at time instant ι expressed with respect to E^{ι}, $\mathfrak{B}$ – desired value of reciprocal of damping, I – desired value of inertia, Δt – duration of a single step $(\iota \rightarrow \iota + 1)$, l – right subscript part in square brackets denotes a coordinate of a vector. The vector components are referred to by x, y, z (linear coordinates) and a_x, a_y, a_z (angular coordinates).

Each of the three elementary behaviours is obtained by assigning specific values to the parameters of the control law (21), i.e.:

- UNGUARDED – $\mathfrak{B}_{[l]}$ is set to zero (damping becomes infinite for the force portion of the control law, i.e. force does not cause any displacement),
- CONTACT – the desired velocity ${}^{E^{\iota}}_{E}\dot{r}^{i+1}_{(A)d[l]}$ becomes zero,
- GUARDED – the desired force ${}^{E^{\iota}}\mathcal{F}^{i+1}_{d[l]}$ is set to zero.

The computed velocity ${}^{E^{\iota}}\dot{r}^{\iota+1}_{(A)c}$ is transformed into the desired step increment ${}^{E^{\iota}}_{E^{\iota+1}}r_{(A)c}$:

$$ {}^{E^{\iota}}_{E^{\iota+1}}r_{(A)c} = {}^{E^{\iota}}\dot{r}^{\iota+1}_{(A)c}\Delta t. \tag{22} $$

This is executed by the position axis-controller after transformation by the inverse kinematics procedure. A detailed presentation of the driver is contained in [42].

8. Example: copying drawings by a multi–robot system

The utilization of the above mentioned formal considerations will be presented here on an example of the specification of a controller for a robot system reproducing the taught–in drawings. Both the teach–in phase and the reproduction phase will be specified.

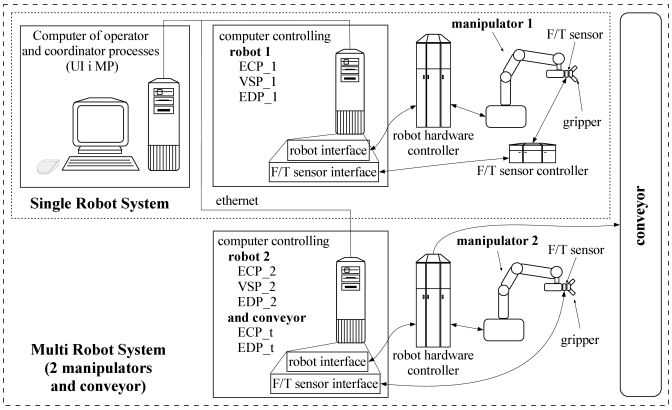

Fig. 2. Experimental setup

The experimental setup (Fig. 2) consists of two modified IRb-6 manipulators with additional active degree of freedom located in the wrist [43] and force/torque sensors, conveyor, PC computers connected by an Ethernet network supervised by the QNX Neutrino real-time operating system.

Reproducing a drawing by a robot has attracted the attention of other researchers [39]. In our investigations [44–48] the force sensor is used to manually guide the robot holding a pen through the motions producing a drawing and then to reproduce it either by the same robot (Fig. 7) or simultaneously by two robots. Only the latter is specified here.

The teach-in process is conducted by an operator leading the robot arm and thus producing the original drawing. The reproduction phase is done automatically by two robots.

The force sensors play a dual role. On the one hand, they are involved in continuous limb control, thus they are treated as a proprioceptors, and, on the other hand, they detect events occurring in the environment, thus they behave as an exteroceptor. The latter behavior requires the creation of virtual sensors.

Each virtual sensor monitors the state of the drawing process. To do so it contains a finite state automaton (Fig. 3) that monitors the current state of the pen. The force and position measurements are the input obtained directly from the Effector Driver. The current state is memorized in the internal virtual sensor memory $v_{c_{j_k}}$. Drawing starts in the *Above paper* state. Arc A is activated when a downward jerk is detected. The automaton state changes to *Lowering*. The B arc is activated when the impact is detected, leading to the *Paper surface* state. The C arc is associated with an upward jerk and the automaton changes its state to *Lift-off*. Then in the teach-in phase the system switches immediately to the *Above paper* state (arc D). Thus by traversing the states of this automaton the virtual sensor is able to notify the agent whether the pen tip is currently on the paper surface or above it, and so how should it behave during the drawing reproduction phase. The $v^i_{j_k} \in \{Above\ paper, Lowering, Paper\ surface, Lift-off\}$ informs in which state is the automaton, what reflects the current state of drawing.

The experimental system [44] consists of two robots. Both manipulators reproduce drawings, while only one of the manipulators is used to teach them. We started with single robot tasks and gradually shifted out attention to more complex multi-robot tasks. The structure of a multi-robot drawing system is very similar to the structure of the Rubik's cube puzzle solving system [36] (however the former does not require vision, whereas the latter does).

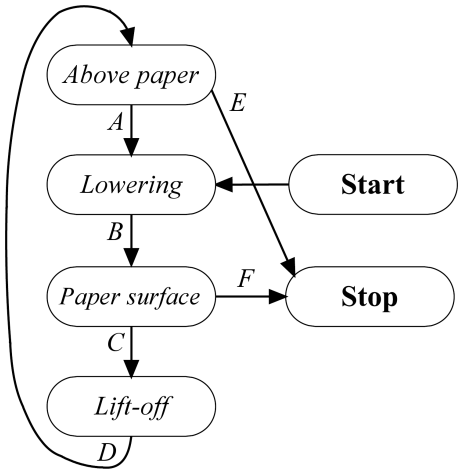

Fig. 3. Graph of the finite state automatons governing the actions of the agent as well as the virtual sensor

The system consists of three agents: two embodied agents a_1, a_2 ($j = 1, 2$) each controlling one robot arm (treated as an effector) and the system coordinator a_0 ($j = 0$). This structure was chosen, because for this task the manipulators have to be continuously coordinated. Thus all decision have to be made in one location and that can be done by a singled out system coordinator a_0. In this case the embodied agents are transparent and simply copy the input data from effector ${}_xc_{e_j}$ and an associated virtual sensor ${}_xc_{V_j}$ to the transmission image ${}_yc_{T_{j0}}$ sent to the coordinator and use the data from the transmission image ${}_xc_{T_{j0}}$ from the coordinator as the command (inserted into the output image ${}_yc_{e_j}$) for the effector. The trajectory that is to be reproduced is stored in the memory of the coordinator, thus the coordinator is responsible for enforcing its execution by the two effectors. The coordinator treats the effectors as its slaves. Obviously other structures could be considered, e.g., a two agent structure, where the agent executing the drawing teach-in phase assumes the responsibilities of the coordinator in the drawing reproduction phase, hence producing an asymmetrical system structure. By using the proposed specification tool many such structures can be discussed and evaluated prior to the start of their implementation. This produces:

- considerable save of time as backtracking is tedious when a large portion of software has already been written,
- reduces the complexity of the resulting software as any changes when already some software has been created result in baroque additions,
- facilitates the distribution of work among the implementation team as right form the onset of the implementation effort each agent (implemented by a subteam) is well defined.

The two robot drawing reproduction system with a coordinator governing the actions of two agents is presented in the

following. First the data structures on which each agent operates are presented and then the transition functions defining the behaviour of each agent.

8.1. Structure of the images. In the following the variables, described in general by (5)–(14), are presented for all of the mentioned agents.

Images of the coordinator – agent a_0. The input transmission image ${}_xc_{T_{01}}$ from agent a_1 consists of three variables. Thus $n_{Tx01} = 3$. The input transmission image ${}_xc_{T_{02}}$ from agent a_2 consists of a single variable ($n_{Tx02} = 1$). The values of those images are defined within agents a_1 and a_2.

The input transmission image ${}_xc_{T_{0h}}$ originates with the operator interface ($n_{Tx0h} = 1$):

$$ {}_xc^i_{T_{0h[1]}} = o^i_h, \tag{23}$$

where $o^i_h \in \{continue, trigger\}$ is the signal sent by an operator. Here the operator is treated as the fourth agent a_h, its internal structure is not elaborated (for obvious reasons).

The input image of the virtual sensor ${}_xc_{V_0}$ is not used ($n_{Vx0} = 0$), thus

$$ {}_xc_{V_0} = \langle\bullet\rangle. \tag{24}$$

The output image of transmission image ${}_yc_{T_{0h}}$ and the output images of the virtual sensor ${}_yc_{V_0}$ are not used ($n_{Ty0h} = 0$, $n_{Vy0} = 0$), thus

$$ \begin{aligned} {}_yc_{T_{0h}} &= \langle\bullet\rangle, \\ {}_yc_{V_0} &= \langle\bullet\rangle. \end{aligned} \tag{25}$$

Thus there is no need to define ${}^m f_{c_{T_{0h}}}$ and ${}^m f_{c_{V_0}}$.

All of the data that must be memorized is extracted from input images ${}_xc_0$ and is stored in c_{c_0}, where $n_{cc0} = 6$. An agent stores the trajectory which is memorized during teach–in process of a single manipulator and then reproduced during reproduction phase in two manipulator system. The description of the trajectory consists of the components of the first four variables presented below:

- $c_{c_{0[1][p]}}$ – the list of the manipulator end–effector velocities $p = 1, \ldots, n$,
- $c_{c_{0[2][p]}}$ – the list of the states of drawing as defined by the graph in Fig. 3,
- $c_{c_{0[3]}}$ – the number (label) of the current node of the trajectory (p),
- $c_{c_{0[4]}}$ – total number of trajectory nodes (n),
- $c_{c_{0[5]}}$ – the current end–effector pose obtained from ${}_xc_{T_{01[1]}}$,
- $c_{c_{0[6]}}$ – a certain time instant obtained from ${}_xc_{T_{01[2]}}$.

The current index p used to index $c_{c_{0[1][p]}}$ and $c_{c_{0[2][p]}}$ lists indicates the currently processed node of the trajectory, while $c_{c_{0[4]}}$ is the total number of trajectory nodes and $c_{c_{0[3]}}$ is the number of the currently memorized or reproduced node (Fig. 4).

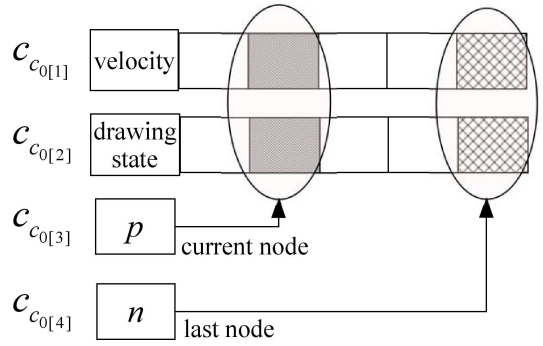

Fig. 4. Data structures used for memorizing the trajectory during the teach–in process

Output transmission buffers ${}_yc_{T_{0j}}$ to agents a_1 and a_2 consists of ten elements each, thus $n_{Ty0j} = 10,\ j = 1, 2$ as defined by (33).

Images of agents a_1 and a_2. The input virtual sensor images ${}_xc_{V_j}$ acquire the drawing state that is produced by the virtual sensor and is defined by the graph presented in Fig. 3 ($n_{Vxj} = 1$) ($j = 1, 2$)

$$ {}_xc^i_{V_{j[1]}} = v^i_{j_1}. \tag{26}$$

In this case the virtual sensor v_{j_1} aggregates information from the proprioceptors. Technically this information is delivered by the Effector Driver.

The general formula (1) assumes the following form ($j = 1, 2$):

$$ \begin{aligned} v^i_{j_1} &= f_{v_{j_1}}(v^{i-1}_{c_{j_1}}, R^i_j) = \\ &= f_{v_{j_1}}(v^{i-1}_{c_{j_1}}, {}^0_{E^i}\mathcal{T}_{m_j}, {}^{E^i}\mathcal{F}^i_{a_j}), \end{aligned} \tag{27}$$

where ${}^0_E\mathcal{T}_{m_j}$ – the current manipulator tool E with respect to the base frame 0 obtained by agent a_j, $v^i_{j_1}$ – the state of the pen attached to the effector of agent a_j produced by the virtual sensor of this agent and (2) assumes the form:

$$ v^i_{c_{j_1}} = f_{vc_{j_1}}(v^{i-1}_{c_{j_1}}, R^i_j) = f_{vc_{j_1}}(v^{i-1}_{c_{j_1}}, {}^0_{E^i}\mathcal{T}_{m_j}, {}^{E^i}\mathcal{F}^i_{a_j}), \tag{28}$$

where $v_{j_1}, v_{c_{j_1}} \in \{$*Above paper*, *Lowering*, *Paper surface*, *Lift-off*$\}$. The embodied agents a_1 and a_2 do not send any commands to their virtual sensors, hence the output images of virtual sensors are not used ($n_{Vyj} = 0$) ($j = 1, 2$)

$$ {}_yc_{V_j} = \langle\bullet\rangle. \tag{29}$$

Thus there is no need to define ${}^m f_{c_{V_j}}$ ($j = 1, 2$).

The input effector image ${}_xc_{e_1}$ contains $n_{ex1} = 2$ data items:

$$ \begin{aligned} {}_xc^i_{e_{1[1]}} &= {}^0_{E^i}\mathcal{T}_{m_1} \\ {}_xc^i_{e_{1[2]}} &= i. \end{aligned} \tag{30}$$

The first is the record of the current end–effector location and the second of the current time. The input effector image ${}_xc_{e_2}$ is not used (i.e., $n_{ex2} = 0$):

$$ {}_xc^i_{e_2} = \langle\bullet\rangle. \tag{31}$$

Input transmission buffers ${}_xc_{T_{j0}}$ of agents a_1 and a_2 consist of ten elements ($n_{Txj0} = n_{Ty0j} = 10,\ j = 1, 2$), the variables holding the effector command sent by the coordinator a_0.

The output effector images ${}_yc_{e_j}$ contain $n_{eyj} = n_{Txj0} = n_{Ty0j} = 10,\ j = 1, 2$, components of the command sent to the effector, initially prepared by the coordinator.

The agent a_1 sends both the input effector image and the input virtual sensor image to the coordinator. This is done through the output transmission images: ${}_yc_{T_{10}}$ consisting of 3 variables ($n_{Ty10} = n_{ex1} + n_{vx1} = n_{Tx01} = 3$).

The agent a_2 sends its input virtual sensor image to the coordinator. This is done through the output transmission images: ${}_yc_{T_{20}}$ consisting of a single variable ($n_{Ty20} = n_{vx2} = n_{Tx02} = 1$).

$$ {}_yc^i_{T_{20[1]}} = v^i_{2_1}. \tag{32} $$

8.2. Transition functions and terminal conditions. In the following the transition functions essential for teach–in and reproduction subtasks are presented. Those functions take as arguments the variables defined in Subsec. 8.1 and produce values that are inserted into output data structures. Each subtask needs several functions ${}^mf_{c_j}$ (Figs. 5, 6), hence the left superscript m is:

- $m = 1, 2$ – drawing teach–in,
- $m = 3, 4, 5, 6, 7$ – drawing reproduction.

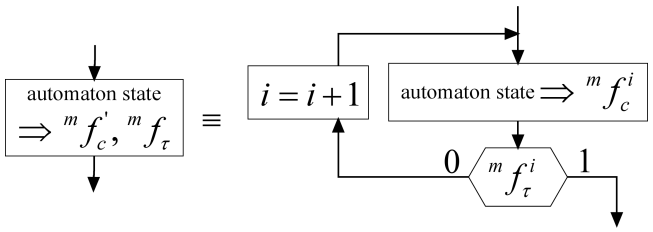

Fig. 5. Single block of the automaton presented in Fig. 6

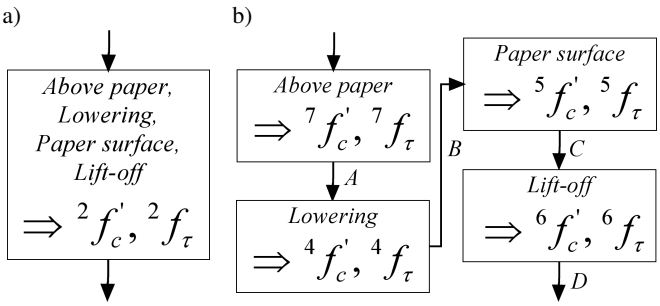

Fig. 6. The correspondence between the state of the automaton presented in Fig. 3 and the transition functions and terminal conditions responsible for the execution of the drawing task. (a) Drawing teach-in phase, (b) Drawing reproduction phase

The coordinator a_0 produces the contents of both output transmission images ${}_yc_{T_{0j}}$ by using ${}^mf'_{c_{T_{0j}}}$ ($j = 1, 2$), ($m = 1, \ldots, 7$):

$$ {}^mf'_{c_{T_{0j}}}({}_xc^i_0) \triangleq \begin{cases} {}_yc^{i+1}_{T_{0j[1]}} = {}^mb_1 \\ {}_yc^{i+1}_{T_{0j[2]}} = {}^m\mathcal{F}_{d_1} \\ {}_yc^{i+1}_{T_{0j[3]}} = {}^m\dot{r}^{i+1}_{(A)d_1} \\ {}_yc^{i+1}_{T_{0j[4]}} = {}^mI_1 \\ {}_yc^{i+1}_{T_{0j[5]}} = {}^m\mathfrak{B}_1 \\ {}_yc^{i+1}_{T_{0j[6]}} = n_{s_1} = \text{const} \\ {}_yc^{i+1}_{T_{0j[7]}} = n_{Q_1} = n_{s_1} - 1 = \text{const} \\ {}_yc^{i+1}_{T_{0j[8]}} = \mu_1 = \texttt{TCIM-velocity} = \text{const} \\ {}_yc^{i+1}_{T_{0j[9]}} = g_{gd_1} = \text{const} \\ {}_yc^{i+1}_{T_{0j[10]}} = {}^W_E\mathcal{T}^{i+1}_{d_1} = \text{const} \end{cases}, \tag{33} $$

where W – frame affixed to the manipulator wrist, b – type of elementary behaviour, n_s – the number of steps that the Effector Driver divides the macrostep into, n_Q – the step number in which the Effector Driver communicates with the control subsystem, $\mu_1 = \texttt{TCIM-velocity}$ – the choice of task coordinates interpolated motion specified in terms of velocity, g – the distance between the gripper jaws (in this example it is disregarded), ${}^W_E\mathcal{T}_{d_1}$ – the manipulator tool E with respect to the wrist frame W.

The symbol $\triangleq$ should be read as: "is defined as". In the case of the drawing reproduction phase the parameters sent to both embodied agents are exactly the same in each interval (macrostep).

Embodied agent a_1 employs the functions ${}^mf'_{c_{T_{10}}}$ ($m = 1, \ldots, 7$) to transfer data to the coordinator.

$$ {}^mf'_{c_{T_{10}}}({}_xc^i_1) \triangleq \begin{cases} {}_yc^{i+1}_{T_{10[1]}} = {}_xc^i_{e_{1[1]}} \\ {}_yc^{i+1}_{T_{10[2]}} = {}_xc^i_{e_{1[2]}} \\ {}_yc^{i+1}_{T_{10[3]}} = {}_xc^i_{V_{1[1]}} \end{cases}. \tag{34} $$

Similarly embodied agent a_2 employs the functions ${}^mf'_{c_{T_{20}}}$ ($m = 1, \ldots, 7$) to transfer data to the coordinator. This function simply copyies the input virtual sensor image ${}_xc_{V_2}$ to the output transmission image ${}_yc_{T_{20}}$:

$$ {}^mf'_{c_{T_{20}}}({}_xc^i_2) \triangleq \left\{ {}_yc^{i+1}_{T_{20}} = {}_xc^i_{V_2} \right. . \tag{35} $$

Analogically, for both embodied agents a_1 and a_2, the function ${}^mf'_{c_{e_j}}$ ($m = 1, \ldots, 7$) produces the Effector Driver command ${}_yc_{e_j}$ by copying the input transmission image ${}_xc_{T_{j0}}$ containing the command that had been previously prepared by the system coordinator a_0:

$$ {}^mf'_{c_{e_j}}({}_xc^i_j) \triangleq \left\{ {}_yc^{i+1}_{e_j} = {}_xc^i_{T_{j0}} \quad \text{for} \quad j = 1, 2 \right. . \tag{36} $$

where $j = 1, 2$.

In the following the transition functions essential for teach–in and reproduction subtasks are presented.

Drawing teach-in phase. As it was previously mentioned, the teach–in process for multiple robots utilizes a single manipulator. The manipulator commanded by agent a_2 stands still (Table 1). The manipulator used in the teach–in process (a_1) is compliant in linear directions, but its orientation is fixed (Table 2). It should be reminded here that the primary job of transition functions is to produce the parameters of the control law for each of the Effector Drivers and for both phases. In every case the parameters that have to be produced will be presented in tabular form. By dispatching the values of those parameters each embodied agent (a_1 and a_2) forces its effector to behave as required. Transition functions obviously produce other values, governing inter–agent transmissions, management of internal data structures etc.

Table 1
Manipulator controlled by the agent a_2 is blocked ($m = 1, 2$)

c	${}^m b_2$	${}^m\mathcal{F}_{d_2}$	${}^m\dot{r}^{i+1}_{(A)d_2}$	${}^m I_2$	${}^m\mathcal{B}_2$
x	u	–	0	–	–
y	u	–	0	–	–
z	u	–	0	–	–
a_x	u	–	0	–	–
a_y	u	–	0	–	–
a_z	u	–	0	–	–

Table 2
Effector Driver command parameters for compliant motion in linear directions ($m = 1, 2, 3$)

c	${}^m b_1$	${}^m\mathcal{F}_{d_1}$	${}^m\dot{r}^{i+1}_{(A)d_1}$	${}^m I_1$	${}^m\mathcal{B}_1$
x	c	0	–	20	0.005
y	c	0	–	20	0.005
z	c	0	–	20	0.005
a_x	u	–	0	–	–
a_y	u	–	0	–	–
a_z	u	–	0	–	–

The function ${}^1f'_{c_{c_0}}$ initiates the internal variables:

$$ {}^1f'_{c_{c_0}}({}_x c^i_0) \triangleq \begin{cases} c^{i+1}_{c_{0[3]}} = 1 \\ c^{i+1}_{c_{0[4]}} = 0 \\ c^{i+1}_{c_{0[5]}} = {}_x c^i_{T_{01[1]}} \end{cases} . \tag{37} $$

The current node of the trajectory is set to 1. This phase finishes when the pen touches the paper:

$$ {}^1f_{\tau_0}({}_x c^i_0) \triangleq \begin{cases} 0 & \text{for } {}_x c^i_{T_{01[3]}} \neq \textit{Paper surface} \\ 1 & \text{otherwise} \end{cases} . \tag{38} $$

The next phase is the teach-in process itself (recording of the drawing). Now the manipulator is moved in the same way, but the motion trajectory is recorded by memorizing the velocity of the pen tip $\dot{r}^i_{(A)r_1}$ on the surface of the paper ($x-y$ coordinates), at constant intervals of time ($n_{s_1} = 20$ ms).

The function ${}^2f'_{c_{c_0}}$ is defined as:

$$ {}^2f'_{c_{c_0}}({}_x c^i_0) \triangleq \begin{cases} c^{i+1}_{c_{1[1][p]}} = \begin{cases} c^i_{c_{0[1][p]}} & \text{for } p=1,\ldots,c^i_{c_{0[3]}}-1 \\ \dot{r}^i_{(A)r_0} & \text{for } p=c^i_{c_{0[3]}} \end{cases} \\ c^{i+1}_{c_{0[2][p]}} = \begin{cases} c^i_{c_{0[2][p]}} & \text{for } p=1,\ldots,c^i_{c_{0[3]}}-1 \\ {}_x c^i_{T_{01[3]}} & \text{for } p=c^i_{c_{0[3]}} \end{cases} \\ c^{i+1}_{c_{0[3]}} = c^i_{c_{0[3]}} + 1 \\ c^{i+1}_{c_{0[4]}} = c^i_{c_{0[4]}} + 1 \\ c^{i+1}_{c_{0[5]}} = {}_x c^i_{T_{01[1]}} \end{cases} . \tag{39} $$

It, among others, adds a new node to the $c_{c_{0[1][p]}}$ and $c_{c_{0[2][p]}}$ lists, where:

$$ \dot{r}^i_{(A)r_0} = \frac{\overset{E^{i-1}}{E^i} r_{(A)m_1}}{\Delta t\ n_{s_1}} = \frac{\mathcal{A}_A^{-1}((c^i_{c_{0[5]}})^{-1}\ {}_x c_{T_{01[1]}})}{\Delta t\ n_{s_1}}, \tag{40} $$

where the operator $\mathcal{A}_A^{-1}$ transforms the homogeneous matrix into column vector containing three Cartesian coordinates supplemented by angle and axis representation of orientation. The operator decides when the drawing teach-in process is finished by sending a dedicated signal to the agent:

$$ {}^2f_{\tau_0}({}_x c^i_0) \triangleq \begin{cases} 0 & \text{for } {}_x c^i_{T_{0h[1]}} = \textit{continue} \\ 1 & \text{for } {}_x c^i_{T_{0h[1]}} = \textit{trigger} \end{cases} . \tag{41} $$

Drawing reproduction phase. The second phase of the task execution consists in the reproduction of the memorized drawing. Now the coordinator sends exactly the same commands to both manipulators. At the beginning the operator moves two compliant manipulators to the initial locations, in which the memorized drawing has to be reproduced (Table 2). The function ${}^3f'_{c_{c_0}}$ is an identity function retaining the previously memorized data:

$$ {}^3f'_{c_{c_0}}({}_x c^i_0) \triangleq \left\{ c^{i+1}_{c_{0[l]}} = c^i_{c_{0[l]}} \quad l = 1,\ldots,6 \right. \tag{42} $$

In the initial location the pen should be above the paper surface. Then the operator signals the system to start the automatic reproduction process .

$$ {}^3f_{\tau_0}({}_x c^i_0) \triangleq {}^2f_{\tau_0}({}_x c^i_0). \tag{43} $$

First, both pens are moved down (the *Lowering* state) (Table 3a).

The function ${}^4f'_{c_{c_0}}$ is an identity function retaining the previously memorized data:

$$ {}^4f'_{c_{c_0}}({}_x c^i_0) \triangleq {}^3f'_{c_{c_0}}({}_x c^i_0). \tag{44} $$

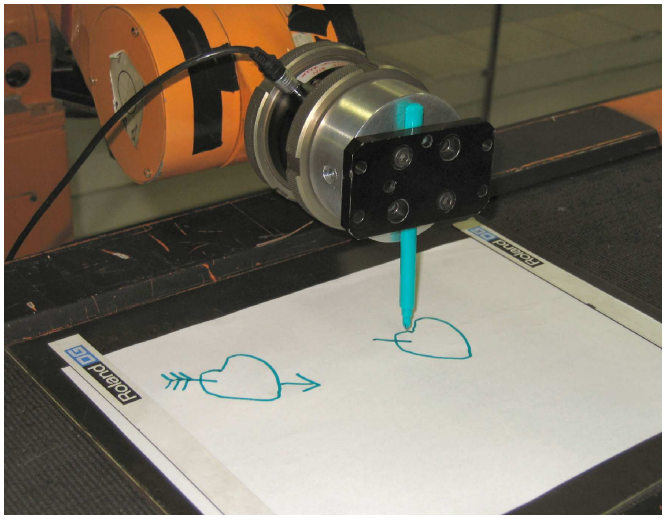

Fig. 7. Copying drawings – the reproduction phase

Table 3
Drawing copying task Effector Driver command arguments

a) lowering phase ($m = 4$)

c	${}^4b$	${}^4\mathcal{F}_d$	${}^4\dot{r}^{i+1}_{(A)d}$	${}^4I$	${}^4\mathfrak{B}$
x	u	–	0	–	–
y	u	–	0	–	–
z	g	–	0.01	20	0.005
a_x	u	–	0	–	–
a_y	u	–	0	–	–
a_z	u	–	0	–	–

b) lift off ($m = 6$)

c	${}^6b$	${}^6\mathcal{F}_d$	${}^6\dot{r}^{i+1}_{(A)d}$	${}^6I$	${}^6\mathfrak{B}$
x	u	–	0	–	–
y	u	–	0	–	–
z	u	–	−0.01	–	–
a_x	u	–	0	–	–
a_y	u	–	0	–	–
a_z	u	–	0	–	–

The terminal condition checks if both pens have hit the paper surface:

$$ {}^4f_{\tau_0}({}_xc^i_0) \triangleq \begin{cases} 0 & \text{for } {}_xc^i_{T_{01[3]}} \neq \textit{Paper surface} \vee \\ & \quad {}_xc^i_{T_{02[1]}} \neq \textit{Paper surface} \\ 1 & \text{otherwise} \end{cases} . \quad (45) $$

If one of the pens hits the paper surface it starts to push first, it pushes the paper surface with the desired force as it can be derived from the control law and the parameters contained in Table 3a. Then after the second pen strikes the surface of the paper the system starts to draw two pictures with the same speed and of the same size. The robots are mutually synchronized (45) before each new line segment is reproduced, thus the two drawings will appear approximately at the same time – this is due to the implementation of continuous coordination.

Hence after impact the vertical motion stops and the horizontal motion on the surface of the paper is induced with simultaneous desired force set in vertical direction (Table 4a) (the *Paper surface* state), where

$$ \dot{r}^i_{(A)d_j[x]} \triangleq c^i_{c_{0[1][p][x]}}, \quad \dot{r}^i_{(A)d_j[y]} \triangleq c^i_{c_{0[1][p][y]}}, \quad j = 1, 2 \quad (46) $$

and $p \triangleq c^i_{c_{0[3]}}$ is the index of the currently reproduced drawing node.

Table 4
Copying drawing task Effector Driver command arguments

a) horizontal motion on the surface of the paper ($m = 5$)

c	${}^5b$	${}^5\mathcal{F}_d$	${}^5\dot{r}^{i+1}_{(A)d}$	${}^5I$	${}^5\mathfrak{B}$
x	u	–	$\dot{r}^{i+1}_{(A)d[x]}$	–	–
y	u	–	$\dot{r}^{i+1}_{(A)d[y]}$	–	–
z	c	1	–	20	0.005
a_x	u	–	0	–	–
a_y	u	–	0	–	–
a_z	u	–	0	–	–

b) horizontal motion above the surface of the paper ($m = 7$)

c	${}^7b$	${}^7\mathcal{F}_d$	${}^7\dot{r}^{i+1}_{(A)d}$	${}^7I$	${}^7\mathfrak{B}$
x	u	–	$\dot{r}^{i+1}_{(A)d[x]}$	–	–
y	u	–	$\dot{r}^{i+1}_{(A)d[y]}$	–	–
z	u	1	0	–	–
a_x	u	–	0	–	–
a_y	u	–	0	–	–
a_z	u	–	0	–	–

The function ${}^5f'_{c_{c_0}}$ increments the current time instant for the memorized trajectory node:

$$ {}^5f'_{c_{c_0}}({}_xc^i_0) \triangleq \begin{cases} c^{i+1}_{c_{0[3]}} = c^i_{c_{0[3]}} + 1 \\ c^{i+1}_{c_{0[l]}} = c^i_{c_{0[l]}} \qquad \text{for } l = 1, 2, 4, 5, 6 \end{cases} . \quad (47) $$

The drawing of a single segment lasts until the pen tip reaches the location in which an upward jerk was recorded in the teach-in phase or reproduction is finished:

$$ {}^5f_{\tau_0}({}_xc^i_0) \triangleq \begin{cases} 0 & \text{for } c^i_{c_{0[2][p]}} \neq \textit{Lift-off} \wedge p = c^i_{c_{0[3]}} \neq c^i_{c_{0[4]}} \\ 1 & \text{otherwise} \end{cases} . \quad (48) $$

Then the pen is raised above the paper (Table 3b). The function ${}^6f'_{c_{c_0}}$ memorizes the initial time instant of the current transition function execution (beginning of the lift-off operation):

$$ {}^6f'_{c_{c_0}}({}_xc^i_0) \triangleq \begin{cases} c^{i+1}_{c_{0[6]}} = {}_xc^i_{T_{01[2]}} & \text{for} \quad i = i_0 \\ c^{i+1}_{c_{0[6]}} = c^i_{c_{0[6]}} & \text{for} \quad i \neq i_0 \\ c^{i+1}_{c_{0[l]}} = c^i_{c_{0[l]}} & \forall i \wedge l = 1, 2, 3, 4, 5 \end{cases} . \quad (49) $$

The operation is executed until the desired time elapses:

$$ {}^6f_{\tau_0}({}_xc^i_0) \triangleq \begin{cases} 0 & \text{for } {}_xc^i_{T_{01[2]}} - c^i_{c_{0[6]}} < i_d \\ 1 & \text{otherwise} \end{cases} , \quad (50) $$

where i_d is the desired duration.

The trajectory that the pen tip traverses above the paper surface in the horizontal plane is an accurate copy of the memorized trajectory in the same plane (46). In the reproduction phase the motions executed in the *Above paper* and *Lift-off* states are fully position controlled, whilst in the other state hybrid position-force control is utilized. The *Above paper* motions are executed in a horizontal plane (the z coordinate is kept constant (Table 4b), even if it varied during the teach-in phase). The function ${}^{7}f'_{c_{c_0}}$ increments the current time stamp of the memorized trajectory pose:

$$ {}^{7}f'_{c_{c_0}}({}_{x}c_0^i) \triangleq {}^{5}f'_{c_{c_0}}({}_{x}c_0^i). \tag{51} $$

The motion above the paper lasts until the pen tip reaches the location in which an impact was recorded in the teach-in phase.

$$ {}^{7}f_{\tau_0}({}_{x}c_0^i) \triangleq \begin{cases} 0 \text{ for } c^i_{c_{0[2][p]}} \neq Paper\ surface \wedge c^i_{c_{0[3]}} \neq p = c^i_{c_{0[4]}} \\ 1 \text{ otherwise} \end{cases}. \tag{52} $$

Then the pen can move down again to start reproducing of the following segment.

8.3. Drawing reproduction using a conveyor. The multi-robot drawing task was also executed in another configuration. The motion of the manipulators in the y direction was substituted by the motion of a conveyor on which the two drawing papers were located. So instead of moving the robots in the y direction the conveyor shifted the paper in that direction. Hence, the system consisted of three motion inducing devices, each capable of independent motion. Thus a three effector MRROC++ based system was created. The tests demonstrated that the so extended system also works correctly.

8.4. Experimental results. The presented formal specification was used as the description of the controller of a robot capable of reproducing taught–in drawings. The controller was implemented using the MRROC++ [36, 48, 49] robot programing framework. This controller was used in the below described experiments. Figure 8 presents the three dimensional trajectories of the end-effector motion during teach-in and reproduction of the six feathers of an arrow drawn by the operator (Fig. 7). A visible difference between the graphs is caused by the way the pen moves up and down and above the paper. The operator makes unconstrained moves, hence the trajectory above the paper is uneven in the vertical direction. The reproduction algorithm produces exact horizontal motions, thus the trajectories above the paper are horizontal. This is evident in the graph in Fig. 9. The plots obtained for both effectors during the reproduction phase are very similar, thus only the plots for one of the effectors are presented here.

There are four segments of the trajectory marked as: **1**, **2**, **3** and **4** in the graphs in Figs. 9 and 10. All the four segments occur while drawing each feather of an arrow: **1** – motion on the paper surface, **2** – pen tip lift-off, **3** – motion above the paper, **4** - lowering of the pen tip. The symbol "$*$" draws attention to the fragment of the plot representing the impact caused by the pen tip hitting the surface of the paper. During the whole of the teach-in phase and in segment **1** of the reproduction phase, the Effector Driver is commanded to reach the vertical force of $1N$, however, in the segments: 2, 3 and 4 of the reproduction phase, the motion is purely position controlled. Experiments show that the applied algorithms are robust enough to execute the whole task correctly.

a) the teach-in phase

b) the reproduction phase

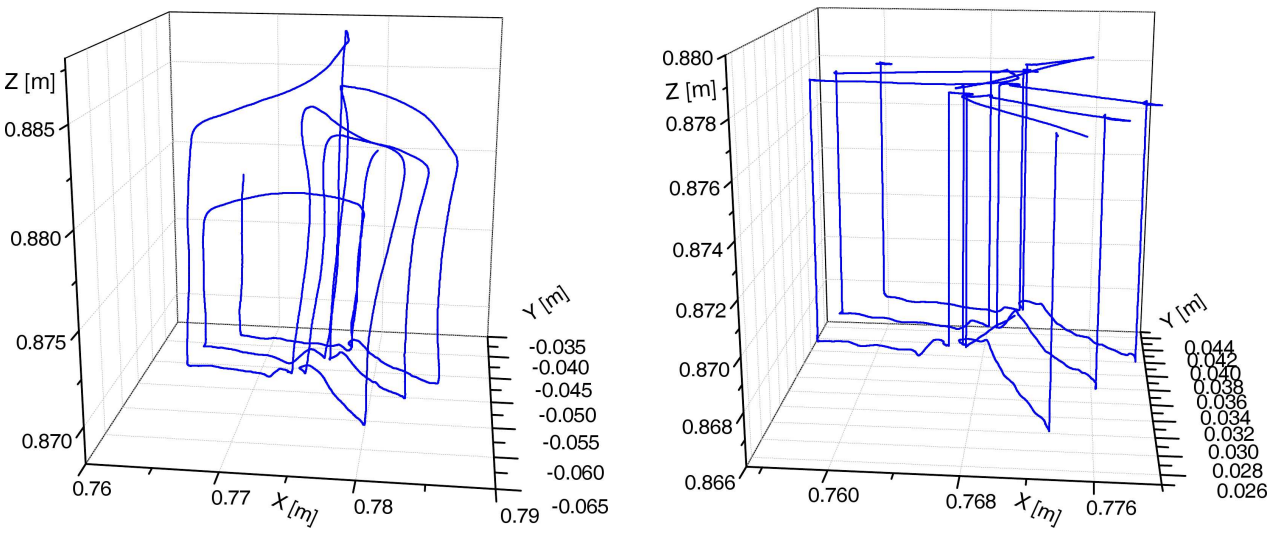

Fig. 8. Three dimensional motion trajectory during copying drawings

a) the teach-in phase b) the reproduction phase

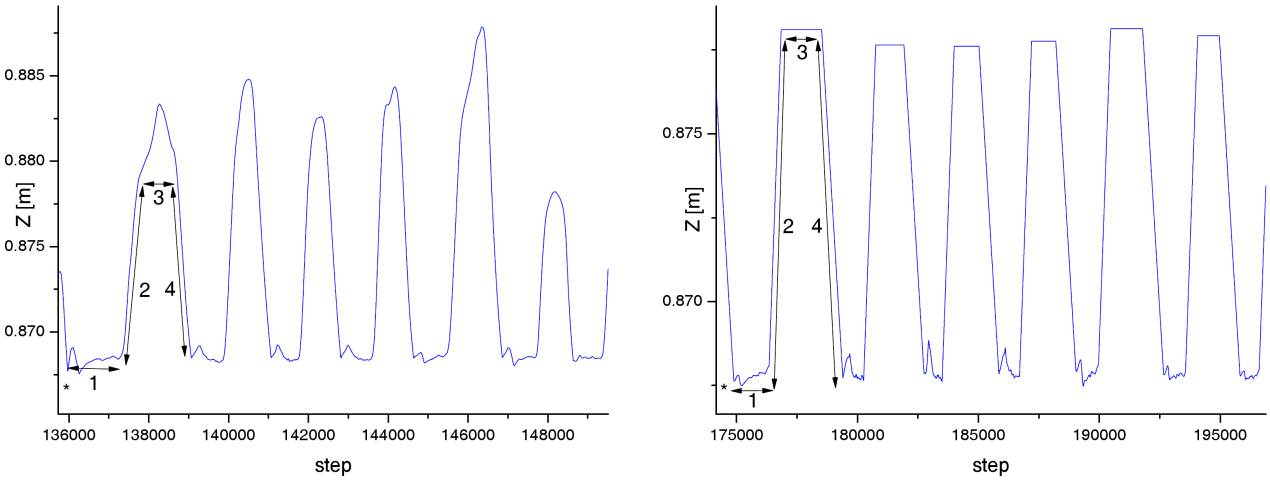

Fig. 9. The z coordinate of the pen tip pose during copying drawings

a) the teach-in phase b) the reproduction phase

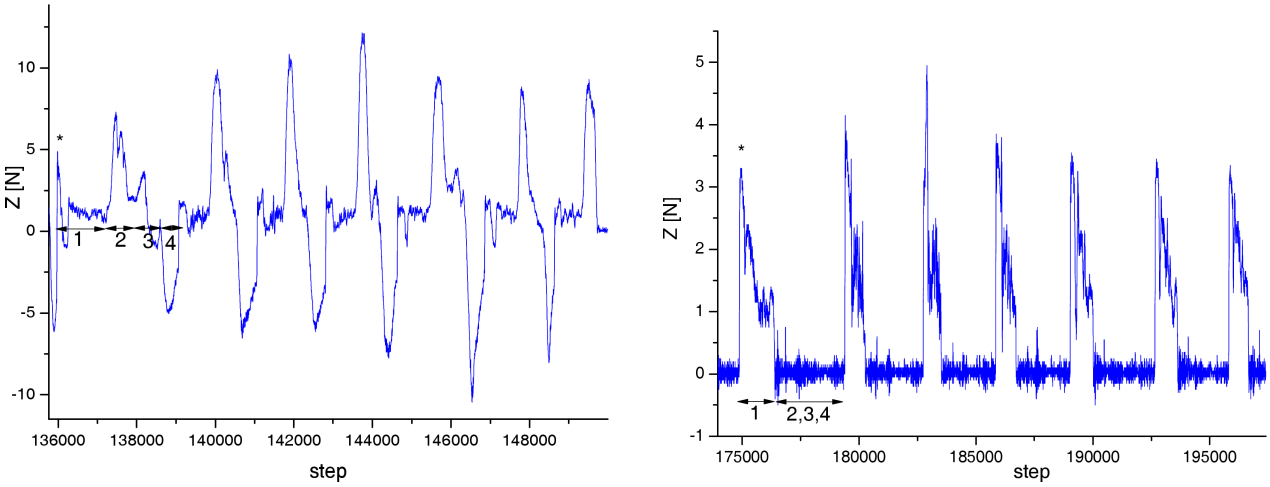

Fig. 10. Force applied in the vertical direction during copying drawings

9. Conclusions

The paper presents a formal approach to the specification of controllers executing diverse tasks. This approach assumes that a multi-robot system is composed of embodied agents, where each such agent has its effector, receptors and a capability to exchange information with other such agents.

The structure is expressed in terms of interconnections between the agents constituting the system, while its operation is described in terms of transition functions governing the actions of each agent. This part of the presented approach is general. Every system can be described in such a way. However, the paper goes deeper into the description of a particular system containing manipulators interacting with their environment. In this case three elementary behaviours have been distinguished and a control law enabling the implementation of those behaviours has been formulated. The presented method of system specification falls into the category of top-down methods, where the general description is refined going into ever more detailed description. The general approach has been exemplified by specifying the operation of a two effector robot system capable of reproducing drawings. This specification was subsequently used as the basis for the implementation of the system. `MRROC++` robot programming framework was used as an implementation tool for it. This design procedure has been used also for the implementation of other systems, e.g.: two-handed system solving a Rubik's cube puzzle, two-effector haptic device, where one of the arms was used as a master and the other as a slave device. In all of those cases the proposed design procedure led to a quick and effective result.

Acknowledgements. The authors gratefully acknowledge the support of the Ministry of Science and Higher Education grant N N514 1287 33.

REFERENCES

[1] D.M. Lyons and M.A. Arbib, "A formal model of computation for sensory-based robotics", *IEEE Transactions on Robotics and Automation* 5 (3), 280–293 (1989).

[2] C. Zieliński, "Description of semantics of robot programming languages", *Mechatronics* 2 (2), 171–198 (1992).

[3] A.C. Dominguez-Brito, D. Hernández-Sosa, J. Isern-González, and J. Cabrera-Gámez, "CoolBOT: a component model and software infrastructure for robotics", *Software Engineering for Experimental Robotics* 1, 143–168 (2007).

[4] D. Brugali, "Stable analysis patterns for robot mobility", *Software Engineering for Experimental Robotics* 9–30 (2007).

[5] D. Brugali, A. Agah, B. MacDonald, I. Nesnas, and W. Smart, "Trends in Robot Software Domain Engineering", *Software Engineering for Experimental Robotics* 1, 3–8 (2007).

[6] I.A.D. Nesnas, "The CLARAty project: Coping with hardware and software heterogenity", *Software Engineering for Experimental Robotics* 30, 31–70 (2007).

[7] D. Brugali, M. Alencastre-Miranda, L. Muñoz-Gómez, D. Botturi, and L. Cragg, "Trends in Software Environments for Networked Robotics", *Software Engineering for Experimental Robotics* 1, 401–408 (2007).

[8] 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 (1986).

[9] V. Hayward and S. Hayati, "Kali: an environment for the programming and control of cooperative manipulators", *7th American Control Conf.* 1, 473–478 (1988).

[10] V. Hayward, L. Daneshmend, and S. Hayati, "An overview of KALI: a system to program and control cooperative manipulators", *Advanced Robotics* 1, 547–558 (1989).

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

[12] C. Blume and W. Jakob, *Programming Languages for Industrial Robots*, Springer-Verlag, Berlin, 1986.

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

[14] C. Zieliński, "Flexible controller for robots equipped with sensors", *9th Symp. Theory and Practice of Robots and Manipulators* 187, 205–214 (1993).

[15] C. Zieliński, "Control of a multi-robot system", *2nd Int. Symp. Methods and Models in Automation and Robotics MMAR'95* 1, 603–608 (1995).

[16] C. Zieliński, *Object–oriented Programming of Multi-robot Systems*, Cambridge University, Cambridge, 1997.

[17] C. Zieliński, "The MRROC++ system", *First Workshop on Robot Motion and Control, RoMoCo'99* 1, 147–152 (1999).

[18] S. Fleury and M. Herrb, "Genom user's guide", in *Report, LAAS*, Publishing House of Toulouse, Toulouse, 2001.

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

[20] L. Petersson, D. Austin, and H. Christensen, "DCA: a distributed control architecture for robotics", *Proc. Int. Conf. on Intelligent Robots and Systems IROS* 1, CD-ROM (2001).

[21] R. Simmons, R. Goodwin, C. Fedor, and J. Basista, *Task Control Architecture: Programmer's Guide to Version 8.0*, Carnegie Mellon University, Pittsburgh, 1997.

[22] R. Simmons and D. Apfelbaum, "A task description language for robot control", *Int. Conf. on Itelligent Robots and Systems IROS'98*, 1, CD-ROM (1998).

[23] E.R. Morales, "GENERIS: The EC-JRC generalised software control system for industrial robots", *Industrial Robot* 26 (1), 26–32 (1999).

[24] H. Bruyninckx, *OROCOS – Open Robot Control Software*, http://www.orocos.org/ (2002).

[25] H. Bruyninckx, "The real-time motion control core of the OROCOS project", *Proc. IEEE Int. Conf. on Robotics and Automation* 14, 2766–2771 (2003).

[26] J. Cabrera-Gámez, A.C. Domínguez-Brito, and D. Hernández-Sosa, "Sensor based intelligent robots", in: *A Component-Oriented Programming Framework for Robotics*, pages 282–304, Springer, Berlin, 2002.

[27] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Orebäck, "Towards component-based robotics", *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* 1, 163–168 (2005).

[28] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Orebäck, "Orca: A component model and repository", *Software Engineering for Experimental Robotics* 1, 231–251 (2007).

[29] B.P. Gerkey, R.T. Vaughan, and A. Howard, "The player/stage project: tools for multi-robot and distributed sensor systems", *Proc. Int. Conf. Advanced Robotics* 1, 317–323 (2003).

[30] T. Collett, B. MacDonald, and B. Gerkey, "Player 2.0: toward a practical robot programming framework", *Australasian Conf. on Robotics and Automation* 2, CD-ROM (2005).

[31] R.T. Vaughan, B.P. Gerkey, and A. Howard, "Reusable robot software and the player/stage project", *Software Engineering for Experimental Robotics* 1, 267–290 (2007).

[32] K. Slonneger and B.L. Kurtz, *Formal Syntax and Semantics of Programming Languages: a Laboratory Based Approach*, Addison-Wesley Publishing Company, Reading, 1995.

[33] S. Ambroszkiewicz, "Entish: A language for describing data processing in open distributed systems", *Fundamenta Informaticae* 60 (1–4), 41–66 (2004).

[34] C. Zieliński, "Transition-function based approach to structuring robot control software", in: *Robot Motion and Control: Recent Developments, Lecture Notes in Control and Information Sciences*, Vol. 335, pages 265–286, ed. K. Kozłowski, Springer Verlag, Berlin, 2006.

[35] C. Zieliński, "Formal approach to the design of robot programming frameworks: the behavioural control case", *Bull. Pol. Ac.: Tech.* 53 (1), 57–67 (2005).

[36] 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: Int. J.* 34 (5), 368–375 (2007).

[37] M.S. Lim, J. Lim, and S.R. Oh, "Stiffness adaptation and force regulation using hybrid systemapproach for constrained robots", *Intelligent Robots and Systems, Int. Conf.* 2, CD-ROM (1999).

[38] R.V. Dubey, T.F. Chan, and S.E. Everett, "Variable damping impedance control of a bilateral teleroboticsystem", *Control Systems Magazine, IEEE* 17 (1), 37–45 (1997).

[39] T. Tsumugiwa, R.Yokogawa, and K.Hara, "Variable impedance control based on estimation of human arm stiffness for human-robot cooperative calligraphic task", *Proc. IEEE Conf. on Robotics and Automation* 1, 644–650 (2002).

[40] H. Bruyninckx and J. De Schutter, "Specification of force-controlled actions in the task frame formalism: a synthesis", *IEEE Trans. on Robotics and Automation* 12 (4), 581–589 (1996).

[41] O. Khatib, "A unified approach for motion and force control of robot manipulators: the operational space formulation", *Int. J. Robotics and Automation* RA-3 (1), 43–53 (1987).

[42] M. Staniak, T. Winiarski, and C. Zieliński, "Parallel visual-force control", *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* 1, 937–942 (2008).

[43] K. Mianowski, "Analysis of properties of special gripper for a service robot", *Materials of TMM Conf.* 1, 185–190 (2006), (in Polish).

[44] T. Winiarski and C. Zieliński, "Force control in dual arm systems", *9-th National Conf. on Robotics – Advances in Robotics* 2, 267–276 (2006), (in Polish).

[45] T. Winiarski and C. Zieliński, "Position-force controller experimental station", *Robotic's Progress: Control of Robots with environment Perception* 1, 85–94 (2004), (in Polish).

[46] T. Winiarski and C. Zieliński, "Implementation of position–force control in MRROC++", *Proc. 5th Int. Workshop on Robot Motion and Control* 1, 259–264 (2005).

[47] C. Zieliński, W. Szynkiewicz, and T. Winiarski, "Applications of MRROC++ robot programming framework", *Proc. 5th Int. Workshop on Robot Motion and Control* 1, 251–257 (2005).

[48] C. Zieliński, T. Winiarski, W. Szynkiewicz, M. Staniak, W. Czajewski, and T. Kornuta, "MRROC++ based controller of a dual arm robot system manipulating a Rubik's cube", *Technical Report* 06–10, 167–171 (2006).

[49] C. Zieliński, "Motion generators in MRROC++ based robot controller", *14th CISM–IFToMM Symposium on Robotics* 1, 299–306 (2002).