BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 68, No. 1, 2020
DOI: 10.24425/bpasts.2020.131827

A universal architectural pattern and specification method for robot control system design

T. KORNUTA[1,2]*, C. ZIELIŃSKI[2], and T. WINIARSKI[2]

[1] IBM Research, Almaden Research Center, 650 Harry Road, San Jose, CA 95120, United States
[2] Warsaw University of Technology, Institute of Control and Computation Engineering, Nowowiejska 15/19, 00-665 Warsaw, Poland

Abstract. The paper presents a universal architectural pattern and an associated specification method that can be applied in the design of robot control systems. The approach describes the system in terms of embodied agents and proposes a multi-step decomposition enabling precise definition of their inner structure and operation. An embodied agent is decomposed into effectors, receptors, both real and virtual, and a control subsystem. Those entities communicate through communication buffers. The activities of those entities are governed by FSMs that invoke behaviours formulated in terms of transition functions taking as arguments the contents of input buffers and producing the values inserted into output buffers. The method is exemplified by applying it to the design of a control system of a robot executing one of the most important tasks for a service robot, i.e. picking up, by a position–force controlled robot, an object located using an RGB-D image acquired from a Kinect. Moreover in order to substantiate the universality of the presented approach we present how classical, known from the literature, robotic architectures can be expressed as systems composed of one or more embodied agents.

Key words: Autonomous Agents, Control Architectures and Programming, Service Robots, Range Sensing, Recognition, Grasping.

1. Introduction

Although robots are meant to be universal programmed machines in reality they are designed for a specific class of tasks. Their universality is limited to that category. The class of tasks dictates both the mechanical structure of the system and the functions that its control system should realise, thus defining the behaviour of the designed system as a whole. In robot systems the diversity of effectors (e.g. manipulators, grippers and other tools, wheels, legs, tracks, propellers, as well as their diverse actuators) and receptors (e.g. cameras, laser scanners, sonars, inertial and force sensors, proximity sensors, compasses) is vast. On top of this diversity the mentioned hardware can be put to work on very different tasks.

The main problem that this paper addresses is whether such a diversity of robot effectors, receptors and tasks can be abstracted away to provide a universal description of the system structure and its behaviour facilitating the design of the control system of a specific robotic system.

1.1. Subject of the quest. As robot control systems are computer based, the software is of paramount importance. Software engineering provides clues on to how to specify such systems in general, but robotics is a well established discipline, thus a lot of precious domain specific knowledge would be neglected if we were to rely only on the available general approaches. However, the hints provided by software engineering also should not be neglected. The first hint is associated with the search for a universal system architecture [36]. Such a general architecture should not over-constrain the designer, enabling him/her to structure the system as required, however providing helping guidelines how to do this appropriately. The elements that should be left to the designer of the system are the overall system structure, the implementation paradigm (e.g. procedural, object-oriented, component based) and implementation means (e.g. operating system, programming language). Thus the design method should be immune to those elements. It should provide general architectural patterns, which if followed by the designer should lead to a well formed control system. Moreover, design patterns enabling the creation of system components should be made available [27, 30]. Emergence of general architectural patterns does not only imply code reusability [18] and decrease in the design and implementation costs, but also facilitates replication of solutions and experiments, thus helps overcoming the *reproducibility crisis* identified as one of the main problems that both robotics [11] and artificial intelligence [33] communities are struggling with. Last but not least, the approach should provide a universal symbolic notation by which the designed systems will be specified. This notation should enable an in-depth discussion of the proposed solutions. This often leads to the disclosure of otherwise unforeseen properties or misfeatures of the system. Such an abstraction should also enable formulation of a design method, general enough to produce specification of all kinds of robotic systems.

1.2. Evolution of the embodied agent-based architecture. Proper design of any software, and control software specifically, requires two major steps: specification and implementa-

*e-mail: tkornuta@gmail.com

Manuscript submitted 2019-01-23, revised 2019-09-15, initially accepted for publication 2019-10-09, published in February 2020

tion, where the first defines what has to be done and the second focuses on how it has to be done. This paper concentrates on the specification phase, as the best implementation will not improve upon poor specification. Software engineering provides general hints how to design any software systems, but particular systems are always domain specific. The developed specification methodology relies on the concept of an embodied agent [13, 14], being the solution to the problem formulated in Section 1.1. The architectural pattern of the embodied agent has matured over years, as it has been used for the development of diverse robot control systems [93]. The origins of this approach can be traced back to our work on multi-robot control systems [82, 83] and programming frameworks for development of distributed controllers (i.e. controllers consisting of many collaborating processes). In [84] we introduced the concept of embodied agent as an entity consisting of a single Control Subsystem and controlling from zero or more Effectors and Receptors and focused on the structure of the controllers of several classical force-control benchmarks such as following of an unknown contour or rotating a crank and copying drawings by a single-robot arm. Subsequently dual arm system solving the Rubik's cube puzzle [90] was created. In [91] we focused on further formalisation of the operation of the Control Subsystem and proposed the abstraction of the position–force control law realised by the Virtual Effectors, seen as three elementary behaviours from the point of view of the Control Subsystem, while [92] focused on the crucial aspects of motion-generation realised by processes constituting a single embodied agent. Visual servoing was also included [69, 70]. In [42] we have generalized the concept of behaviour parameterised by transition functions to describe the operation of Control Subsystem as well as Virtual Receptors and Effectors. In [88] we presented a specification of a part of the system of our dual-arm Velma robot, which used 3D object recognition with visual servoing and impedance control in the task of putting a lid on the box, while in [41] we have made an effort to formalize robotics skills as an assembly of several behaviours on the example of grasping object. In our most recent works we deliberate on using the embodied agent-based approach for designing of robotic systems possessing both fixed [93] and variable [89] structure of the controller with cloud support [24].

1.3. Contributions of the paper. The main contributions of this paper are as follows:

- We present in a comprehensive way a robotic system specification method that defines both the system structure and its activities, avoiding the notoriously imprecise style of box and arrow presentations of architectures dominating the domain of robotics. Our specification method enables the replication of robotic systems in other laboratories, thus facilitates benchmarking.
- We refine the architectural pattern of an embodied agent by describing the activities of each of the subsystems by an FSM and introducing transitions between states defined by combinations of terminal and initial conditions, which results in an effective mechanism of switching behaviours.
- We present the enhanced version of the step-by-step design procedure leading to a detailed specification of the considered robotic system.
- Finally, we showcase our approach on a control system of a service robot executing a complex task. The selected example utilizes both the state-of-the-art visual perception (relying on RGB-D sensors) and force sensing, a combination that is crucial for every modern service robot.

Both the comprehensive presentation of the methodology and the exemplary specification resulting from it are novel.

1.4. Structure of the paper. Section 2 introduces an embodied agent, being the central tenet of the presented approach, and briefly explains all related concepts. Section 3 describes the associated methodology of robot control system design. Section 4 presents the application of the proposed approach to the design of an exemplary robotic system. The system consists of a single-arm robot (manipulator) equipped with a force-torque sensor and a two-finger gripper, supplemented with an RGB-D sensor. This system is put to the task of acquiring specific randomly located objects. The task requires several crucial capabilities of modern service robots, i.e. combines 3D visual perception with force sensing and motion control. The specification is followed by a brief description of the experiments validating the developed controller. As the claim of this paper is that the proposed robotic control system design and specification method is universal, Section 5 discusses how it can be used to produce diverse architectures that have been presented by others, thus relating our work to that of other researchers. Section 6 concludes the paper, pointing out the major advantages of the presented approach.

2. Universal model of a robotic system

The proposed design method requires the specification of a specific model of a robotic system (a single- or a multi-robot system including cooperating devices) executing the predefined task. This model is produced on the basis of a universal model of a robotic system described below. In this approach robots are represented as conglomerates of embodied agents (possibly composed of just one such agent). As embodied agents are the most general forms of agents, out of them any robot system can be designed. The thus produced specification is used as a blueprint for the implementation of the system.

2.1. General inner structure of an Embodied Agent. A robotic system is represented as a set of agents a_j, $j = 1, \ldots, n_a$, where n_a is the number of agents (j designates a particular agent). Embodied agents have physical bodies interacting with the environment. This paper focuses on embodied agents [42], but all other agents can be treated as special cases with no body, thus the presentation is general. An embodied agent a_j, or simply an agent, possesses real effectors E_j, which exert influence over the environment, real receptors R_j (exteroceptors), which gather information from the surroundings, and a control system C_j that governs the actions of the

agent in such a way that its task will be executed. The exteroceptors of the agent a_j are numbered (or named), hence $R_{j,l}$, $l = 1,\ldots,n_R$, and so are its effectors $E_{j,h}$, $h = 1,\ldots,n_E$. Both the receptor readings and the effector commands undergo transformations into a form that is convenient from the point of view of the task, hence the virtual receptors r_j and virtual effectors e_j transform raw sensor readings and motor commands into abstract concepts required by the control subsystem to match the task formulation. Thus the control system C_j is decomposed into: virtual effectors $e_{j,n}$, $n = 1,\ldots,n_e$, virtual receptors $r_{j,k}$, $k = 1,\ldots,n_r$, and a single control subsystem c_j (Fig. 1). Virtual receptors perform sensor reading aggregation, consisting in either the composition of information obtained from several exteroceptors or in the extraction of the required data from one complex sensor (e.g. camera). Moreover the readings obtained from the same exteroceptors $R_{j,l}$ may be processed in different ways, so many virtual receptors $r_{j,k}$ can be formed. The control loop is closed through the environment, i.e. exteroceptor readings $R_{j,l}$ are aggregated by virtual receptors to be transmitted to the control subsystem c_j which generates appropriate commands for the virtual effectors e_j to translate into signals driving the effectors E_j. This primary loop is supplemented by links going in the opposite direction. The control subsystem c_j can both reconfigure exteroceptors R_j and influence the method how the virtual receptors r_j aggregate readings, thus a link from the control subsystem to the receptor emerges. The control subsystem also acquires proprioceptive data from the effectors. An agent through its control subsystem is able to establish a two-way communication with other agents $a_{j'}$, $j \neq j'$.

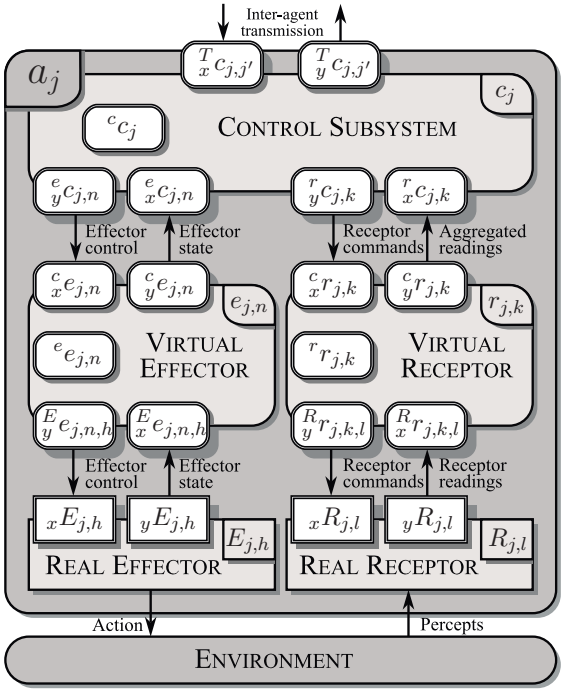

Fig. 1. Internal structure of an embodied agent a_j

The control subsystem as well as the virtual effectors and receptors use communication buffers to transmit or receive information to/from the other components (Fig. 1). A systematic denotation method is used to designate both the components and their buffers. To make the description of such a system concise no distinction is being made between the denotation of a buffer and its state (its content) – the context is sufficient. In the assumed notation a one-letter symbol located in the centre (i.e. E, R, e, r, c) designates the subsystem. To reference its buffers or to single out the state of this component at a certain instant of time extra indices are placed around this central symbol. The left superscript designates the subsystem to which the buffer is connected. The right superscript designates the time instant at which the state is being considered. The left subscript tells us whether this is an input (x) or an output (y) buffer. When the left subscript is missing the internal memory of the subsystem is referred to. The right subscript may be complex, with its elements separated by comas. They designate the particular: agent, its subsystem and buffer element. Buffer elements can also be designated by placing their names in square brackets. For instance ${}^{e}_{x}c^{i}_{j}[\text{pose}]$ denotes the contents of the variable "pose" located in the control subsystem input buffer of the agent a_j acquired from the virtual effector at instant i. Similarly functions are labelled. The central symbol for any function is f, the left superscript designates the type of the owner of the function and the type of the subsystem that this function produces the result of its computations for, the right superscript: τ, σ, ε refer to the terminal, initial and error conditions respectively (each one of them being a predicate). A missing right superscript denotes a transition function. The list of right subscripts designates a particular function, and the order of designators is: agent, subsystem, particular function. Finally, all the buffers may contain many variables – referred to within the square brackets. The ˜ symbol indicates the place holder in the buffer in which the variable is stored, whereas its lack refers to the value of that variable.

2.2. Types of agents. An embodied agent exhibits four types of activities: influencing the environment through effectors (both real and virtual treated in conjunction) – denoted by E, gathering the information from the environment through receptors (again both real and virtual) – denoted by R, transmission to/from the other agents – denoted by T and last but not least control of the agent (conducted by the control subsystem) – denoted by C. Out of the enumerated four activity types only C is indispensable. The agent can be deficient with respect to some of the others, thus seven useful types of agents emerge [93], i.e. an embodied agent with full capabilities is of the type CERT, a purely computational agent is of CT type, a monitoring agent is of a CR type, while a blind agent is of a CE type, a teleoperated agent is of a CET type, a remote sensor is of a CRT type, and an autonomous agent is of a CER type. An agent of C type is useless. Out of agents having T in the denotation of their type, networks of agents can be produced, thus diverse architectures can be represented. A single robot can be produced of one or many agents. Multi-robot systems require multiple agents. The architectures of robotic systems differ not only in structure, i.e. admissible connections between agents, but also in composition of agent types [93].

2.3. General subsystem behaviour. Fig. 2 presents the general work-cycle of any subsystem s_j, where $s \in \{c, e, r\}$, of an agent a_j. The designator of a concrete subsystem is the subscript u (there may be many virtual effectors and virtual receptors, thus this differentiation is necessary). The $s_{j\bullet}$, where $j^\bullet \in \{j, j'\}$, denotes all subsystems associated with s_j (the control subsystem of one agent may communicate with the control subsystem of another agent, hence j'). The functioning of a subsystem $s_{j,u}$ requires the processing of a transition function which uses as arguments the data contained in the input buffers ${}^x s_{j,u}$ and the memory ${}^s s_{j,u}$, to produce the output buffer values ${}^y s_{j,u}$ and new memory contents ${}^s s_{j,u}$. Hence the subsystem behaviour is described by a transition function ${}^s f_{j,u}$ defined as:

$$\left[{}^s s_{j,u}^{i+1}, {}^y s_{j,u}^{i+1} \right] := {}^s f_{j,u}({}^s s_{j,u}^{i}, {}^x s_{j,u}^{i}), \tag{1}$$

where i and $i+1$ are the consecutive discrete time stamps and $:=$ is the assignment operator. Function (1) describes the evolution of the state of a subsystem $s_{j,u}$. As a single function (1) would be too complex to define it in a monolithic form, thus it is usually decomposed into a set of partial functions:

$$\left[{}^s s_{j,u}^{i+1}, {}^y s_{j,u}^{i+1} \right] := {}^s f_{j,u,\xi}({}^s s_{j,u}^{i}, {}^x s_{j,u}^{i}), \tag{2}$$

where $\xi = 1, \ldots, n_{f_{s,u}}$, designates particular transition functions. Capabilities of the agent arise from the multiplicity, $n_{f_{s,u}}$, and diversity of the partial functions of its subsystems. Transition functions can be defined mathematically (e.g. [92]), or data flow diagrams can be used (e.g. [42]), if utmost precision is not required, however also pseudo-code can be used (e.g. [34]), if implementation is at the focus.

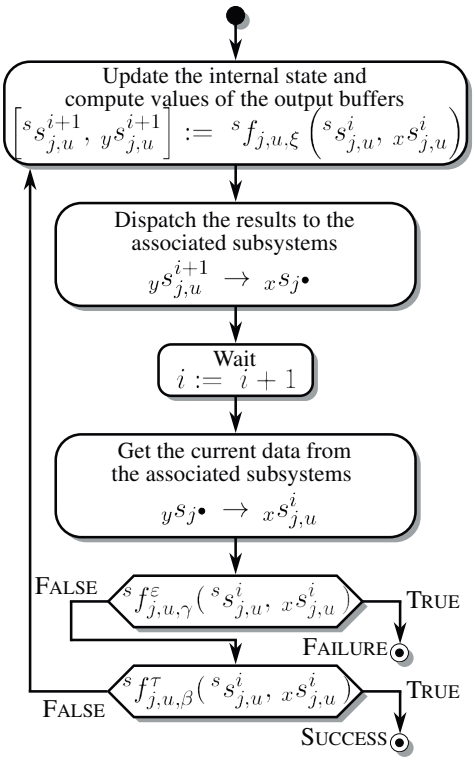

Fig. 2. General flow chart of a subsystem behaviour ${}^s\mathcal{B}_{j,u,\eta}({}^s f_{j,u,\xi}, {}^s f_{j,u,\beta}^{\tau}, {}^s f_{j,u,\gamma}^{\varepsilon})$, where $\bullet$ represents any subsystem including another agent

Such a prescription requires rules of switching between different partial transition functions of a subsystem, thus three additional Boolean valued functions (predicates) are required:

- ${}^s f_{j,u,\alpha}^{\sigma}$ defining the initial condition,
- ${}^s f_{j,u,\beta}^{\tau}$ representing the terminal condition and
- ${}^s f_{j,u,\gamma}^{\varepsilon}$ representing the error condition.

The first one selects the behaviour, while the second determines when the cyclic execution of the behaviour should terminate. The last one detects possible errors in the execution of the behaviour. Hence a multi-step evolution of the subsystem in a form of a behaviour $\mathcal{B}_{j,u,\xi}$ is defined as:

$${}^s\mathcal{B}_{j,u,\eta} \triangleq {}^s\mathcal{B}_{j,u,\eta}\left({}^s f_{j,u,\xi}, {}^s f_{j,u,\beta}^{\tau}, {}^s f_{j,u,\gamma}^{\varepsilon} \right). \tag{3}$$

It should be noted that a certain behaviour designated by η is defined in terms of functions designated by ξ, β and γ. This is because each of those functions belongs to a different set, and the behaviour can be defined as any three-element combination of functions picked from those sets (this facilitates the modularity of the resulting implementation of the control system).

The flowchart presented in Fig. 2 contains two blocks requiring the transfer of data between subsystems. The communication problem is not discussed in this paper, however it has been dealt with in [25, 85].

2.4. FSM governing behaviour switching. The activities of each subsystem are described by a finite state machine (FSM). Those activities are represented by a graph. Each node of this graph represents a state ${}^s\mathcal{S}_{j,u,\varepsilon}$ of the FSM, $\varepsilon = 1, \ldots, n_s$, where n_s is the number of states of this FSM. With each of the nodes of the graph a behaviour is associated. Thus in each state the FSM executes a single behaviour (this is a Moore type automaton). The nodes of the graph are connected by directed arcs labelled with initial conditions ${}^s f_{j,u,\alpha}^{\sigma}$. When the FSM is in its ${}^s\mathcal{S}_{j,u,\varepsilon}$ state and the terminal condition ${}^s f_{j,u,\beta}^{\tau}$ or error condition ${}^s f_{j,u,\gamma}^{\varepsilon}$ of the associated behaviour ${}^s\mathcal{B}_{j,u,\eta}$ is fulfilled, the FSM is ready to switch to its another state (thus to change the subsystem behaviour). If the arc of the graph of the FSM connecting states ${}^s\mathcal{S}_{j,u,\varepsilon}$ and ${}^s\mathcal{S}_{j,u,\varepsilon'}$ is labelled by the initial condition ${}^s f_{j,u,\alpha}^{\sigma}$, and this condition is fulfilled, the FSM will switch to the state ${}^s\mathcal{S}_{j,u,\varepsilon'}$. The state transition table of the FSM consists of two major parts. First part determines the current state and terminal (or error) condition terminating the current behaviour, whereas the second indicates the initial condition along with the next state. Each row of the table defines a single transition. It should be underscored that the transition takes place only when terminal/error condition of the associated behaviour and the initial condition of the destination state are fulfilled. If the terminal and error conditions of the behaviour associated with current state are not fulfilled the FSM stays in this state further executing the associated behaviour. Additionally, as a given behaviour can be associated with many different states, a second table is

introduced, mapping the states of the considered subsystem to its behaviours.

As an example a simple automaton of a subsystem $s_{j,u}$ exhibiting three behaviours is considered. Two of them, ${}^{s}\mathcal{B}_{j,u,0}$ and ${}^{s}\mathcal{B}_{j,u,1}$, are executed to execute the task under normal conditions. The third, ${}^{s}\mathcal{B}_{j,u,\varepsilon}$, is associated with error recovery. The error can occur only when the system is executing behaviour ${}^{s}\mathcal{B}_{j,u,1}$. Thus behaviour ${}^{s}\mathcal{B}_{j,u,0}$ terminates only when its terminal condition ${}^{s}f^{\tau}_{j,u,0}$ is fulfilled. Behaviour ${}^{s}\mathcal{B}_{j,u,1}$ terminates either if an error is detected by the error condition ${}^{s}f^{\varepsilon}_{j,u,1}$ or the terminal condition ${}^{s}f^{\tau}_{j,u,1}$ is fulfilled. It should be noted that according to the flowchart presented in Fig. 2 the priority of error detection is higher than that of terminal condition satisfaction, thus the value of the terminal condition is irrelevant when the error condition is fulfilled. Behaviour ${}^{s}\mathcal{B}_{j,u,\varepsilon}$ terminates when its terminal condition ${}^{s}f^{\tau}_{j,u,\varepsilon}$ is satisfied. Each of the mentioned behaviours is associated with one state of the FSM, as presented by Table 1.

Table 1
Mapping of the states of the exemplary subsystem to its behaviours

State	Behaviour	Description
${}^{s}\mathcal{S}_{\mathrm{j,u,0}}$	${}^{s}\mathcal{B}_{\mathrm{j,u,0}}$	First behaviour
${}^{s}\mathcal{S}_{\mathrm{j,u,1}}$	${}^{s}\mathcal{B}_{\mathrm{j,u,1}}$	Second behaviour
${}^{s}\mathcal{S}_{\mathrm{j,u,\varepsilon}}$	${}^{s}\mathcal{B}_{\mathrm{j,u,\varepsilon}}$	Error recovery

The graph of the FSM is presented in Fig. 3, while its state transition table in Table 2. The initial state is ${}^{s}\mathcal{S}_{\mathrm{j,u,0}}$. When its associated behaviour terminates the FSM transits to the state

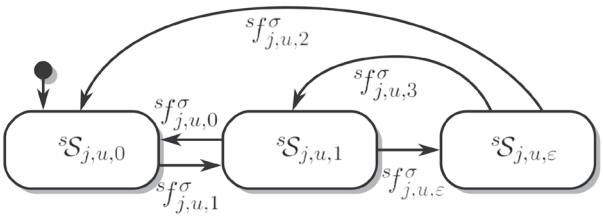

Fig. 3. Graph of an exemplary finite state automaton

Table 2
State transition table of the exemplary subsystem FSM

Current state		Next state	
State	Terminal/Error Condition	Initial Condition	State
${}^{s}\mathcal{S}_{\mathrm{j,u,0}}$	${}^{s}f^{\tau}_{\mathrm{j,u,0}} = \mathrm{TRUE}$	${}^{s}f^{\sigma}_{\mathrm{j,u,1}} = \mathrm{TRUE}$	${}^{s}\mathcal{S}_{\mathrm{j,u,1}}$
${}^{s}\mathcal{S}_{\mathrm{j,u,1}}$	${}^{s}f^{\tau}_{\mathrm{j,u,1}} = \mathrm{TRUE}$	${}^{s}f^{\sigma}_{\mathrm{j,u,0}} = \mathrm{TRUE}$	${}^{s}\mathcal{S}_{\mathrm{j,u,0}}$
${}^{s}\mathcal{S}_{\mathrm{j,u,1}}$	${}^{s}f^{\varepsilon}_{\mathrm{j,u,1}} = \mathrm{TRUE}$	${}^{s}f^{\sigma}_{\mathrm{j,u,\varepsilon}} = \mathrm{TRUE}$	${}^{s}\mathcal{S}_{\mathrm{j,u,\varepsilon}}$
${}^{s}\mathcal{S}_{\mathrm{j,u,\varepsilon}}$	${}^{s}f^{\tau}_{\mathrm{j,u,\varepsilon}} = \mathrm{TRUE}$	${}^{s}f^{\sigma}_{\mathrm{j,u,2}} = \mathrm{TRUE}$	${}^{s}\mathcal{S}_{\mathrm{j,u,0}}$
		${}^{s}f^{\sigma}_{\mathrm{j,u,3}} = \mathrm{TRUE}$	${}^{s}\mathcal{S}_{\mathrm{j,u,1}}$

${}^{s}\mathcal{S}_{\mathrm{j,u,1}}$ if the initial condition ${}^{s}f^{\sigma}_{j,u,1}$ is fulfilled. As it labels the only arc emerging from ${}^{s}\mathcal{S}_{\mathrm{j,u,0}}$ it must be always TRUE – for the sake of completeness of the graph. The behaviour associated with ${}^{s}f^{\sigma}_{j,u,1}$ can either terminate with an error, and then the FSM transits to ${}^{s}\mathcal{S}_{\mathrm{j,u,\varepsilon}}$ (in this case the initial condition ${}^{s}f^{\sigma}_{\mathrm{j,u,\varepsilon}}$ must be TRUE), or if no error has been detected it can terminate due to the satisfaction of the terminal condition ${}^{s}f^{\tau}_{j,u,1}$, and then the FSM transits to the state ${}^{s}\mathcal{S}_{\mathrm{j,u,0}}$, when the initial condition ${}^{s}f^{\sigma}_{j,u,0}$ is TRUE. When the error recovery behaviour terminates the FSM either transits to ${}^{s}\mathcal{S}_{\mathrm{j,u,0}}$ or ${}^{s}\mathcal{S}_{\mathrm{j,u,1}}$. In the former case when ${}^{s}f^{\sigma}_{j,u,2}$ is TRUE and in the latter case when ${}^{s}f^{\sigma}_{j,u,3}$ is fulfilled. Again, for the sake of completeness ${}^{s}f^{\sigma}_{j,u,2} \vee {}^{s}f^{\sigma}_{j,u,3} = TRUE$, and obviously ${}^{s}f^{\sigma}_{j,u,2} \wedge {}^{s}f^{\sigma}_{j,u,3} = FALSE$.

3. Design procedure

System design is always an iterative process. The design process progresses both by stepwise refinement and backtracking, where an increased detail of specification possibly requires modifications of the previous stages in the project. The presented procedure consists of a set of items rather than a list of consecutive steps. The designer selects both the level of details that he/she wants to express (i.e. ontological level) and the order of the design steps. The general structure of the system is as presented in Section 2. However this structure has to be tailored to the given task and the equipment that will be utilized.

The design procedure (Fig. 4) contains the following:

- Brief description of the scenario, which will further serve as a guide during assignment of roles of the agents and their subsystems;

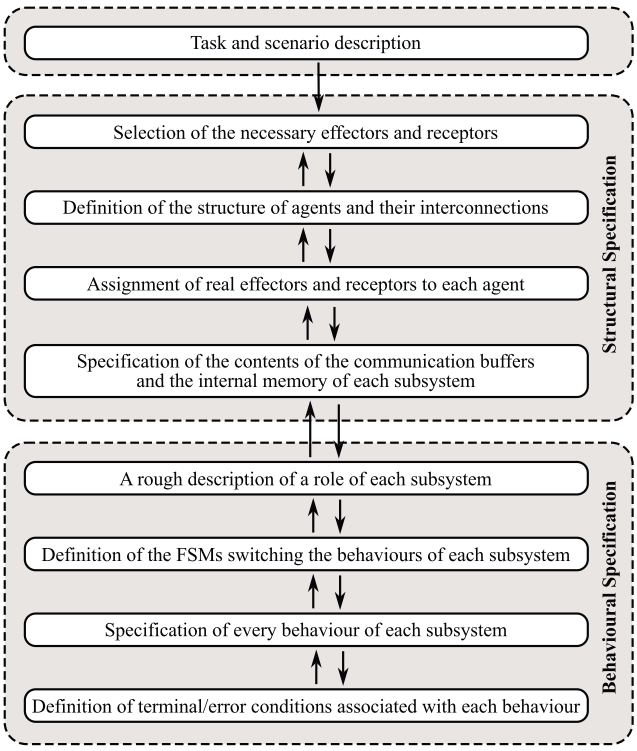

Fig. 4. System design procedure

- Selection of the necessary effectors and receptors, taking into account the class of tasks that is to be realised – those constitute the real effectors and receptors;
- Definition of the structure of the system in terms of agents and their interconnections, taking into account the considered task and the selected hardware. In the process of division into agents one has to take into account the transmission delays of inter-agent communication and the necessary computational power of the hardware which will execute the agent's code;
- Assignment of real effectors and receptors to each agent, taking into account the job that the agent has to execute;
- Specification of the contents of the communication buffers and the internal memory of each subsystem. It should be noted that the structure of the output buffer of one subsystem has to be exactly the same as that of the input buffer of the one it is connected to;
- A rough description of a role of each subsystem – it enables to distinguish the main required behaviours, their number and function. In here the designer should take into consideration the sampling rates not only of a given subsystem, but also of all the subsystems associated with it (as it will determine, e.g., how often new data will appear in its buffers);
- Definition of the FSMs switching the behaviours of subsystems – at this stage the designer needs to define the initial conditions of each of the behaviours associated with the FSM states – this results in a graph of transitions between states associated with the given subsystem behaviours;
- Specification of every behaviour of each subsystem, defined in terms of transition functions operating on the contents of buffers. As those transitions functions might be quite sophisticated, at this point the designer can create additional data-flow diagrams, facilitating the analytical definitions;
- Definition of terminal and error conditions associated with each behaviour.

The presented steps clearly indicate that there is an order of operations that the designer should follow. However, as it was mentioned, the above presented procedure is iterative, mainly due to the fact that as the designer dives deeper into the details of the operation of a given subsystem it might emerge that an important element has been missed in the previous step. For example, when defining a transition function of a given behaviour it might become obvious that additional information produced by yet another subsystem might be needed – a step back is required consisting in addition of information to the input/output buffers of both subsystems.

4. Example of application of the design method

The proposed design method is exemplified here by producing a control system of a robot capable of picking up an object. The object is located in a certain area, but its exact location is not known. A brief and easy to follow exemplary scenario is described in Section 4.1. The hardware that is necessary to execute it is deduced from the task formulation (Section 4.2). To formulate the task formally, initially the coordinate frames and transformations characterising the system need to be specified (Section 4.3). Then, the system structure is proposed in terms of an embodied agent (Section 4.4) taking into account both the hardware chosen to perform the task and the task scenario. The structure consists of a virtual effector controlling the gripper (Section 4.6), virtual effector controlling the manipulator (Section 4.5), control subsystem (Section 4.8) and virtual receptor aggregating data obtained from the RGB-D sensor (Section 4.7).

4.1. Description of the scenario. The following scenario is assumed. In the first step the system visually analyses the scene in order to recognize objects of interest and estimate their poses. Next it selects a single object of interest and generates adequate grasp poses (pre-grasp and grasp poses, grasping points etc.). Having those computed, it executes the grasp, by generation and realization of the approach trajectory, followed by grasping of the selected object. After that the object is picked-up and subsequently dropped at a random. Finally the manipulator returns to the initial pose and the whole procedure is repeated. The task is simple, yet it needs the majority of low-level skills required of a service robot, as it will become evident later. The resulting outline of the finite state machine is presented in Fig. 5.

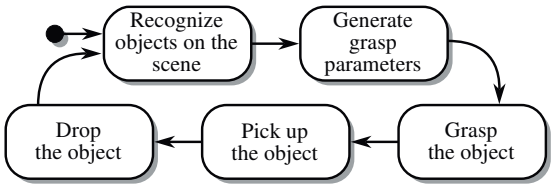

Fig. 5. Graph of the FSM of the object picking task

4.2. Hardware setup. To realise grasping both a manipulator equipped with a gripper and a camera to localise an object to be grasped are needed. The visual subsystem forms an exteroceptor (e.g. an RGB-D camera) that provides a rough localisation of the object. It is very difficult to localise the object so precisely to be able to execute pure position control of the manipulator and succeed. In consequence the manipulator, that forms the effector, should be equipped with extra force/torque or tactile sensors to execute position–force control suitable for interacting with the environment. As simple grasping suffices here the gripper does not need extra manipulation capabilities. A two-finger gripper, that is treated as another effector, is sufficient for the above scenario. The hardware setup is presented in Fig. 6. It consists of a modified IRb-6 serial manipulator (named IRp-6) having 6 DOF (Degrees Of Freedom), a two-fingered 1 DOF gripper, a 6 component force/torque sensor mounted in the wrist and a Kinect sensor placed above the scene (an immobile, stand-alone camera).

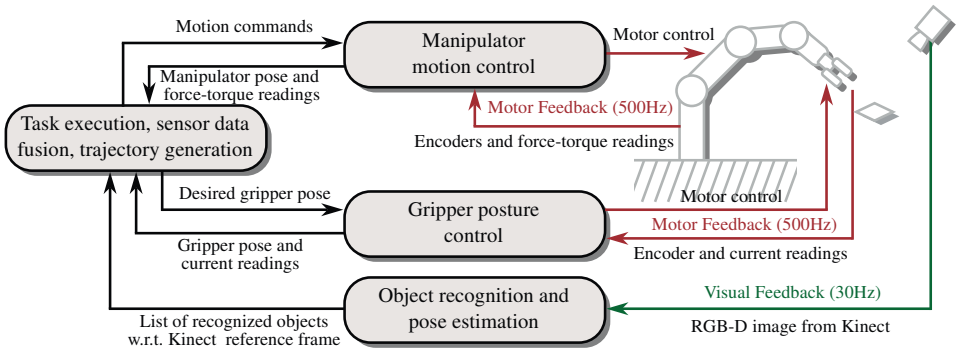

Fig. 6. General structure of the designed controller

4.3. Major coordinate reference frames and transformations. Fig. 7 indicates the most important (from the point of view of task execution) coordinate frames and transformations between them. B represents the robot base reference frame (treated in here as the global reference frame), E is the end-effector frame, K is the Kinect sensor reference frame, while v is the frame assigned to the v-th object (verified hypothesis of an object present in the scene). The most important transformations are: ${}^{B}_{K}T$ is the pose of the Kinect sensor with respect to the global reference frame (constant, computed during the calibration of the system), ${}^{B}_{E}T_{c}$ represents the current pose of the tip of the end-effector (computed on the basis of the current configuration of the manipulator joints – i.e. direct kinematics), whereas ${}^{K}_{v}T$, ${}^{E}_{v}T$ and ${}^{B}_{v}T$ represent the pose of the v-th object with respect to the: Kinect, end-effector and global reference frame respectively.

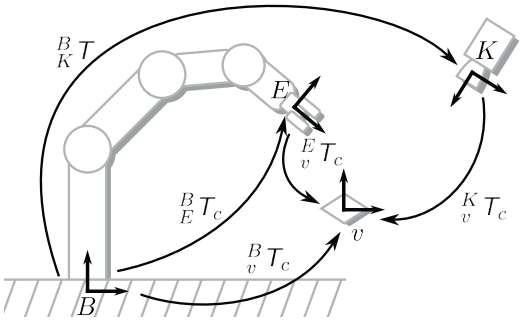

Fig. 7. Coordinate frames and transformations between them

4.4. Determination of the system structure. The structure of the system conforms to the structure represented by the general pattern of vision guided machines [29], where the control loop starts with the acquisition of the environment image, which is subsequently subjected to multi-phase processing, resulting in the formulation of a control decision commanding the actuators influencing the environment. The system structure is presented in Fig. 6. Here three control loops are distinguished: two inner motor loops controlling the manipulator and the gripper, and an outer visual loop responsible for recognition of objects of interest and estimation of their poses on the basis of RGB-D images received from the Kinect sensor.

The agent a_{irp} is responsible for controlling a modified IRb-6 manipulator (denoted as $E_{\text{irp,m}}$), its gripper ($E_{\text{irp,g}}$) and the Kinect sensor ($R_{\text{irp,k}}$), thus is classified as CERT – an embodied agent with full capabilities (please refer to Section 2.2). This suggests that the agent's control system should be decomposed into: the control subsystem c_{irp}, supplemented by two Virtual Effectors ($e_{\text{irp,m}}$ controlling the manipulator $E_{\text{irp,m}}$ and $e_{\text{irp,g}}$ controlling its gripper $E_{\text{irp,g}}$), and a Virtual Receptor $r_{\text{irp,k}}$ responsible for aggregation of sensoric data received from the Kinect sensor $R_{\text{irp,k}}$. The control subsystem c_{irp} is responsible for the realization of the object picking task, namely for storing the state of the scene (consisting of the recognized objects along with their poses), selection of the object of interest, generation of the the grasp and for picking of the object, whereas the virtual entities $e_{\text{irp,m}}$, $e_{\text{irp,g}}$ and $r_{\text{irp,k}}$ form a hardware abstraction layer, forming an interface between the control subsystem and the real hardware, thus simplifying the formulation and hence the realization of the task. The structure of the resulting embodied agent is presented in Fig. 8.

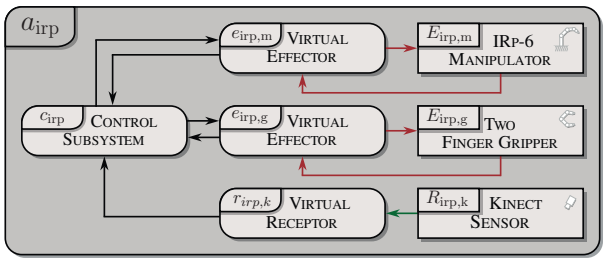

Fig. 8. Internal structure of the agent controlling the IRp-6 manipulator, the two-finger gripper and the Kinect sensor

4.5. Virtual effector controlling the manipulator. Virtual effector $e_{\text{irp,m}}$ controls the modified IRp-6 manipulator and exhibits several behaviours, offering diverse control abstractions. However in this paper, only two are exploited: the first realising the Point to Point (P2P) motion and the second implementing the position-force control. Besides that an idle behaviour is defined. It is executed in the case when the control subsystem does not send any control commands to the manipulator.

We decided that from the point of view of the task realized by the control system the most convenient form of abstraction

of the real effector will be its Cartesian pose. This will enable the control subsystem to focus on the Cartesian (operational) space and express commands as Cartesian coordinates. Despite that, the trajectory still can be interpolated in several ways. In here, to keep it simple, we decided to use interpolation in the motor space. This implied that both the P2P motion and idle behaviours could relay on a single (inner) control loop operating on the motor values, responsible for reaching the desired positions. The position-force control, however, needs an extra outer control loop, utilizing the wrench measurements for the computation of the desired end-effector Cartesian poses, being subsequently transformed into the desired values used by the inner control loop.

Internal structure. Fig. 9 presents the inner structure of the virtual effector $e_{\text{irp,m}}$. The input buffer associated with the control subsystem can be divided into two subsets, being used depending on the type of control/behaviour. In P2P control mode only a single component of the input buffer is used:

- ${}^{c}_{x}e_{\textbf{irp,m}}$ $[{}^{B}_{E}\widetilde{T}_d]$ – desired Cartesian pose of the end-effector (E) w.r.t. the robot base (B),

whereas in position-force control mode the following components of the input buffer are used:

- ${}^{c}_{x}e_{\textbf{irp,m}}$ $[\widetilde{b}]$ – the operational modes of the positional force-controllers (for details please refer to Section 4.5),
- ${}^{c}_{x}e_{\textbf{irp,m}}$ $[\widetilde{F}_d]$ – the desired forces exerted by the manipulator,
- ${}^{c}_{x}e_{\textbf{irp,m}}$ $[\widetilde{V}_d]$ – the desired manipulator velocity,
- ${}^{c}_{x}e_{\textbf{irp,m}}$ $[\widetilde{D}_d]$ – the desired value of damping,
- ${}^{c}_{x}e_{\textbf{irp,m}}$ $[\widetilde{I}_d]$ – the desired value of inertia,

where here and throughout the paper the $\widetilde{\ }$ symbol indicates the place holder in a given buffer. It is worth noting that all the variables related to position-force control contain six elements, with the former three containing the translational components of the motion (along the X, Y and Z axes) and the latter three containing rotational components (around the X, Y and Z axes), respectively, e.g., $\widetilde{F}_d = \left[{}_x\widetilde{F}_d, {}_y\widetilde{F}_d, {}_z\widetilde{F}_d, {}_{ax}\widetilde{F}_d, {}_{ay}\widetilde{F}_d, {}_{az}\widetilde{F}_d\right]$. For more details please refer to Section 4.5 devoted to that type of control.

The output buffers to the control subsystem contains:

- ${}^{c}_{y}e_{\textbf{irp,m}}$ $[{}^{B}_{E}\widetilde{T}_c]$ – current Cartesian pose of the end–effector w.r.t. the robot base (B) (a homogeneous matrix),
- ${}^{c}_{y}e_{\textbf{irp,m}}$ $[{}^{E}\widetilde{F}_c]$ – current force/torque exerted by the end-effector w.r.t. the end-effector reference frame E.

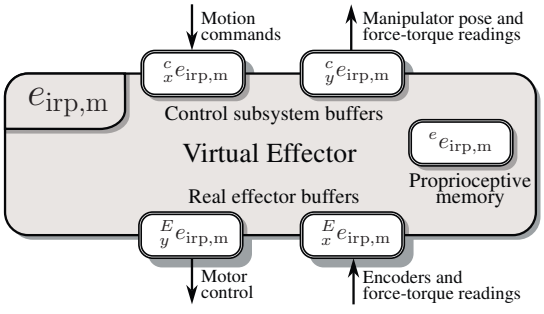

Fig. 9. Inner structure of the virtual effector $e_{\text{irp,m}}$

The buffers connected to the real effector are defined as:

- ${}^{E}_{x}e_{\textbf{irp,m}}$ $[\widetilde{m}_m]$ – currently measured motor positions,
- ${}^{E}_{y}e_{\textbf{irp,m}}$ $[\widetilde{c}_d]$ – desired values of current driving the manipulator motors.

Finally, the virtual effector has to store in its memory the following variables:

- ${}^{e}e_{\textbf{irp,m}}$ $[\widetilde{m}_d]$ – desired motor absolute positions,
- ${}^{e}e_{\textbf{irp,m}}$ $[\widetilde{q}_p]$ – previous joint configuration, which is used for the selection of the appropriate solution of the inverse kinematic problem.

Finite state automaton of the virtual effector. Having several behaviours appropriate initial conditions are needed to switch between them. The conditions for the ${}^{e}\mathcal{B}_{\text{irp,m,1}}$ behaviour realising P2P motion are quite obvious. The behaviour is activated when the control subsystem sends the desired Cartesian pose:

$$ {}^{e}f^{\sigma}_{\text{irp,m,1}}\left({}^{c}_{x}e^{\text{l}}_{\text{irp,m}}\right) \triangleq \mathcal{P}\left({}^{c}_{x}e_{\text{irp,m}}[{}^{B}_{E}\widetilde{T}_d]\right), \tag{4}$$

and is executed until the motion is completed. As a result a similar condition can be formulated as in the case of the ${}^{e}\mathcal{B}_{\text{irp,g,1}}$ behaviour of the virtual effector controlling the gripper, terminating the behaviour when the current values will be equal to the desired ones. However, in this case the values are expressed in the operational space, i.e.:

$$ {}^{e}f^{\tau}_{\text{irp,m,1}}\left({}^{E}_{x}e^{\text{l}}_{\text{irp,m}}, {}^{c}_{x}e^{\text{l}}_{\text{irp,m}}\right) \triangleq \left({}^{B}_{E}T_c = {}^{B}_{E}T_d\right). \tag{5}$$

No special error conditions are assumed here:

$$ {}^{e}f^{\varepsilon}_{\text{irp,m,1}}() \triangleq \text{FALSE}. \tag{6}$$

The behaviour ${}^{e}\mathcal{B}_{\text{irp,m,2}}$ (associated with the state ${}^{e}\mathcal{S}_{\text{irp,m,2}}$) realizes the position-force control. It becomes active when the control subsystem sends a command starting the position-force control, determining in fact the modes of operation along all six motion components:

$$ {}^{e}f^{\sigma}_{\text{irp,m,2}}\left({}^{c}_{x}e^{\text{l}}_{\text{irp,m}}\right) \triangleq \mathcal{P}\left({}^{c}_{x}e_{\text{irp,m}}[\widetilde{b}]\right) \wedge \left(\forall_{\xi}\, \xi b \in \{\mathfrak{u}, \mathfrak{c}, \mathfrak{g}\}\right), \tag{7}$$

where ξ denotes one of the six motion components (three linear and three angular) and $\mathfrak{u}$, $\mathfrak{c}$, $\mathfrak{g}$, $\mathfrak{s}$ denote different modes of the operation of the position-force control regulators. In particular, the $\mathfrak{s}$ mode indicates that the regulator for a given motion component should be stopped (turned off). For detailed explanation of the former three modes along with the principles of operation of position-force control please refer to Section 4.5. In here, for the brevity of the lecture, we must however explain the reasons for the introduction of the stop mode $\mathfrak{s}$. In fact, there is a multitude of possible termination conditions of that behaviour, which might result from different modes of the utilization of force sensing and decomposition of the motion into six independent components, each of which requiring a separate terminal condition etc. Experience implied that it is most convenient to send the current proprioceptive information (consisting of the end-effector pose and currently measured

forces) to the control subsystem and let it decide when to terminate the position-force control. This resulted in the introduction the stop mode, being one method of terminating the position-force control on the virtual effector side.

Additionally, we also allow the control system to send the desired Cartesian pose – in this case the virtual effector switches from the behaviour ${}^{e}\mathcal{B}_{\text{irp,m,1}}$ realising the position-force control to ${}^{e}\mathcal{B}_{\text{irp,m,1}}$ realising a simple P2P motion.

We also allow a third type of termination – in the case when the control subsystem sends new parameters for the position-force controllers along with one of the previously mentioned modes $(\mathfrak{u}, \mathfrak{c}, \mathfrak{g})$. This results in fulfilling of the condition (7) and activation of the position-force control once again. Thus the terminal condition is formulated as:

$$ {}^{e}f^{\tau}_{\text{irp,m,2}}\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}\right) \triangleq \mathcal{P}\left({}^{c}_{x}e_{\text{irp,m}}[\widetilde{b}]\right) \vee \mathcal{P}\left({}^{c}_{x}e_{\text{irp,m}}[{}^{B}_{E}\widetilde{T}_{d}]\right). \quad (8) $$

The first part of the formulated above condition checks whether there are some new values in the buffer containing the operation modes, whereas it is the role of initial conditions of all three behaviours to select the right transition. In this case we also assume that no special error condition is needed, thus:

$$ {}^{e}f^{\varepsilon}_{\text{irp,m,2}}() \triangleq \text{FALSE}. \quad (9) $$

At the end we have to define the conditions for the idle behaviour ${}^{e}\mathcal{B}_{\text{irp,m,0}}$. The behaviour becomes active when there is no new desired Cartesian pose in the buffer ${}^{c}_{x}e_{\text{irp,m}}[{}^{B}_{E}\widetilde{T}_{d}]$, there are no new parameters of the position-force control ${}^{c}_{x}e_{\text{irp,m}}[\widetilde{b}]$ or when a command for stopping the position-force control was received, i.e. $\mathcal{P}\left({}^{c}_{x}e_{\text{irp,m}}[\widetilde{b}]\right) \wedge \left(\exists \xi\, \xi b \in \{\mathfrak{s}\}\right)$. Thus the initial condition is defined as:

$$ \begin{aligned} {}^{e}f^{\sigma}_{\text{irp,m,0}}\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}\right) \triangleq {} & \neg\mathcal{P}\left({}^{c}_{x}e_{\text{irp,m}}[{}^{B}_{E}\widetilde{T}_{d}]\right) \vee \neg\mathcal{P}\left({}^{c}_{x}e_{\text{irp,m}}[\widetilde{b}]\right) \\ & \vee \left(\mathcal{P}\left({}^{c}_{x}e_{\text{irp,m}}[\widetilde{b}]\right) \wedge \left(\exists \xi\, \xi b = \mathfrak{s}\right)\right). \end{aligned} \quad (10) $$

Typically for idle behaviours, it lasts one step:

$$ {}^{e}f^{\tau}_{\text{irp,m,0}}() \triangleq \text{TRUE} \quad (11) $$

and error conditions do not occur, thus:

$$ {}^{e}f^{\varepsilon}_{\text{irp,m,0}}() \triangleq \text{FALSE}. \quad (12) $$

The resulting graph of the behaviour selection automaton is presented in Fig. 10. All the behaviours were mapped one to one to the states of the FSM, as presented in Table 3. Table 4 describes transitions between those states, being equivalent to the FSM from Fig. 10.

Table 3
Mapping of the states of the virtual effector to its behaviours

State	Behaviour	Description
${}^{e}\mathcal{S}_{\text{irp,m,0}}$	${}^{e}\mathcal{B}_{\text{irp,m,0}}$	Idle
${}^{e}\mathcal{S}_{\text{irp,m,1}}$	${}^{e}\mathcal{B}_{\text{irp,m,1}}$	Execute P2P motion
${}^{e}\mathcal{S}_{\text{irp,m,2}}$	${}^{e}\mathcal{B}_{\text{irp,m,2}}$	Execute position-force cont.

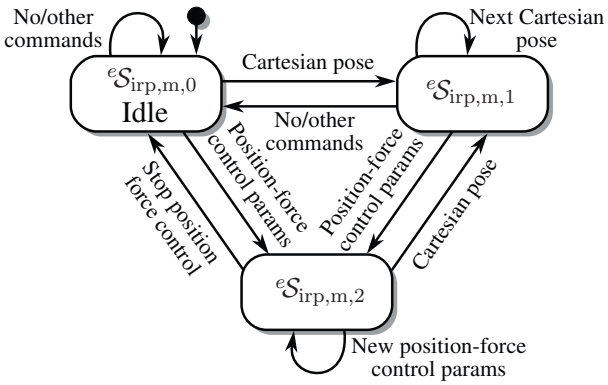

Fig. 10. Graph of the finite state automaton of the virtual effector controlling the manipulator

Table 4
State transition table of the virtual effector FSM (T. denotes TRUE)

Current state		Next state	
State	Terminal/ Error Cond.	Initial condition	State
${}^{e}\mathcal{S}_{\text{irp,m,0}}$	${}^{e}f^{\tau}_{\text{irp,m,0}} = \text{T.}$	${}^{e}f^{\sigma}_{\text{irp,m,0}} = \text{T.}$	${}^{e}\mathcal{S}_{\text{irp,m,0}}$
		${}^{e}f^{\sigma}_{\text{irp,m,1}} = \text{T.}$	${}^{e}\mathcal{S}_{\text{irp,m,1}}$
		${}^{e}f^{\sigma}_{\text{irp,m,2}} = \text{T.}$	${}^{e}\mathcal{S}_{\text{irp,m,2}}$
${}^{e}\mathcal{S}_{\text{irp,m,1}}$	${}^{e}f^{\tau}_{\text{irp,m,1}} = \text{T.}$	${}^{e}f^{\sigma}_{\text{irp,m,0}} = \text{T.}$	${}^{e}\mathcal{S}_{\text{irp,m,0}}$
		${}^{e}f^{\sigma}_{\text{irp,m,1}} = \text{T.}$	${}^{e}\mathcal{S}_{\text{irp,m,1}}$
		${}^{e}f^{\sigma}_{\text{irp,m,2}} = \text{T.}$	${}^{e}\mathcal{S}_{\text{irp,m,2}}$
${}^{e}\mathcal{S}_{\text{irp,m,2}}$	${}^{e}f^{\tau}_{\text{irp,m,2}} = \text{T.}$	${}^{e}f^{\sigma}_{\text{irp,m,0}} = \text{T.}$	${}^{e}\mathcal{S}_{\text{irp,m,0}}$
		${}^{e}f^{\sigma}_{\text{irp,m,1}} = \text{T.}$	${}^{e}\mathcal{S}_{\text{irp,m,1}}$
		${}^{e}f^{\sigma}_{\text{irp,m,2}} = \text{T.}$	${}^{e}\mathcal{S}_{\text{irp,m,2}}$

Behaviour ${}^{e}\mathcal{B}_{\text{irp,m,0}}$. When there are no commands from c_{irp}, the $e_{\text{irp,m}}$ activates the default behaviour ${}^{e}\mathcal{B}_{\text{irp,m,0}}$ responsible for holding the manipulator in the same configuration (by sending the same desired motor positions to the real effector) and returning the proprioceptive information to the control subsystem. The idle behaviour ${}^{e}\mathcal{B}_{\text{irp,m,0}}$ is defined as:

$$ {}^{e}\mathcal{B}_{\text{irp,m,0}} \triangleq {}^{e}\mathcal{B}_{\text{irp,m,0}}\left({}^{e,E}f_{\text{irp,m,0}}, {}^{e,c}f_{\text{irp,m,0}}, {}^{e}f^{\tau}_{\text{irp,m,0}}, {}^{e}f^{\varepsilon}_{\text{irp,m,0}}\right). \quad (13) $$

Data flow of the idle behaviour is presented in Fig. 11. There are two major data flows: first associated with the effector control function ${}^{e,E}f_{\text{irp,m,0}}$ and second with the proprioceptive function ${}^{e,c}f_{\text{irp,m,0}}$.

The effector control function acquires through ${}^{E}_{x}e_{\text{irp,m}}$ the current (measured) encoder–based positions of motors m_c. Taking into account the desired positions m_d (that are stored in the

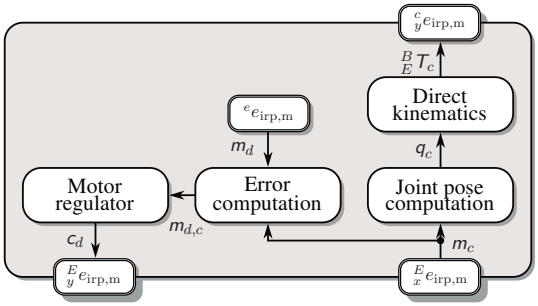

Fig. 11. Data flow diagram of the computations performed by the idle behaviour ${}^{e}\mathcal{B}_{\text{irp,m,0}}$

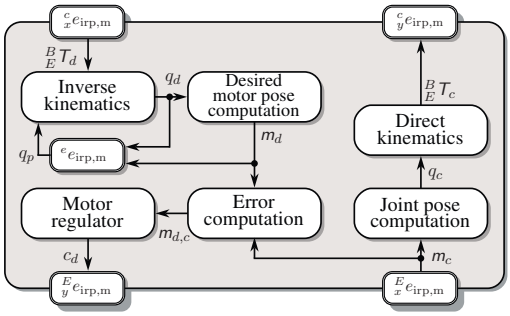

Fig. 12. Data flow diagram of the computations performed by the behaviour ${}^{e}\mathcal{B}_{\text{irp,m,1}}$

memory) it computes the error $m_{d,c} = m_d - m_c$. Holding the manipulator still simply means that the error $m_{d,c}$ must be reduced to zero. So it is thus passed to the regulator $\mathcal{MR}$, which computes the desired value of motor currents, i.e. c_d, sent via ${}^{E}_{y}e_{\text{irp,m}}$ to the real effector motors. Hence ${}^{e,E}f_{\text{irp,m,0}}$ is defined as:

$$
\begin{aligned}
{}^{E}_{y}e^{t+1}_{\text{irp,m}}[\widetilde{c}_d] &:= {}^{e,E}f_{\text{irp,m,0}}({}^{e}e^{t}_{\text{irp,m}}, {}^{E}_{x}e^{t}_{\text{irp,m}}) \triangleq \\
&\triangleq \mathcal{MR}(m_d - m_c).
\end{aligned}
\tag{14}
$$

Proprioceptive function is responsible for returning the current end-effector pose to the control subsystem. For this purpose is takes the vector of current motor positions m_c, transforms it into joint positions q_c ($\mathcal{MJ}$ stands for Motor to Joint) and uses manipulator direct kinematics ($\mathcal{DK}$) to compute the pose of the end-effector with respect to the robot base. The function can be formulated analytically as:

$$
\begin{aligned}
{}^{c}_{y}e^{t+1}_{\text{irp,m}}[{}^{B}_{E}\widetilde{T}_c] &:= {}^{e,c}f_{\text{irp,m,0}}\left({}^{E}_{x}e^{t}_{\text{irp,m}}\right) \triangleq \\
&\triangleq \mathcal{DK}\left(\mathcal{MJ}\left(m_c\right)\right).
\end{aligned}
\tag{15}
$$

Behaviour ${}^{e}\mathcal{B}_{\text{irp,m,1}}$. The goal of this behaviour is to control the manipulator motion in order to reach the desired pose, expressed as a homogeneous matrix containing a Cartesian pose with respect to the robot base reference frame. This can be realised in several ways, e.g. by linear interpolation [86] or spherical linear interpolation (SLERP) [67]. To deal with kinematic singularities, the motor space interpolation was chosen. The algorithm is similar to the one for the behaviour ${}^{e}\mathcal{B}_{\text{irp,g,1}}$ controlling the posture of the gripper.

Besides active control of the manipulator the behaviour also computes the current Cartesian pose of the end-effector and provides it to c_{irp}. Moreover, at the onset it memorizes the desired joint pose (so it can be used in the next step for the selection of the solution of inverse kinematics) as well as the desired motor pose (which in turn is used by the idle behaviour).

The postulated decomposition requires the specification of: effector proprioceptive function ${}^{e,c}f_{\text{irp,m,1}}$, real effector control function ${}^{e,E}f_{\text{irp,m,1}}$ and virtual effector memory function ${}^{e,e}f_{\text{irp,m,1}}$, what implies that ${}^{e}\mathcal{B}_{\text{irp,m,1}}$ is defined as:

$$
\begin{aligned}
{}^{e}\mathcal{B}_{\text{irp,m,1}} \triangleq {}^{e}\mathcal{B}_{\text{irp,m,1}}({}^{e,E}f_{\text{irp,m,1}}, {}^{e,c}f_{\text{irp,m,1}}, {}^{e,e}f_{\text{irp,m,1}}, \\
{}^{e}f^{\tau}_{\text{irp,m,1}}, {}^{e}f^{\varepsilon}_{\text{irp,m,1}}),
\end{aligned}
\tag{16}
$$

The goal of the function ${}^{e,E}f_{\text{irp,m,1}}$ is to control the motion of the manipulator, i.e. produce adequate values of the current sent to the motors through the whole time of behaviour execution. The control subsystem delivers, through the virtual effector buffer ${}^{c}_{x}e_{\text{irp}}$, the desired end-effector pose ${}^{B}_{E}T_d$. The homogeneous matrix ${}^{B}_{E}T_d$ and the memorized previous joint positions q_p are fed into the inverse kinematics ($\mathcal{IK}$) to obtain the desired joint positions q_d. The desired joint positions are used to compute the desired motor positions m_d, which are subsequently compared with the current motor positions m_c in order to compute the error in the motor space $m_{d,c} = m_d - m_c$. This error is reduced by regulator $\mathcal{MR}$, which returns the desired value of motor currents c_d, sent through ${}^{E}_{y}e_{\text{irp,m}}$ to the motors. The resulting definition of the function is:

$$
\begin{aligned}
{}^{E}_{y}e^{t+1}_{\text{irp,m}}[\widetilde{c}_d] &:= {}^{e,E}f_{\text{irp,m,1}}({}^{e}e^{t}_{\text{irp,m}}, {}^{c}_{x}e^{t}_{\text{irp,m}}, {}^{E}_{x}e^{t}_{\text{irp}}) \triangleq \\
&\triangleq \mathcal{MR}\left(\mathcal{JM}\left(\mathcal{IK}\left({}^{B}_{E}T_d, q_p\right)\right) - m_c\right).
\end{aligned}
\tag{17}
$$

The proprioceptive function ${}^{e,c}f_{\text{irp,m,1}}$ has to return the homogeneous matrix ${}^{B}_{E}T_c$ (the pose of the end-effector E with respect to the global reference frame B) to the control subsystem. For this it acquires through ${}^{E}_{x}e_{\text{irp,m}}$ the current positions of motors m_c and converts them into joint positions q_c ($\mathcal{MJ}$) to subsequently compute the homogeneous matrix ${}^{B}_{E}T_c$ by solving the direct kinematics problem ($\mathcal{DK}$):

$$
{}^{c}_{y}e^{t+1}_{\text{irp,m}}[{}^{B}_{E}\widetilde{T}_c] := {}^{e,c}f_{\text{irp,m,1}}\left({}^{E}_{x}e^{t}_{\text{irp,m}}\right) \triangleq \mathcal{DK}\left(\mathcal{MJ}\left(m_c\right)\right). \tag{18}
$$

Finally, some of the results computed along the presented above dataflow additionally have to be stored in the memory. So we have to define two partial memory update functions:

$$
\begin{aligned}
{}^{e}e^{t+1}_{\text{irp,m}} &:= {}^{e,e}f_{\text{irp,m,1}}\left({}^{e}e^{t}_{\text{irp,m}}, {}^{c}_{x}e^{t}_{\text{irp,m}}\right) \triangleq \\
&\triangleq \begin{cases} {}^{e,e}f_{\text{irp,m,1,1}}\left({}^{e}e^{t}_{\text{irp,m}}, {}^{c}_{x}e^{t}_{\text{irp,m}}\right), \\ {}^{e,e}f_{\text{irp,m,1,2}}\left({}^{e}e^{t}_{\text{irp,m}}, {}^{c}_{x}e^{t}_{\text{irp,m}}\right), \end{cases}
\end{aligned}
\tag{19}
$$

being responsible for remembering the current manipulator joint configuration:

$$
\begin{aligned}
{}^{e}e^{t+1}_{\text{irp,m}}[\widetilde{q}_p] &:= {}^{e,e}f_{\text{irp,m,1,1}}\left({}^{e}e^{t}_{\text{irp,m}}, {}^{c}_{x}e^{t}_{\text{irp,m}}\right) \triangleq \\
&\triangleq \mathcal{IK}\left({}^{B}_{E}T_d, q_p\right)
\end{aligned}
\tag{20}
$$

as well as for memorization of the desired motor pose:

$$ {}^{e}e^{t+1}_{\text{irp,m}}[\widetilde{m}_d] := {}^{e,e}f_{\text{irp,m,1,2}}\left({}^{e}e^{t}_{\text{irp,m}}, {}^{c}_{x}e^{t}_{\text{irp,m}}\right) \triangleq \\ \triangleq \mathcal{JM}\left(\mathcal{IK}\left({}^{B}_{E}T_d, q_p\right)\right). \tag{21} $$

Behaviour ${}^{e}\mathcal{B}_{\text{irp,m,2}}$. The control law of the behaviour ${}^{e}\mathcal{B}_{\text{irp,m,2}}$ was described in details in [80, 92] and extends the control algorithms from previous works e.g. [66]. The main principle of the operation stems from the Task Frame Formalism [54, 17] or operational space formulation [38] and decomposes the motion into three translational (x, y, z) and three rotational (a_x, a_y, a_z) components (angle and axis representation). The resulting six regulators are parameterized by: desired force/torque F_d exerted on the object, desired velocity V_d of the end effector, damping D_d and inertia I_d. Each of those vectors has 6 components indicated by subscript ξ, where $\xi \in \{x, y, z, a_x, a_y, a_z\}$. Units of those parameters differ depending on the component (Table 5). Each of the decoupled regulators has three modes of operation, depending on the relation between the robot and the object of manipulation, i.e.:

- $\mathfrak{u}$ – unguarded motion (pure position control, without interaction with the environment),
- $\mathfrak{c}$ – pure force control (contact with the environment),
- $\mathfrak{g}$ – guarded motion (position control anticipating contact).

Each mode uses a slightly different subset of parameters, some being user defined, and some being constant, as presented in Table 6. Both the values of parameters and modes can be different for regulators controlling different motion components.

Table 5
Units of the parameters of the position-force generator for different motion components: x, y, z denote translational components, whereas a_x, a_y, a_z denote rotational components

Motion component	Force	Vel.	Damp.	Iner.
	${}_{\xi}F_d$	${}_{\xi}V_d$	${}_{\xi}D_d$	${}_{\xi}I_d$
$\xi \in \{x, y, z\}$	[N]	$\left[\frac{\text{m}}{\text{s}}\right]$	$\left[\frac{\text{kg}}{\text{s}}\right]$	[kg]
$\xi \in \{a_x, a_y, a_z\}$	[N m]	$\left[\frac{1}{\text{s}}\right]$	$\left[\frac{\text{kg m}^2}{\text{s}}\right]$	[kg m^2]

Table 6
Values of parameters for different operation modes ${}_{\xi}b$ of the position-force regulators. + denotes the value sent by the control subsystem, 0 and ∞ are the constant values produced by the virtual effector in response to the commanded mode

Mode	Force	Velocity	Damping	Inertia
${}_{\xi}b$	${}_{\xi}F_d$	${}_{\xi}V_d$	${}_{\xi}D_d$	${}_{\xi}I_d$
$\mathfrak{c}$	+	0	+	+
$\mathfrak{g}$	0	+	+	+
$\mathfrak{u}$	0	+	∞	0

The behaviour ${}^{e}\mathcal{B}_{\text{irp,m,2}}$ is defined as:

$$ {}^{e}\mathcal{B}_{\text{irp,m,2}} \triangleq {}^{e}\mathcal{B}_{\text{irp,m,2}}\left({}^{e,E}f_{\text{irp,m,2}}, {}^{e,c}f_{\text{irp,m,2}}, {}^{e,e}f_{\text{irp,m,2}}, \right. \\ \left. {}^{e}f^{\tau}_{\text{irp,m,2}}, {}^{e}f^{\varepsilon}_{\text{irp,m,2}}\right), \tag{22} $$

with the transition functions defined below.

Data flow diagram of ${}^{e}\mathcal{B}_{\text{irp,m,2}}$ is presented in Fig. 13. One can distinguish two main flow directions: top-down, responsible for control of the real effector, and bottom-up, responsible for producing the proprioceptive data for c_{irp}.

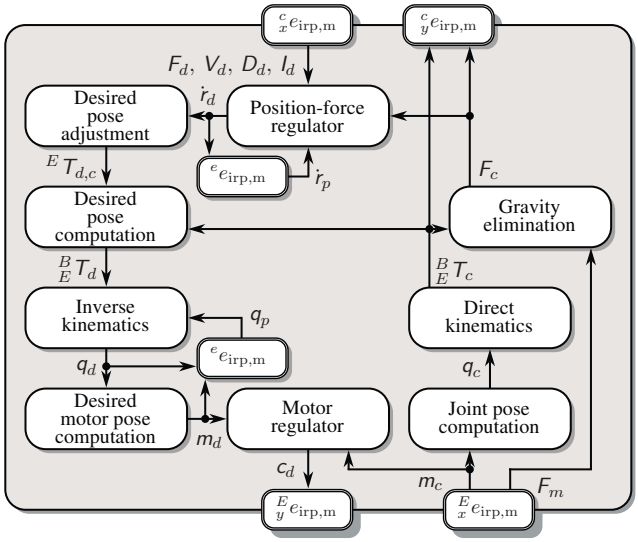

Fig. 13. Data flow diagram of the behaviour ${}^{e}\mathcal{B}_{\text{irp,m,2}}$

First the proprioceptive bottom-up flow will be considered, because the results produced by it are also used by the other flows. This flow is spoilt into two paths, so the proprioceptive transition function is defined as two partial functions:

$$ {}^{c}_{y}e^{t+1}_{\text{irp,m}} := {}^{e,c}f_{\text{irp,m,2}}\left({}^{E}_{x}e^{t}_{\text{irp,m}}\right) \triangleq \begin{cases} {}^{e,c}f_{\text{irp,m,2,1}}\left({}^{E}_{x}e^{t}_{\text{irp,m}}\right), \\ {}^{e,c}f_{\text{irp,m,2,2}}\left({}^{E}_{x}e^{t}_{\text{irp,m}}\right). \end{cases} \tag{23} $$

The first proprioceptive partial function ${}^{e}f_{\text{irp,m,2,1}}$ is responsible for producing the current end-effector pose. The pose is computed by solving the direct kinematics ($\mathcal{DK}$) for the current joint configuration $r_c = \mathcal{DK}(q_c)$, which, in turn, is obtained by transformation of the current motor increments received from the motor encoders, i.e. $q_c = \mathcal{MJ}(m_c)$, where $\mathcal{MJ}$ denotes the motor to joint transformation:

$$ {}^{c}_{y}e^{t+1}_{\text{irp,m}}[{}^{B}_{E}\widetilde{T}_c] := {}^{e,c}f_{\text{irp,m,2,1}}\left({}^{E}_{x}e^{t}_{\text{irp,m}}\right) \triangleq \\ \triangleq \mathcal{DK}(\mathcal{MJ}(m_c)), \tag{24} $$

whereas the second partial function ${}^{e}f_{\text{irp,m,2,2}}$ computes the currently exerted force, calculated on the basis of the measured force F_m exerted by the end-effector, after the elimina-

tion of the influence of gravity ($\mathcal{GE}$). The function ${}^{e}f_{\text{irp,m,2,2}}$ takes into account the the current orientation of the end-effector $F_c = \mathcal{GE}\left(F_m, {}^{B}_{E}T_c\right)$:

$$\begin{aligned} {}^{c}_{y}e^{\iota+1}_{\text{irp,m}}[\widetilde{F}_c] &:= {}^{e,c}f_{\text{irp,m,2,2}}\left({}^{E}_{x}e^{\iota}_{\text{irp,m}}\right) \triangleq \\ &\triangleq \mathcal{GE}\left(F_m, \mathcal{DK}\left(\mathcal{MJ}\left(m_c\right)\right)\right). \end{aligned} \tag{25}$$

The presented computations are also used by the effector control function. In particular, the current force F_c is transferred, along with the parameters of the position-force control taken from the input buffer ${}^{c}_{x}e_{\text{irp,m}}$, to the position-force regulator $\mathcal{PF}$. The regulator, in fact consisting of six independent regulators for each of the six motion components ξ, is responsible for the computation of the desired velocity $\dot{r}_d$ of the end-effector in the operational space:

$$\begin{aligned} {}_{\xi}\dot{r}_d &:= \mathcal{PF}\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}, {}^{e}e^{\iota}_{\text{irp,m}}\right) \triangleq \\ &\triangleq \mathcal{PF}\left({}_{\xi}F_d, {}_{\xi}V_d, {}_{\xi}D_d, {}_{\xi}I_d, {}_{\xi}\dot{r}_p, {}_{\xi}F_c\right) \triangleq \\ &\triangleq \frac{\left(\frac{1}{{}_{\xi}D_d}({}_{\xi}F_d - {}_{\xi}F_c) + {}_{\xi}V_d\right)\Delta\iota + \frac{1}{{}_{\xi}D_d}\,{}_{\xi}I_d\,{}_{\xi}\dot{r}_p}{\Delta\iota + \frac{1}{{}_{\xi}D_d}\,{}_{\xi}I_d}, \end{aligned} \tag{26}$$

where the first four parameters $({}_{\xi}F_d, {}_{\xi}V_d, {}_{\xi}D_d, {}_{\xi}I_d)$ are the position-force control parameters sent by the control subsystem, $\dot{r}_p$ is the desired velocity vector computed in the previous step of the controller operation, whereas $\Delta\iota$ denotes the motion time (in our case equal to 2 ms, as the frequency of the operation of the virtual effector is 500 Hz). The idea of using previous desired pose r_p instead of taking the current value r_c during the motion realization is explained in details in [92].

On the basis of the end-effector velocity $\dot{r}_d$ the pose increment is computed and subsequently transformed from angle-axis into homogeneous matrix representation ${}^{E}T_{d,c} = \mathcal{AA}(\dot{r}_d * \Delta\iota)$. The resulting pose increment in a homogeneous matrix form is next used along with the current end-effector pose ${}^{B}_{E}T_c$ for computation of the desired end-effector pose ${}^{B}_{E}T_d = {}^{B}_{E}T_c\,{}^{E}T_{d,c}$. The desired end-effector pose is subsequently, together with the joint configuration from the previous step q_p, used for solving the inverse kinematics: $q_d = \mathcal{IK}({}^{B}_{E}T_d, q_p)$. The desired joint pose is transformed into the motor pose $m_d = \mathcal{JM}(q_d)$. The the so obtained motor pose and the current motor pose m_c retrieved from the ${}^{E}_{x}e_{\text{irp,m}}$ buffer, are transferred to the motor regulator $\mathcal{MR}$. The current c_d produced by the regulator is subsequently sent through the ${}^{E}_{y}e_{\text{irp,m}}$ buffer to the hardware controlling the manipulator motors. The function ${}^{e,E}f_{\text{irp,m,2}}$ controlling the real effector is defined as:

$$\begin{aligned} {}^{E}_{y}e^{\iota+1}_{\text{irp,m}}[\widetilde{c}_d] := {}^{e,E}f_{\text{irp,m,2}}&\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}, {}^{E}_{x}e^{\iota}_{\text{irp,m}}, {}^{e}e^{\iota}_{\text{irp,m}}\right) \triangleq \\ &\mathcal{MR}(\mathcal{JM}(\mathcal{IK}(\mathcal{DK}(\mathcal{MJ}(m_c)) \\ &\mathcal{AA}(\mathcal{PF}(F_d, V_d, D_d, I_d, \dot{r}_p, \\ &\mathcal{GE}(F_m, \mathcal{DK}(\mathcal{MJ}(m_c))))) * \Delta\iota), q_p)), m_c). \end{aligned} \tag{27}$$

Additionally in each step, in the virtual effector memory the desired velocity of the end-effector along with the desired joint configuration must be stored, so they can be used as the previous values in the next control step. Besides that, the desired motor values m_d must also be memorized, so they can be used by the idle behaviour, thus:

$$\begin{aligned} {}^{e}e^{\iota+1}_{\text{irp,m}} &:= {}^{e,e}f_{\text{irp,m,2}}\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}, {}^{e}e^{\iota}_{\text{irp,m}}, {}^{E}_{x}e^{\iota}_{\text{irp,m}}\right) \triangleq \\ &\triangleq \begin{cases} {}^{e,e}f_{\text{irp,m,2,1}}\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}, {}^{E}_{x}e^{\iota}_{\text{irp,m}}\right), \\ {}^{e,e}f_{\text{irp,m,2,2}}\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}, {}^{e}e^{\iota}_{\text{irp,m}}, {}^{E}_{x}e^{\iota}_{\text{irp,m}}\right), \\ {}^{e,e}f_{\text{irp,m,2,3}}\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}, {}^{e}e^{\iota}_{\text{irp,m}}, {}^{E}_{x}e^{\iota}_{\text{irp,m}}\right), \end{cases} \end{aligned} \tag{28}$$

being defined as:

$$\begin{aligned} {}^{e}e^{\iota+1}_{\text{irp,m}}[\widetilde{r}_p] &:= {}^{e,e}f_{\text{irp,m,2,1}}\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}, {}^{E}_{x}e^{\iota}_{\text{irp,m}}\right) \triangleq \\ &\mathcal{PF}(F_d, V_d, D_d, I_d, \dot{r}_p, \mathcal{GE}(F_m, \mathcal{DK}(\mathcal{MJ}(m_c)))), \end{aligned} \tag{29}$$

$$\begin{aligned} {}^{e}e^{\iota+1}_{\text{irp,m}}[\widetilde{q}_p] &:= {}^{e,e}f_{\text{irp,m,2,2}}\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}, {}^{e}e^{\iota}_{\text{irp,m}}, {}^{E}_{x}e^{\iota}_{\text{irp,m}}\right) \triangleq \\ &\triangleq \mathcal{IK}(\mathcal{DK}(\mathcal{MJ}(m_c)) \\ &\mathcal{AA}(\mathcal{PF}(F_d, V_d, D_d, I_d, \dot{r}_p, \\ &\mathcal{GE}(F_m, \mathcal{DK}(\mathcal{MJ}(m_c)))) * \Delta\iota), q_p). \end{aligned} \tag{30}$$

and:

$$\begin{aligned} {}^{e}e^{\iota+1}_{\text{irp,m}}[\widetilde{m}_p] &:= {}^{e,e}f_{\text{irp,m,2,3}}\left({}^{c}_{x}e^{\iota}_{\text{irp,m}}, {}^{e}e^{\iota}_{\text{irp,m}}, {}^{E}_{x}e^{\iota}_{\text{irp,m}}\right) \triangleq \\ &\triangleq \mathcal{JM}(\mathcal{IK}(\mathcal{DK}(\mathcal{MJ}(m_c)) \\ &\mathcal{AA}(\mathcal{PF}(F_d, V_d, D_d, I_d, \dot{r}_p, \\ &\mathcal{GE}(F_m, \mathcal{DK}(\mathcal{MJ}(m_c)))) * \Delta\iota)\, q_p)). \end{aligned} \tag{31}$$

4.6. Virtual effector controlling the gripper. Virtual effector $e_{\text{irp,g}}$ is responsible for controlling the gripper. Both its structure and activities are a gross simplification of the virtual effector $e_{\text{irp,m}}$, thus will not be described here in detail. However, as the control subsystem c_{irp} uses the contents of the input and output buffers of $e_{\text{irp,g}}$ connected to it, they have to be described here. Virtual effector $e_{\text{irp,g}}$ input buffer connected to the control subsystem contains:

${}^{c}_{x}e_{\textbf{irp,g}}$ $[\widetilde{d}_d]$ – desired gripper pose, defined as the distance between two fingers.

The output to the control subsystem contains:

${}^{c}_{y}e_{\textbf{irp,g}}$ $[\widetilde{d}_c]$ – current distance between the gripper fingers,

${}^{c}_{y}e_{\textbf{irp,g}}$ $[\widetilde{c}_c]$ – measured motor current, proportional to the force exerted by the fingers on the grasped object.

4.7. Virtual receptor representing the RGB-D sensor. The virtual receptor $r_{\text{irp,k}}$ is responsible for the recognition of household, rich textured objects. It is also supposed to estimate

their poses, enabling their subsequent grasping and manipulation. Thus besides object detection the emphasis is placed on robust estimation of object poses in Cartesian space. Such a goal can be achieved in many ways [32], e.g. using classical computer vision approaches like LINE-MOD [31] or MOPED (Multiple Object Pose Estimation and Detection) [21]. One can also utilize approaches relying on recent achievements in machine/deep learning, e.g. SSD-6D [37] that combines an feature encoding based on the Inception-V4 architecture [75] with SSD-style prediction [48] or 3D Point-Capsule Networks [81] merging capsule networks [62] with PointNets [60].

However, as we wanted to keep the example simple and focus readers attention on the design method, we decided to utilize to solution that, similarly to MOPED, relies on local features extracted from RGB images. Additionally, we decided to use depth information that enables to transform features coordinates into 3D (Cartesian) space. Utilization of 3D information results in more robust feature matching and produces much better estimates of object poses, being the natural side-effects of clustering of the found correspondences. This enabled the simplification of the MOPED pipeline by, e.g., skipping the cluster merging or pose refinement steps, what resulted in increase of the speed of the whole procedure.

Models of objects of interest. The Object recognition relies on matching of point (local) features of the 3D models of objects and features extracted from the current view of the scene. Thus it is required that the system should posses such 3D models of objects. Those models can be generated using solutions such as MODREG [40]. Models used in our experiments are publicly available as part of the WUT Visual Perception Dataset [72], examples are presented in Fig. 14.

Fig. 14. Exemplary models of objects from 3D Models Dataset extracted from WUT Visual Perception Dataset [72]: (a) Herbapol Mint Tea; (b) Foodcan 2; (c) Lipton Yellow Label Tea

At start-up the system loads a set of M object models:

$$_{1..M}C = \left({}_1C, \ldots, {}_MC \right), \tag{32}$$

where each model ${}_mC$, $m = 1, \ldots, M$, consists of two types of point clouds: a dense colour cloud ${}_mC^{\mathrm{RGB}}$ and a sparse cloud of features ${}_mC^{\mathrm{SIFT}}$ (Fig. 15). The system relies on SIFT features (Scale Invariant Feature Transform) [49] as they are still one of the most robust and discriminative features, thus:

$$_mC = \left({}_mC^{\mathrm{RGB}}, {}_mC^{\mathrm{SIFT}}, {}_mi \right), \tag{33}$$

where ${}_mi$ represents id (label) of a given model.

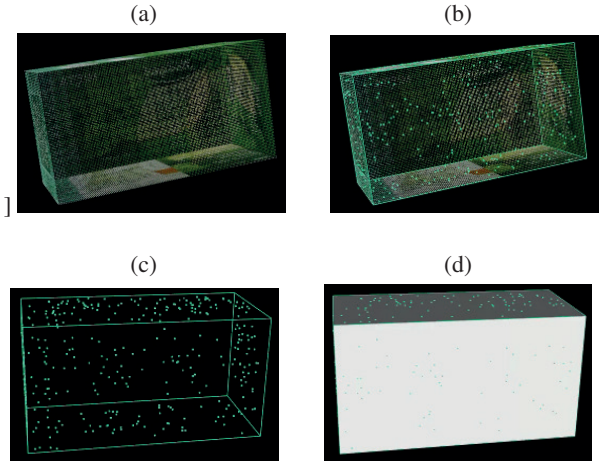

Fig. 15. Exemplary object (Mint Tea) visualizations: (a) RGB point cloud; (b) RGB and SIFT point clouds with a bounding box (bb); (c) SIFT point cloud with a bb; (d) SIFT point cloud with a bb and meshes representing object faces

Recognized objects. It is assumed that in a given (current) image frame (consisting of an RGB image I_c^{RGB} and a depth map I_c^{D}) the perception subsystem can recognize several objects at the same time:

$$_{1..V}^{K}O_c = \left({}_1^KO_c, \ldots, {}_V^KO_c \right). \tag{34}$$

Every object ${}_v^KO_c$, $v = 1, \ldots, V$, is a tuple, representing a single verified hypothesis, with respect to the Kinect frame K:

$${}_v^KO_c = \left({}_v^KC_c^{\mathrm{RGB}}, {}_v^KC_c^{\mathrm{SIFT}}, {}_v^KT_c, {}_vc_c, {}_vi_c \right). \tag{35}$$

${}_v^KC_c^{\mathrm{RGB}}$ and ${}_v^KC_c^{\mathrm{SIFT}}$ represent object model point clouds projected onto the scene (Kinect frame), ${}_v^KT_c$ is the object pose, whereas ${}_vc_c$ represents the object recognition confidence, being a ratio between the number of matches (found between the features of the scene and the given model) and the total number of features of the considered given model. Finally, ${}_vi_c$ stores the object id (label), being a concatenation of the model id and the consecutive number of the recognized object.

Internal structure of the virtual receptor. Fig. 16 presents the inner structure of the Virtual Receptor $r_{\mathrm{irp,k}}$. Its input buffer receiving data from the Kinect sensor contains:

${}_x^Rr_{\mathbf{irp,k}}$ $[\widetilde{I}_c^{\mathrm{RGB}}]$ – RGB image (image with three channels),

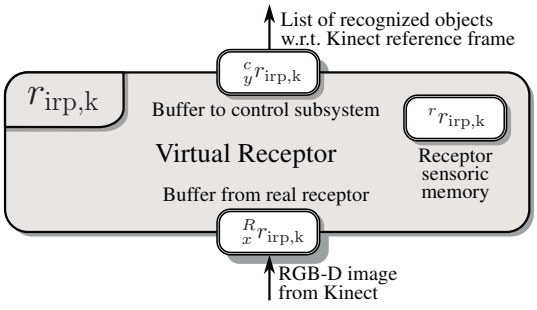

Fig. 16. Inner structure of the virtual receptor $r_{\mathrm{irp,k}}$

${}^{R}_{x}r_{\mathbf{irp,k}}$ $[\widetilde{I}^{\mathrm{D}}_{c}]$ – depth map (a single-channel image),
whereas the output to the control subsystem contains:
${}^{c}_{y}r_{\mathbf{irp,k}}$ $[{}^{K}_{1..V}\widetilde{O}_{c}]$ – list of verified object hypothesis w.r.t the Kinect reference frame.
Besides that, the receptor holds in its memory the following:
${}^{r}r_{\mathbf{irp,k}}$ $[{}^{}_{1..M}\widetilde{C}]$ – list of object models,
${}^{r}r_{\mathbf{irp,k}}$ $[\widetilde{P}]$ – intrinsic parameters of the Kinect sensor (camera matrix and distortion coefficients).

Finite state automaton of the virtual receptor. It was assumed that the virtual receptor has to recognize objects and estimate their poses in each RGB-D image acquired from the Kinect sensor independently. Hence an adequate behaviour had to be defined, namely ${}^{r}\mathcal{B}_{\mathrm{irp,k,1}}$, triggered by the presence of data acquired from the Kinect sensor. The ${}^{r}\mathcal{B}_{\mathrm{irp,k,1}}$ behaviour is triggered by the presence of RGB-D image in the input buffer ${}^{R}_{x}r_{\mathrm{irp,k}}$ from the real receptor, thus the initial condition activating the behaviour is defined as follows:

$$ {}^{r}f^{\sigma}_{\mathrm{irp,k,1}} \triangleq \mathcal{P}\left({}^{R}_{x}r_{\mathrm{irp,k}}[\widetilde{I}^{\mathrm{RGB}}_{c}]\right) \wedge \mathcal{P}\left({}^{R}_{x}r_{\mathrm{irp,k}}[\widetilde{I}^{\mathrm{D}}_{c}]\right), \tag{36} $$

where $\mathcal{P}$ denotes the presence of fresh data in a given buffer. It is assumed that the behaviour takes one step:

$$ {}^{r}f^{\tau}_{\mathrm{irp,k,1}} \triangleq \mathrm{TRUE}, \tag{37} $$

and that there is no special error condition required:

$$ {}^{r}f^{\varepsilon}_{\mathrm{irp,k,1}} \triangleq \mathrm{FALSE}. \tag{38} $$

Additionally, an idle behaviour ${}^{r}\mathcal{B}_{\mathrm{irp,k,0}}$ must be defined. It is executed when the input buffer from the real receptor has no new data:

$$ \begin{aligned} {}^{r}f^{\sigma}_{\mathrm{irp,k,0}} &\triangleq \neg\mathcal{P}\left({}^{R}_{x}r_{\mathrm{irp,k}}[\widetilde{I}^{\mathrm{RGB}}_{c}]\right) \vee \neg\mathcal{P}\left({}^{R}_{x}r_{\mathrm{irp,k}}[\widetilde{I}^{\mathrm{D}}_{c}]\right) = \\ &= \neg\,{}^{r}f^{\sigma}_{\mathrm{irp,k,1}}. \end{aligned} \tag{39} $$

Here it is assumed that this behaviour ends after a single step:

$$ {}^{r}f^{\tau}_{\mathrm{irp,k,0}} \triangleq \mathrm{TRUE}, \tag{40} $$

and that there are no special error conditions:

$$ {}^{r}f^{\varepsilon}_{\mathrm{irp,k,0}} \triangleq \mathrm{FALSE}. \tag{41} $$

The resulting FSM (Fig. 17) consists of two states: ${}^{r}\mathcal{S}_{\mathrm{irp,k,0}}$ associated with the idle behaviour ${}^{r}\mathcal{B}_{\mathrm{irp,k,0}}$ and state ${}^{r}\mathcal{S}_{\mathrm{irp,k,1}}$ associated with ${}^{r}\mathcal{B}_{\mathrm{irp,k,1}}$. Table 7 and Table 8 present state–behaviour association and state transitions respectively.

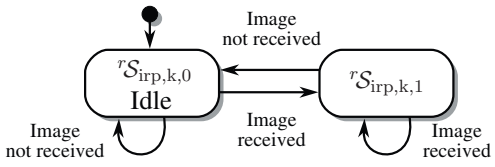

Fig. 17. Graph of the finite state automaton realising the behaviour selection of the $r_{\mathrm{irp,k}}$ virtual receptor

Table 7
Mapping of the states of the virtual receptor to its behaviours

State	Behaviour	Description
${}^{r}\mathcal{S}_{\mathrm{irp,k,0}}$	${}^{r}\mathcal{B}_{\mathrm{irp,k,0}}$	Idle
${}^{r}\mathcal{S}_{\mathrm{irp,k,1}}$	${}^{r}\mathcal{B}_{\mathrm{irp,k,1}}$	Object recognition

Table 8
State transition table of the virtual receptor FSM after reduction (T. denotes TRUE)

Current state		Next state	
State	Terminal/Error Condition	Initial Condition	State
${}^{r}\mathcal{S}_{\mathrm{irp,k,0}}$	${}^{r}f^{\tau}_{\mathrm{irp,k,0}}$ = T.	${}^{r}f^{\sigma}_{\mathrm{irp,k,0}}$ = T.	${}^{r}\mathcal{S}_{\mathrm{irp,k,0}}$
		${}^{r}f^{\sigma}_{\mathrm{irp,k,1}}$ = T.	${}^{r}\mathcal{S}_{\mathrm{irp,k,1}}$
${}^{r}\mathcal{S}_{\mathrm{irp,k,1}}$	${}^{r}f^{\tau}_{\mathrm{irp,k,1}}$ = T.	${}^{r}f^{\sigma}_{\mathrm{irp,k,0}}$ = T.	${}^{r}\mathcal{S}_{\mathrm{irp,k,0}}$
		${}^{r}f^{\sigma}_{\mathrm{irp,k,1}}$ = T.	${}^{r}\mathcal{S}_{\mathrm{irp,k,1}}$

Behaviour ${}^{r}\mathcal{B}_{\mathbf{irp,k,0}}$. When there is no image to process, the virtual receptor enters the idle state and executes behaviour ${}^{r}\mathcal{B}_{\mathrm{irp,k,0}}$. The behaviour is not responsible for any computations and depends only on the terminal and error conditions, defined as (40) and (41) respectively. In short, the idle behaviour terminates after a single step, disregarding the state of its buffers, memory etc. So ${}^{r}\mathcal{B}_{\mathrm{irp,k,1}}$ is simply defined as:

$$ {}^{r}\mathcal{B}_{\mathrm{irp,k,0}} \triangleq {}^{r}\mathcal{B}_{\mathrm{irp,k,0}}\left({}^{r}f^{\varepsilon}_{\mathrm{irp,k,0}}, {}^{r}f^{\tau}_{\mathrm{irp,k,0}}\right). \tag{42} $$

Behaviour ${}^{r}\mathcal{B}_{\mathbf{irp,k,1}}$. The main goal of this behaviour is to aggregate the sensory data, thus aside of the previously defined terminal (37) and error (38) conditions the reading aggregation function ${}^{r,c}f_{\mathrm{irp,k,1}}$ must be defined. As a result the ${}^{r}\mathcal{B}_{\mathrm{irp,k,1}}$ behaviour is defined as:

$$ {}^{r}\mathcal{B}_{\mathrm{irp,k,1}} \triangleq {}^{r}\mathcal{B}_{\mathrm{irp,k,1}}\left({}^{r,c}f_{\mathrm{irp,k,1}}, {}^{r}f^{\varepsilon}_{\mathrm{irp,k,1}}, {}^{r}f^{\tau}_{\mathrm{irp,k,1}}\right). \tag{43} $$

Dataflow diagram of the ${}^{r,c}f_{\mathrm{irp,k,1}}$ function responsible for object recognition is presented in Fig. 18. The input to the vir-

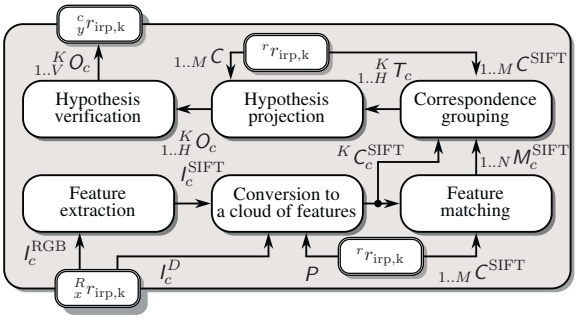

Fig. 18. Data flow diagram of the virtual receptor reading aggregation function ${}^{r,c}f_{\mathrm{irp,k,1}}$ responsible for model-based object recognition in RGB-D images

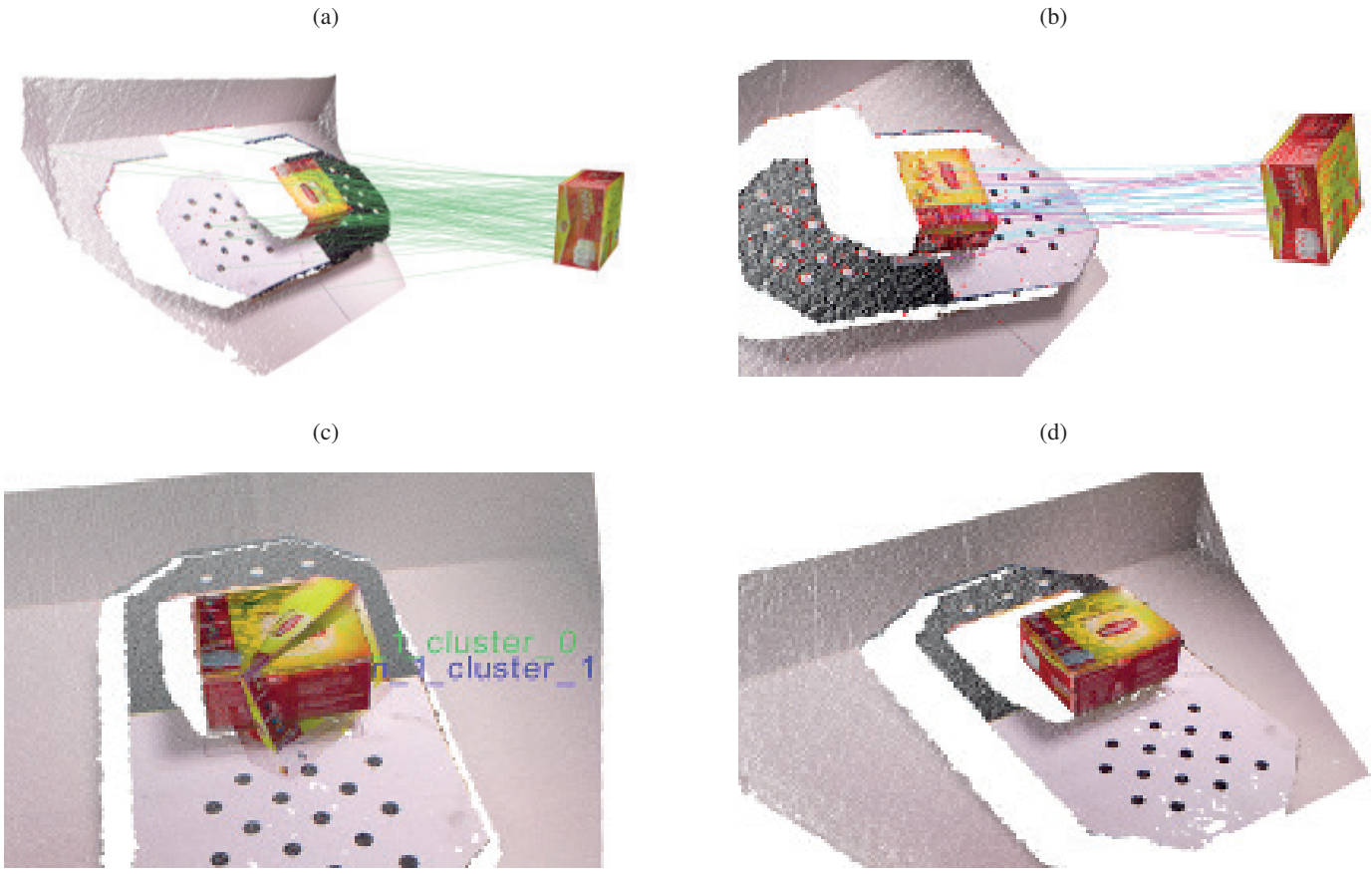

Fig. 19. Visualization of the consecutive steps of the recognition of a single object (Lipton Yellow Label Tea) utilising a scene chosen from Test Scenes Dataset from WUT Visual Perception Dataset [72]: (a) feature matching, (b) correspondence grouping, (c) hypothesis projection (two hypotheses are visible) and (d) verification (weaker hypothesis, that was in conflict, was rejected)

tual receptor consists of an RGB image I_c^{RGB} along with the associated depth map I_c^{D}, obtained from the Kinect sensor via the ${}^{R}_{x}r_{\mathrm{irp,k}}$ buffer. The first step of the recognition procedure involves extraction of SIFT features ($\mathcal{FE}$) from the I_c^{RGB} image:

$$I_c^{\mathrm{SIFT}} = \mathcal{FE}\left(I_c^{RGB}\right). \tag{44}$$

Subsequently their coordinates are transformed from the image to the Cartesian space ($\mathcal{IC}$). This is done with the use of their known distances from the sensor (the depth map I_c^{D}) and intrinsic parameters of the Kinect sensor P. This operation results in a sparse cloud of SIFT features with Cartesian coordinates with respect to the Kinect sensor reference frame (K superscript), representing the scene:

$${}^{K}C_c^{\mathrm{SIFT}} = \mathcal{IC}\left(I_c^{\mathrm{SIFT}}, I_c^{D}, P\right). \tag{45}$$

The goal of the following step, consisting in feature matching ($\mathcal{FM}$), is the determination of the correspondence between the scene and the object models. This is done by matching the descriptors of features extracted from the scene ${}^{K}C_c^{\mathrm{SIFT}}$ with the descriptors of features of all object models stored in the virtual receptor memory ${}_{1..M}C^{\mathrm{SIFT}}$:

$${}_{1..N}M_c^{\mathrm{SIFT}} = \mathcal{FM}\left({}^{K}C_c^{\mathrm{SIFT}}, {}_{1..M}C^{\mathrm{SIFT}}\right), \tag{46}$$

where N is the number of found model–scene feature matches (in particular, there can be several correspondences found between a given scene feature and features of different models). As the comparison is made in a high-dimensional space (SIFT descriptor is a set of 128 elements), for feature matching FLANN (Fast Library for Approximate Nearest Neighbours) [56] was used. FLANN is an efficient implementation of the k-Nearest Neighbours algorithm. Exemplary results of correspondence estimation are presented in Fig. 19a (red dots indicate scene features, green dots indicate model features and green lines represent found mutual correspondences).

Taking into account that in the scene there might be several instances of an object belonging to the same class (thus matching the same model), as well as that objects belonging to many different classes have to be recognized simultaneously, it was necessary to formulate object hypotheses, based on clusters of correspondences constituting different objects. For that purpose the Geometric Consistency Grouping algorithm was applied. It is based on the proposal presented in [19], which groups correspondences ($\mathcal{CG}$) on the basis of relations between point features belonging to the model and the scene:

$${}^{K}_{1..H}T_c = \mathcal{CG}\left({}_{1..N}M_c^{\mathrm{SIFT}}, {}^{K}C_c^{\mathrm{SIFT}}, {}_{1..M}C^{\mathrm{SIFT}}\right), \tag{47}$$

where ${}_{1..H}^{K}T_c$ represents the set of poses of found hypotheses of objects with respect to the Kinect frame:

$$ {}_{1..H}^{K}T_c = \left\{ {}_{1}^{K}T_c, \ldots, {}_{H}^{K}T_c \right\}, \tag{48} $$

where H is the number of hypotheses. In opposition to the correspondence estimation, during correspondence grouping only the Cartesian coordinates of the features are taken into consideration. Each object hypothesis clusters the correspondences with similar transformation between the model points and the scene points. If there is a correspondence that fits the given hypothesis (i.e. when the projection of the model point using hypothesis transformation matches the scene point of the considered correspondence) then this correspondence is added to the cluster. An exemplary result of correspondence grouping is presented in Fig. 19b. It should be noted that there are two colours of correspondences, which indicate the presence of two different clusters, hence two different object hypotheses.

The next step is responsible for hypothesis projection ($\mathcal{HP}$), being the transformation of the point clouds belonging to the models associated with each of the hypotheses (both dense colour point cloud and sparse feature cloud) into the Kinect sensor reference frame. For each projection the transformation associated with the given hypothesis, found in the correspondence grouping step, was used:

$$ {}_{1..V}^{K}O_c = \mathcal{HP}\left({}_{1..H}^{K}T_c, {}_{1..M}C^{\mathrm{RGB}}, {}_{1..M}C^{\mathrm{SIFT}} \right). \tag{49} $$

The resulting set contains V object hypotheses, with each v-th object hypothesis defined according to (35). Exemplary projections are presented in Fig. 19c.

Having several hypotheses it is necessary to reject the ones that are weak and/or are in conflict with other hypotheses. In the presented system it was decided to incorporate the Greedy Verification algorithm [2]. This method counts the number of features belonging to the given projection of the model, that fit the scene points – disregarding whether there was underlying (i.e. earlier found) correspondence for a given pair or not. The hypothesis is considered valid if the number of inliers is greater than the outliers (the required ratio may be parameterised). The hypothesis verification ($\mathcal{HV}$) step is defined as:

$$ {}_{1..V}^{K}O_c = \mathcal{HV}\left({}_{H}^{K}O_c \right). \tag{50} $$

The resulting list of objects ${}_{1..V}^{K}O_c$ (with each object defined according to (35)) is subsequently conveyed to the control subsystem through the ${}_{y}^{c}r_{\mathrm{irp,k}}$ buffer. Thus the definition of the transition function is as follows:

$$ \begin{aligned} {}_{y}^{c}r_{\mathrm{irp,k}}^{t+1}[{}_{1..V}^{K}\widetilde{O}_c] :=\ & {}^{r,c}f_{\mathrm{irp,k},1}\left({}_{x}^{r}r_{\mathrm{irp,k}}^{t}, {}_{x}^{R}r_{\mathrm{irp,k}}^{t} \right) \triangleq \\ \triangleq\ & \mathcal{HV}(\mathcal{HP}(\mathcal{CG}(\mathcal{FM}(\mathcal{IC}(\mathcal{FE}(I_c^{RGB}), I_c^{D}, P), \\ & {}_{1..M}C^{SIFT}), {}_{1..M}C^{RGB}, {}_{1..M}C^{SIFT}), \\ & {}_{1..M}C^{RGB}, {}_{1..M}C^{SIFT})). \end{aligned} \tag{51} $$

4.8. Control subsystem. The control subsystem c_{irp} is responsible for the realization of the object picking task. In order to fulfil that it must receive the list of the recognized objects from the virtual receptor and aggregate them into a coherent model of the scene, as well as govern the motions of both virtual effectors.

Internal structure of the control subsystem. Fig. 20 presents the inner structure of the control subsystem c_{irp}. The structure of the input and output buffers of the control subsystem c_{irp} has to match that of respective output and input buffers of virtual effectors and receptors, thus: ${}_{x}^{r}c_{\mathrm{irp,k}} = {}_{y}^{c}r_{\mathrm{irp,k}}$, ${}_{x}^{e}c_{\mathrm{irp,g}} = {}_{y}^{c}e_{\mathrm{irp,g}}$, ${}_{x}^{e}c_{\mathrm{irp,m}} = {}_{y}^{c}e_{\mathrm{irp,m}}$, ${}_{y}^{e}c_{\mathrm{irp,g}} = {}_{x}^{c}e_{\mathrm{irp,g}}$, ${}_{y}^{e}c_{\mathrm{irp,m}} = {}_{x}^{c}e_{\mathrm{irp,m}}$.

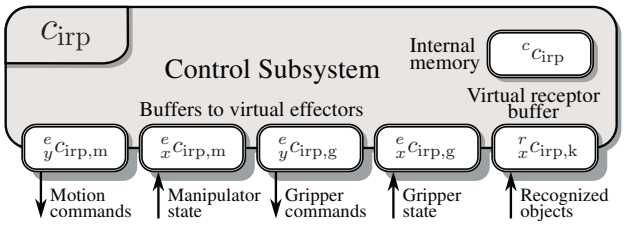

Fig. 20. Inner structure of the control subsystem c_{irp}

The control subsystem stores in its memory several variables, that can be grouped according to their role. First, it stores several variables related to the system calibration:

- ${}^{c}c_{\mathbf{irp}}$ $[{}_{K}^{B}\widetilde{T}]$ – pose of the Kinect sensor with respect to the robot base (world) reference frame,
- ${}^{c}c_{\mathbf{irp}}$ $[{}_{G}^{P}\widetilde{T}]$ – (constant) transformation between the grasp and pre-grasp poses.

The memory also has to store the perceived model of the scene:

- ${}^{c}c_{\mathbf{irp}}$ $[{}_{1..V}^{B}\widetilde{O}_c]$ – a list of verified object hypotheses w.r.t. the robot base.

Next, the memory has to store several desired end-effector and gripper poses, calculated during the task execution:

- ${}^{c}c_{\mathbf{irp}}$ $[{}_{E_g}^{B}\widetilde{T}_d]$ – grasp pose, i.e. desired pose of the manipulator w.r.t. robot base reference frame (calculated for the given object, analogically as the pre-grasp pose), in which the contact with the object is expected,
- ${}^{c}c_{\mathbf{irp}}$ $[{}_{E_p}^{B}\widetilde{T}_d]$ – pre-grasp pose, i.e. desired pose of the end-effector, indicating where from the final grasp approach phase will begin,
- ${}^{c}c_{\mathbf{irp}}$ $[\widetilde{d}_o]$ – desired distance between the two fingers of the gripper, calculated on the basis of object dimensions.

Finally, the control subsystem memory stores several values, used in different phases of the realization of the task:

- ${}^{c}c_{\mathbf{irp}}$ $[{}_{E_s}^{B}\widetilde{T}_d]$ – initial (start) pose of the manipulator w.r.t. the robot base,
- ${}^{c}c_{\mathbf{irp}}$ $[\widetilde{d}_m]$ – maximum distance between the two fingers of the gripper (when the gripper is fully-open),
- ${}^{c}c_{\mathbf{irp}}$ $[{}_{F}^{E}\widetilde{T}]$ – (constant) transformation representing the pose of the center of the gripper (F) w.r.t. end-effector (E),
- ${}^{c}c_{\mathbf{irp}}$ $[{}_{z}\widetilde{V}_d, {}_{z}\widetilde{D}_d, {}_{z}\widetilde{I}_d]$ – values of the velocity, damping and inertia (along the Z axis of the end-effector reference frame), used during the force-monitored approach to the object,

${}^{c}c_{\mathbf{irp}}$ $[{}_{x}\widetilde{D}_d, {}_{y}\widetilde{D}_d, {}_{x}\widetilde{I}_d, {}_{y}\widetilde{I}_d]$ – damping and inertia parameters along the X and Y axes of the end-effector reference frame, used during closing the gripper, thus inducing force-controlled relaxation of tensions between the grasped object and the manipulator,

${}^{c}c_{\mathbf{irp}}$ $[c_l]$ – limit of the measured gripper motor current, indicating that the fingertips have tightened on the object.

Finite state automaton of the control subsystem The task was already defined in the scenario description in Section 4.1. However, it was only a rough description, hiding several details important from the point of view of the control subsystem. For this reason Fig. 21 presents the finite state automaton, being a more detailed version of the automaton shown in Fig. 5.

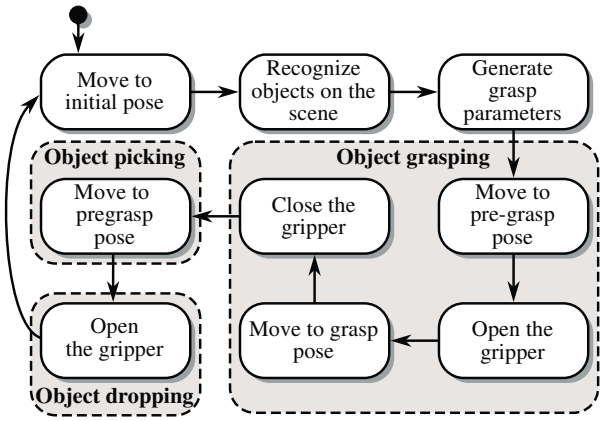

Fig. 21. Detailed graph of the finite state automaton of the control subsystem realizing the object picking task

The resulting scenario is defined as follows. First, it is assumed that the robot should be placed in an initial position – the object picking procedure always starts from this pose. Thus the first state of the FSM, ${}^{c}\mathcal{S}_{\text{irp},1}$, is associated with the first behaviour ${}^{c}\mathcal{B}_{\text{irp},1}$. This behaviour terminates when the end-effector reaches the initial pose:

$$ {}^{c}f^{\tau}_{\text{irp},1} \triangleq \left({}^{B}_{E}T_c = {}^{B}_{E_s}T_d\right). \tag{52}$$

In the next state the system waits until some objects will be recognized in the scene. So the ${}^{c}\mathcal{S}_{\text{irp},2}$ state is associated with the ${}^{c}\mathcal{B}_{\text{irp},2}$ behaviour, which ends when the list ${}^{B}_{1..V}\widetilde{O}_c$ stored in the memory ${}^{c}c_{\text{irp,k}}$ will contain at least one object, i.e.:

$$ {}^{c}f^{\tau}_{\text{irp},2} \triangleq {}^{c}c_{\text{irp}}[{}^{B}_{1..V}\widetilde{O}_c] \neq \text{NIL}. \tag{53}$$

This condition assures that when no objects will be recognized in the scene the control subsystem will remain in the state ${}^{c}\mathcal{S}_{\text{irp},2}$, executing the behaviour ${}^{c}\mathcal{B}_{\text{irp},2}$.

After successful recognition of objects in the scene the control subsystem moves to the state ${}^{c}\mathcal{S}_{\text{irp},3}$, in which it generates adequate grasp parameters for the selected object of interest. Those computations require just one step, thus:

$$ {}^{c}f^{\tau}_{\text{irp},3} \triangleq \text{TRUE}. \tag{54}$$

Object grasping is realized in four consecutive steps, based on the grasp parameters computed earlier. First in the state ${}^{c}\mathcal{S}_{\text{irp},4}$ the manipulator moves to a pre-grasp pose (which is assumed to be 20 cm above the object along the Z axis of the object frame). Thus the behaviour finishes when the pre-grasp pose ${}^{B}_{E_d}T_p$ will be reached:

$$ {}^{c}f^{\tau}_{\text{irp},4} \triangleq \left({}^{B}_{E}T_c = {}^{B}_{E_p}T_d\right). \tag{55}$$

Next, in the state ${}^{c}\mathcal{S}_{\text{irp},5}$, the system opens the robot gripper. It is assumed that the gripper will always open to maximum extent, thus the terminal condition is defined as the result of comparison between the current distance d_c and the maximum distance d_m between the gripper fingers:

$$ {}^{c}f^{\tau}_{\text{irp},5} \triangleq (d_c = d_m). \tag{56}$$

After opening the gripper to the maximum extent the manipulator approachs the object along a trajectory (being a straight line along the Z axis of the end-effector frame), reaching the grasp pose ${}^{B}_{E}T_g$. Additionally, the exerted force must be measured, to stop the motion then a force along the Z axis is detected. Thus the terminal condition of behaviour ${}^{c}\mathcal{B}_{\text{irp},6}$ executed in the state ${}^{c}\mathcal{S}_{\text{irp},6}$ is defined as:

$$ {}^{c}f^{\tau}_{\text{irp},6} \triangleq \left(\left({}^{B}_{E}T_c = {}^{B}_{E}T_g\right) \vee \left({}^{E}_{z}F_c > 0\right)\right). \tag{57}$$

In the state ${}^{c}\mathcal{S}_{\text{irp},7}$ the robot grasps the object by closing the gripper, using the distance between fingers d_o being computed on the basis of the known object dimensions. Moreover, as the object should not be crushed, the forces applied by the fingers on the object should be monitored. However, as the gripped does not possess force/pressure sensors, it was decided to monitor the current in the motor propelling the fingers. Thus the behaviour is terminated when either the prescribed finger distance is reached or the current exceeds a given threshold c_l:

$$ {}^{c}f^{\tau}_{\text{irp},7} \triangleq (d_c = d_o) \vee (c_c \geq c_l). \tag{58}$$

This ends the object grasping phase utilizing position-force control.

In the next state, ${}^{c}\mathcal{S}_{\text{irp},8}$, the associated behaviour is responsible for object lifting. It was decided to use for that purpose the previous pre-grasp pose as the desired pose of the manipulator end-effector, thus the behaviour ${}^{c}\mathcal{B}_{\text{irp},4}$ and its terminal condition ${}^{c}f^{\tau}_{\text{irp},4}$, initially associated with the state ${}^{c}\mathcal{S}_{\text{irp},4}$, can be reused. This also shows that the relationship between states and behaviours does not have to be one–to–one.

In the state ${}^{c}\mathcal{S}_{\text{irp},9}$ the control subsystem opens the gripper, what will result in dropping the object in a random place and with a random orientation. Analogically, in this state we can use the terminal condition and behaviour used in the state ${}^{c}\mathcal{S}_{\text{irp},5}$. After that the procedure is repeated from the beginning.

Additionally, as each two nodes of the automaton graph are connected by single arrows, thus always a single transition between states of the automaton is the only option, rather a trivial

case, all initial conditions are defined as:

$$ {}^{c}f^{\sigma}_{\text{irp},l} \triangleq \text{TRUE}, l = 1,\ldots,9. \tag{59}$$

In this simple scenario no error situations are assumed, thus:

$$ {}^{c}f^{\varepsilon}_{\text{irp},l} \triangleq \text{FALSE}, l = 1,\ldots,9. \tag{60}$$

The above described states, behaviours and conditions form the FSM presented in Fig. 22. Table 9 presents the association between the states of the automaton and behaviours, whereas the state transition table, being equivalent to the above described conditions, is presented in Table 10.

Table 9
Mapping of the control subsystem states to its behaviours

State	Beh.	Description
${}^{c}\mathcal{S}_{\text{irp},1}$	${}^{c}\mathcal{B}_{\text{irp},1}$	Move to initial pose
${}^{c}\mathcal{S}_{\text{irp},2}$	${}^{c}\mathcal{B}_{\text{irp},2}$	Recognize objects in the scene
${}^{c}\mathcal{S}_{\text{irp},3}$	${}^{c}\mathcal{B}_{\text{irp},3}$	Generate grasp parameters
${}^{c}\mathcal{S}_{\text{irp},4}$	${}^{c}\mathcal{B}_{\text{irp},4}$	Move to pregrasp pose
${}^{c}\mathcal{S}_{\text{irp},5}$	${}^{c}\mathcal{B}_{\text{irp},5}$	Open the gripper
${}^{c}\mathcal{S}_{\text{irp},6}$	${}^{c}\mathcal{B}_{\text{irp},6}$	Move to grasp pose (force cont.)
${}^{c}\mathcal{S}_{\text{irp},7}$	${}^{c}\mathcal{B}_{\text{irp},7}$	Close the gripper (force cont.)
${}^{c}\mathcal{S}_{\text{irp},8}$	${}^{c}\mathcal{B}_{\text{irp},4}$	Move to pregrasp pose
${}^{c}\mathcal{S}_{\text{irp},9}$	${}^{c}\mathcal{B}_{\text{irp},5}$	Open the gripper

Table 10
State transition table of the control subsystem FSM

Current state		Next state	
State	Terminal/Error Condition	Initial Condition	State
${}^{c}\mathcal{S}_{\text{irp},1}$	${}^{c}f^{\tau}_{\text{irp},1}$ = TRUE	${}^{c}f^{\sigma}_{\text{irp},2}$ = TRUE	${}^{c}\mathcal{S}_{\text{irp},2}$
${}^{c}\mathcal{S}_{\text{irp},2}$	${}^{c}f^{\tau}_{\text{irp},2}$ = TRUE	${}^{c}f^{\sigma}_{\text{irp},3}$ = TRUE	${}^{c}\mathcal{S}_{\text{irp},3}$
${}^{c}\mathcal{S}_{\text{irp},3}$	${}^{c}f^{\tau}_{\text{irp},3}$ = TRUE	${}^{c}f^{\sigma}_{\text{irp},4}$ = TRUE	${}^{c}\mathcal{S}_{\text{irp},4}$
${}^{c}\mathcal{S}_{\text{irp},4}$	${}^{c}f^{\tau}_{\text{irp},4}$ = TRUE	${}^{c}f^{\sigma}_{\text{irp},5}$ = TRUE	${}^{c}\mathcal{S}_{\text{irp},5}$
${}^{c}\mathcal{S}_{\text{irp},5}$	${}^{c}f^{\tau}_{\text{irp},5}$ = TRUE	${}^{c}f^{\sigma}_{\text{irp},6}$ = TRUE	${}^{c}\mathcal{S}_{\text{irp},6}$
${}^{c}\mathcal{S}_{\text{irp},6}$	${}^{c}f^{\tau}_{\text{irp},6}$ = TRUE	${}^{c}f^{\sigma}_{\text{irp},7}$ = TRUE	${}^{c}\mathcal{S}_{\text{irp},7}$
${}^{c}\mathcal{S}_{\text{irp},7}$	${}^{c}f^{\tau}_{\text{irp},7}$ = TRUE	${}^{c}f^{\sigma}_{\text{irp},8}$ = TRUE	${}^{c}\mathcal{S}_{\text{irp},8}$
${}^{c}\mathcal{S}_{\text{irp},8}$	${}^{c}f^{\tau}_{\text{irp},4}$ = TRUE	${}^{c}f^{\sigma}_{\text{irp},9}$ = TRUE	${}^{c}\mathcal{S}_{\text{irp},9}$
${}^{c}\mathcal{S}_{\text{irp},9}$	${}^{c}f^{\tau}_{\text{irp},5}$ = TRUE	${}^{c}f^{\sigma}_{\text{irp},1}$ = TRUE	${}^{c}\mathcal{S}_{\text{irp},1}$

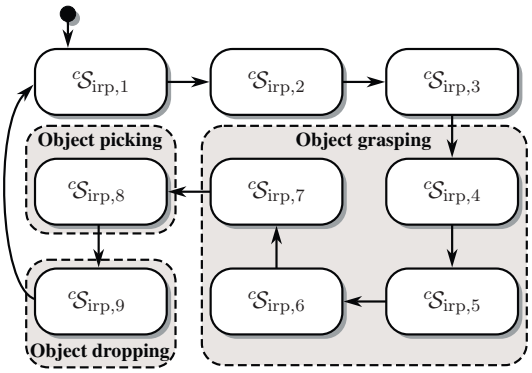

Fig. 22. Detailed graph of the finite state automaton of the control subsystem enumerating its states

Behaviour ${}^{c}\mathcal{B}_{\text{irp,1}}$. Behaviour ${}^{c}\mathcal{B}_{\text{irp},1}$ is responsible for moving the manipulator to the initial position. As it is the only goal thus the only required function is the effector control function ${}^{c,e}f_{irp,1}$, thus it is defined as:

$$ {}^{c}\mathcal{B}_{\text{irp},1} \triangleq {}^{c}\mathcal{B}_{\text{irp},1}\left({}^{c,e}f_{\text{irp},1}, {}^{c}f^{\tau}_{\text{irp},1}, {}^{c}f^{\varepsilon}_{\text{irp},1}\right). \tag{61}$$

The effector control function ${}^{c,e}f_{irp,1}$ sends the desired end-effector pose to the virtual effector through the ${}^{e}_{y}c_{\text{irp,m}}$ buffer:

$$ {}^{e}_{y}c^{i+1}_{\text{irp,m}}[{}^{B}_{E}\widetilde{T}_{d}] := {}^{c,e}f_{\text{irp},1}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq {}^{B}_{E_s}T_{d}. \tag{62}$$

Behaviour ${}^{c}\mathcal{B}_{\text{irp,2}}$. Behaviour ${}^{c}\mathcal{B}_{\text{irp},2}$ is responsible for recognition of the state of the scene. As it was decided that visual perception will be a passive process, the behaviour simply adequately transforms the data received from the virtual receptor and memorises the result. It is defined as follows:

$$ {}^{c}\mathcal{B}_{\text{irp},2} \triangleq {}^{c}\mathcal{B}_{\text{irp},2}\left({}^{c,c}f_{\text{irp},2}, {}^{c}f^{\tau}_{\text{irp},2}, {}^{c}f^{\varepsilon}_{\text{irp},2}\right). \tag{63}$$

Fig. 23 presents the data flow diagram of the memory function ${}^{c,c}f_{\text{irp},2}$. Initially ${}^{c,c}f_{\text{irp},2}$ acquires from the input buffer ${}^{r}_{x}c_{irp,k}$ the list of locations of objects (recognized in the given RGB-D image) w.r.t. Kinect frame and transforms them into the world (robot base) coordinate frame: ${}^{B}_{1..V}O_{c} = {}^{B}_{K}T\,{}^{K}_{1..V}O_{c}$. The operation includes transformation of coordinates of all points of: the dense colour point cloud ${}_{v}C^{\text{RGB}}_{c}$, sparse feature cloud ${}_{v}C^{\text{SIFT}}_{c}$, as well as those of the object pose ${}^{K}_{v}T_{c}$.

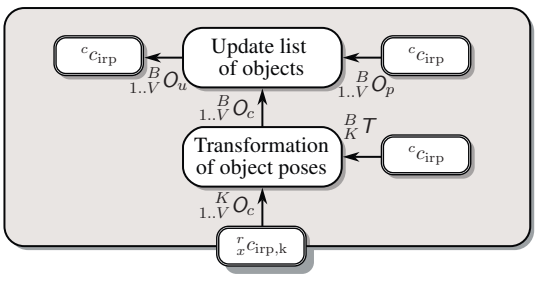

Fig. 23. Data flow diagram of the function ${}^{c,c}f_{\text{irp},2}$ storing in the memory the recognized objects having poses transformed into the robot base frame

The resulting list is next used to update the stored model of the scene by merging it with the stored list of objects recognized in the past ${}_{1..V}^{B}O_p$: ${}_{1..V}^{B}O_u = \mathcal{US}\left({}_{1..V}^{B}O_p, {}_{1..V}^{B}O_c\right)$ ($\mathcal{US}$ stands for Update Scene). The procedure takes into account the id of the object model (extracted from object id ${}_{v}i_c$) and its pose in the scene – if those two match, the currently recognized object ${}_{v}^{B}O_c$ is merged with the respective object from the control subsystem memory ${}_{v}^{B}O_p$, i.e. the control subsystem stores the new object pose ${}_{v}^{B}T_u = {}_{v}^{B}T_c$ and updates the confidence ratio ${}_{v}c_u = {}_{v}c_p/2 + {}_{v}c_c$ (which promotes the objects that are visible currently and were present is several previous frames). Finally, the list is stored once again in the memory buffer, so the memory function is defined as:

$$\begin{aligned} {}^{c}c_{\text{irp}}^{i+1}[{}_{1..V}^{B}\widetilde{O}_c] &:= {}^{c,c}f_{\text{irp},2}\left({}^{c}c_{\text{irp}}^{i}, {}_{x}^{r}c_{\text{irp,k}}^{i}\right) \triangleq \\ &\triangleq \mathcal{US}\left({}_{1..V}^{B}O_p, {}_{K}^{B}T\,{}_{1..V}^{K}O_c\right). \end{aligned} \tag{64}$$

Behaviour ${}^{c}\mathcal{B}_{\text{irp,3}}$. This behaviour is responsible for estimation of the grasp parameters. There are many sophisticated approaches to grasp planning (e.g. [39, 6, 64, 65]), as well as several tools facilitating that process (e.g. GraspIt! [55]). However, as this example has to be kept simple, we decided to use an elementary approach consisting of four consecutive steps, analogical to the one that had been used by two-handed manipulation of a Rubik's cube [90, 92]. It assumes that a proper grasp is realised by approaching the object from above it along a straight line (simplification of trajectory planning) and that the reached pose enables straightforward grasping by simple reconfiguration of the posture of the gripper. The four consecutive steps are: reach the pre-grasp pose, open the gripper, reach the grasp pose, close the gripper. So the goal of the ${}^{c}\mathcal{B}_{\text{irp,3}}$ behaviour is to calculate the three following parameters:

- the pre-grasp pose ${}_{E_p}^{B}T_d$ – pose of the end-effector from which the approach phase will begin,
- the grasp pose ${}_{E_g}^{B}T_d$ – pose of the manipulator in which the contact with the object is expected,
- d_o – desired distance between the two fingers of the gripper enabling them to grasp the object.

Additionally, we assumed that the system holds in its memory the maximum distance between the fingers of the gripper d_m, which will be used for setting the "gripper fully-open" posture.

The behaviour ${}^{c}\mathcal{B}_{\text{irp,3}}$ is thus purely computational, depending only on internal memory of the control subsystem:

$${}^{c}\mathcal{B}_{\text{irp,3}} \triangleq {}^{c}\mathcal{B}_{\text{irp,3}}\left({}^{c,c}f_{\text{irp,3}}, {}^{c}f_{\text{irp,3}}^{\tau}, {}^{c}f_{\text{irp,3}}^{\varepsilon}\right), \tag{65}$$

with ${}^{c,c}f_{\text{irp,3}}$ presented in Fig. 24. As there are three major data flows, resulting in computation of different variables stored in ${}^{c}c_{\text{irp}}$ memory, thus:

$${}^{c}c_{\text{irp}}^{i+1} := {}^{c,c}f_{\text{irp,3}}\left({}^{c}c_{\text{irp}}^{i}\right) \triangleq \begin{cases} {}^{c,c}f_{\text{irp,3,1}}\left({}^{c}c_{\text{irp}}^{i}\right), \\ {}^{c,c}f_{\text{irp,3,2}}\left({}^{c}c_{\text{irp}}^{i}\right), \\ {}^{c,c}f_{\text{irp,3,3}}\left({}^{c}c_{\text{irp}}^{i}\right). \end{cases} \tag{66}$$

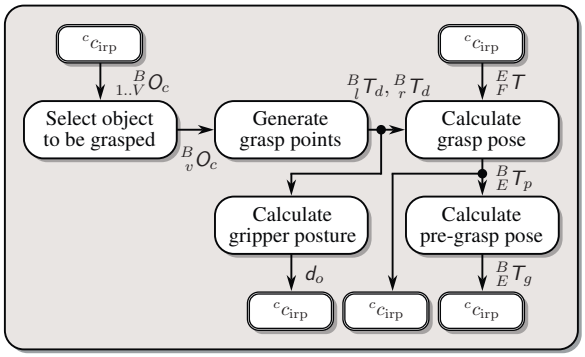

Fig. 24. Data flow of the function ${}^{c,c}f_{\text{irp,3}}$ responsible for generation of the grasp parameters

However, they share some computations. First, the system must decide which object ${}_{v}^{B}O_c$ from the list of objects ${}_{1..V}^{B}O_c$ detected in the scene will be picked. For this purpose the system analyses their confidence ratio c and selects the object ($\mathcal{SO}$) with the highest one: ${}_{v}^{B}O_c = \underset{c_c}{\text{argmax}}\left({}_{1..V}^{B}O_c\right)$. The model of the selected object is used for generation of position of two grasp points ${}_{l}^{B}T_d, {}_{r}^{B}T_d$, being the desired poses of the left and right gripper fingertips respectively (thus the function was named $\mathcal{LR}$). On that basis the first partial function ${}^{c,c}f_{\text{irp,3,1}}$ calculates the desired distance between gripper fingers d_o, which is simply equal to the Euclidean distance between points: $d_o = \left\|{}_{l}^{B}T, {}_{r}^{B}T\right\|$. So the function is defined as:

$${}^{c}c_{\text{irp}}^{i+1}[\widetilde{d}_o] := {}^{c,c}f_{\text{irp,3,1}}\left({}^{c}c_{\text{irp}}^{i}\right) \triangleq \left\|\mathcal{LR}\left(\underset{c_c}{\text{argmax}}\left({}_{1..V}^{B}O_c\right)\right)\right\|. \tag{67}$$

The second partial function is responsible for computation of the grasp pose ${}_{E_g}^{B}T_d$, i.e. the desired pose of the end-effector enabling grasping by a simply closing the gripper. For this reason it calculates the pose in the center of the line connecting the left and right finger poses ${}_{l}^{B}T_d, {}_{r}^{B}T_d$, and uses it for computation of the end-effector pose by multiplying it by the known (constant) transformation between the end-effector and the center of the gripper ${}_{F}^{E}T$, thus:

$$\begin{aligned} {}^{c}c_{\text{irp}}^{i+1}[{}_{E_g}^{B}\widetilde{T}_d] &:= {}^{c,c}f_{\text{irp,3,2}}\left({}^{c}c_{\text{irp}}^{i}\right) \triangleq \\ &\triangleq \mathcal{GC}\left(\mathcal{LR}\left(\underset{c_c}{\text{argmax}}\left({}_{1..V}^{B}O_c\right)\right), {}_{F}^{E}T\right), \end{aligned} \tag{68}$$

where $\mathcal{GC}$ stands for Grasp-pose Calculation.

Finally, the grasp pose enables the computation of the pre-grasp pose by using the transformation ${}_{G}^{P}T$, being a simple translation (by 20 cm) along the Z axis of the end-effector:

$$\begin{aligned} {}^{c}c_{\text{irp}}^{i+1}[{}_{E_p}^{B}\widetilde{T}_d] &:= {}^{c,c}f_{\text{irp,3,3}}\left({}^{c}c_{\text{irp}}^{i}\right) \triangleq \\ &\triangleq \mathcal{GC}\left(\mathcal{LR}\left(\underset{c_c}{\text{argmax}}\left({}_{1..V}^{B}O_c\right)\right), {}_{F}^{E}T\right) {}_{G}^{P}T^{-1}. \end{aligned} \tag{69}$$

Behaviour ${}^{c}\mathcal{B}_{\text{irp,4}}$. The behaviour controls the motion of the end-effector, analogically to the behaviour ${}^{c}\mathcal{B}_{\text{irp,1}}$. The only dif-

ference is that it uses the pre-grasp pose ${}^{B}_{E_p}T_d$ as a goal. The definition of the behaviour is as follows:

$$ {}^{c}\mathcal{B}_{\text{irp},4} \triangleq {}^{c}\mathcal{B}_{\text{irp},4}\left({}^{c,e}f_{\text{irp},4}, {}^{c}f^{\tau}_{\text{irp},4}, {}^{c}f^{\varepsilon}_{\text{irp},4}\right), \tag{70} $$

whereas the transition function controlling the end-effector is:

$$ {}^{e}_{y}c^{i+1}_{\text{irp,m}}[{}^{B}_{E}\widetilde{T}_d] := {}^{c,e}f_{\text{irp},4}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq {}^{B}_{E_p}T_d. \tag{71} $$

Behaviour ${}^{c}\mathcal{B}_{\text{irp,5}}$. The goal of the behaviour ${}^{c}\mathcal{B}_{\text{irp},5}$ is to prepare the robot for grasping by opening the gripper. As its only role is to control the gripper, it requires the definition of a single end-effector control function:

$$ {}^{c}\mathcal{B}_{\text{irp},5} \triangleq {}^{c}\mathcal{B}_{\text{irp},5}\left({}^{c,e}f_{\text{irp},5}, {}^{c}f^{\tau}_{\text{irp},5}, {}^{c}f^{\varepsilon}_{\text{irp},5}\right). \tag{72} $$

The change of the posture is realized by sending a command with the desired (maximal) distance between the fingers d_m to the output buffer ${}^{e}_{y}c_{\text{irp,g}}$ to the gripper virtual effector:

$$ {}^{e}_{y}c^{i+1}_{\text{irp,g}}[\widetilde{d}_d] := {}^{c,e}f_{\text{irp},5}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq d_m, \tag{73} $$

and by monitoring the current distance received through the input butter ${}^{e}_{x}c_{\text{irp,g}}$ as defined in the terminal condition (56).

Behaviour ${}^{c}\mathcal{B}_{\text{irp,6}}$Behaviour ${}^{c}\mathcal{B}_{\text{irp},6}$ moves the manipulator down along the Z axis of the end-effector until contact with the object is detected. For this purpose it utilizes the position-force control behaviour of the virtual effector, presented in Section 4.5. The position-force control realizes a relative motion, i.e. all motion parameters are expressed w.r.t. the end-effector reference frame. The only role of the control subsystem is to set the appropriate operation modes and parameters of the decoupled position-force control regulators for each motion component and send them to the virtual effector, thus:

$$ {}^{c}\mathcal{B}_{\text{irp},6} \triangleq {}^{c}\mathcal{B}_{\text{irp},6}\left({}^{c,e}f_{\text{irp},6}, {}^{c}f^{\tau}_{\text{irp},6}, {}^{c}f^{\varepsilon}_{\text{irp},6}\right). \tag{74} $$

Operation of the position-force regulators is defined by 5 parameters. This results in the decomposition of the effector control function into five partial functions, responsible for setting different parameters of the motion:

$$ {}^{e}_{y}c^{i+1}_{\text{irp,m}} := {}^{c,c}f_{\text{irp},6}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq \begin{cases} {}^{c,c}f_{\text{irp},6,1}\left({}^{c}c^{i}_{\text{irp}}\right), \\ {}^{c,c}f_{\text{irp},6,2}\left({}^{c}c^{i}_{\text{irp}}\right), \\ {}^{c,c}f_{\text{irp},6,3}\left({}^{c}c^{i}_{\text{irp}}\right), \\ {}^{c,c}f_{\text{irp},6,4}\left({}^{c}c^{i}_{\text{irp}}\right), \\ {}^{c,c}f_{\text{irp},6,5}\left({}^{c}c^{i}_{\text{irp}}\right). \end{cases} \tag{75} $$

As the motion is realized along the Z axis of the end-effector, the parameters of the regulators of other 5 motion components will be zeros or uninterpreted, as presented in Table 11.

Table 11
Modes and parameters of the position-force control set by the ${}^{c}\mathcal{B}_{\text{irp},6}$ behaviour. „–” indicates that given parameter is not necessary in the given control mode

Parameter	Motion component					
	x	y	z	a_x	a_y	a_z
b	$\mathfrak{u}$	$\mathfrak{u}$	$\mathfrak{g}$	$\mathfrak{u}$	$\mathfrak{u}$	$\mathfrak{u}$
F_d	–	–	–	–	–	–
V_d	0	0	${}^{c}c_{\text{irp}}[{}_{z}\widetilde{V}_d]$	0	0	0
D_d	–	–	${}^{c}c_{\text{irp}}[{}_{z}\widetilde{D}_d]$	–	–	–
I_d	–	–	${}^{c}c_{\text{irp}}[{}_{z}\widetilde{I}_d]$	–	–	–

The first partial function sets the mode of the regulators, i.e. guarded motion $\mathfrak{g}$ along the Z axis and unguarded motion $\mathfrak{u}$ for the other motion components:

$$ {}^{e}_{y}c^{i+1}_{\text{irp,m}}[\widetilde{b}] := {}^{c,e}f_{\text{irp},6,1}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq [\mathfrak{u},\mathfrak{u},\mathfrak{g},\mathfrak{u},\mathfrak{u},\mathfrak{u}]. \tag{76} $$

The second parameter is the vector of the desired values of the exerted forces, necessary only in the case of the contact mode $\mathfrak{c}$, thus:

$$ {}^{e}_{y}c^{i+1}_{\text{irp,m}}[\widetilde{F}_d] := {}^{c,e}f_{\text{irp},6,2}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq [-,-,-,-,-,-]. \tag{77} $$

As the end-effector has to move only along the Z axis, the linear and angular velocities for the other five components of motion must be set to zero:

$$ {}^{e}_{y}c^{i+1}_{\text{irp,m}}[\widetilde{V}_d] := {}^{c,e}f_{\text{irp},6,3}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq [0,0,{}_{z}V_d,0,0,0], \tag{78} $$

where ${}_{z}V_d$ is the desired, constant velocity along the Z axis (1 cm/s). Additionally, in the case of guarded motion $\mathfrak{g}$ it is required to set the additional motion parameters, i.e. damping:

$$ {}^{e}_{y}c^{i+1}_{\text{irp,m}}[\widetilde{D}_d] := {}^{c,e}f_{\text{irp},6,4}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq [-,-,{}_{z}D_d,-,-,-], \tag{79} $$

and inertia:

$$ {}^{e}_{y}c^{i+1}_{\text{irp,m}}[\widetilde{I}_d] := {}^{c,e}f_{\text{irp},6,5}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq [-,-,{}_{z}I_d,-,-,-]. \tag{80} $$

Behaviour ${}^{c}\mathcal{B}_{\text{irp,7}}$. The goal of the ${}^{c}\mathcal{B}_{\text{irp},7}$ behaviour is to finally realize the object grasp by closing the gripper fingers:

$$ {}^{c}\mathcal{B}_{\text{irp},7} \triangleq {}^{c}\mathcal{B}_{\text{irp},7}\left({}^{c,e}f_{\text{irp},7}, {}^{c}f^{\tau}_{\text{irp},7}, {}^{c}f^{\varepsilon}_{\text{irp},7}\right). \tag{81} $$

Theoretically this behaviour could rely only on the virtual effector controlling the gripper. However, there are always some errors in pose estimation, resulting in stress when contact between the fingers and the object occurs, what may result in slight changes of the object pose. In order to reduce the stress we decided to control the manipulator using position-force control. The idea is to control the manipulator in such a way that

it will become compliant to the forces exerted in the XY plane, trying to reduce them to zero. Thus the effector control function was decomposed into six partial functions:

$$\left[{}^{e}_{y}c^{i+1}_{\text{irp,m}}, {}^{e}_{y}c^{i+1}_{\text{irp,g}}\right] := {}^{c,e}f_{\text{irp},7}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq \begin{cases} {}^{c,e}f_{\text{irp},7,1}\left({}^{c}c^{i}_{\text{irp}}\right), \\ {}^{c,e}f_{\text{irp},7,2}\left({}^{c}c^{i}_{\text{irp}}\right), \\ {}^{c,e}f_{\text{irp},7,3}\left({}^{c}c^{i}_{\text{irp}}\right), \\ {}^{c,e}f_{\text{irp},7,4}\left({}^{c}c^{i}_{\text{irp}}\right), \\ {}^{c,e}f_{\text{irp},7,5}\left({}^{c}c^{i}_{\text{irp}}\right), \\ {}^{c,e}f_{\text{irp},7,6}\left({}^{c}c^{i}_{\text{irp}}\right), \end{cases} \tag{82}$$

where the first five are responsible for setting the modes and parameters of the position-force control of the manipulator virtual effector, and the last one sets the desired distance between the gripper fingers. Table 12 presents the position-force control parameters set by the first five functions.

Table 12
Modes and parameters of the position-force control set by the ${}^{c}\mathcal{B}_{\text{irp},7}$ behaviour. „–" indicates that the given parameter is not necessary in the given control mode

	Motion component					
	x	y	z	a_x	a_y	a_z
b	$\mathfrak{c}$	$\mathfrak{c}$	$\mathfrak{u}$	$\mathfrak{u}$	$\mathfrak{u}$	$\mathfrak{u}$
F_d	0	0	–	–	–	–
V_d	–	–	0	0	0	0
D_d	${}^{c}c_{\text{irp}}[{}_{x}\widetilde{D}_d]$	${}^{c}c_{\text{irp}}[{}_{y}\widetilde{D}_d]$	–	–	–	–
I_d	${}^{c}c_{\text{irp}}[{}_{x}\widetilde{I}_d]$	${}^{c}c_{\text{irp}}[{}_{y}\widetilde{I}_d]$	–	–	–	–

The first partial function sets the operation modes:

$${}^{e}_{y}c^{i+1}_{\text{irp,m}}[\widetilde{b}] := {}^{c,e}f_{\text{irp},7,1}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq [\mathfrak{c},\mathfrak{c},\mathfrak{u},\mathfrak{u},\mathfrak{u},\mathfrak{u}]. \tag{83}$$

The second partial function causes the virtual effector to reduce to zero the forces along the X and Y axes (of the gripper frame as this is a relative motion):

$${}^{e}_{y}c^{i+1}_{\text{irp,m}}[\widetilde{F}_d] := {}^{c,e}f_{\text{irp},7,2}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq [0,0,-,-,-,-]. \tag{84}$$

The other components of the end-effector pose should not change, thus:

$${}^{e}_{y}c^{i+1}_{\text{irp,m}}[\widetilde{V}_d] := {}^{c,e}f_{\text{irp},7,3}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq [-,-,0,0,0,0]. \tag{85}$$

The two motion components that that are set to contact mode $\mathfrak{c}$ the values of inertia and damping must be set:

$${}^{e}_{y}c^{i+1}_{\text{irp,m}}[\widetilde{D}_d] := {}^{c,e}f_{\text{irp},7,4}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq [{}_{x}D_d, {}_{y}D_d,-,-,-,-], \tag{86}$$

$${}^{e}_{y}c^{i+1}_{\text{irp,m}}[\widetilde{I}_d] := {}^{c,e}f_{\text{irp},7,5}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq [{}_{x}I_d, {}_{y}I_d,-,-,-,-]. \tag{87}$$

The partial end-effector control function responsible for control of the gripper pose writes the desired distance between the fingers d_o into the buffer associated with that virtual effector:

$${}^{e}_{y}c^{i+1}_{\text{irp,g}}[\widetilde{d}_d] := {}^{c,e}f_{\text{irp},7,6}\left({}^{c}c^{i}_{\text{irp}}\right) \triangleq d_o. \tag{88}$$

The last three partial functions use several parameters that are stored in the control subsystem memory. The values of the majority of those parameters were determined experimentally, e.g. after several trials of closing the gripper over several objects the current in the gripper motor, that signals adequate grasp, was estimated as $c_I = 100$ mA. Parameters such as damping and inertia required by the position-force control were estimated by a more systematic procedure, i.e. performed several hundred experiments repeating the grasping procedure for different values of those parameters. Generally, the value of damping between 200 $\frac{\text{kg}}{\text{s}}$ and 400 $\frac{\text{kg}}{\text{s}}$ resulted in successful grasps, whereas the value of inertia had no influence on the process, mainly due to the low velocity and acceleration of the motion. For more detailed description of the experimental identification of position-force control parameters refer to [41].

4.9. Experimental verification. Experimental verification of the system performance was done using the IRp-6 6-DOF manipulator. The manipulator is controlled by in-house designed hardware (having a 500Hz position control loop) [77]. The manipulator end-effector is equipped with a two-finger gripper supplemented by a Point Grey Blackfly BFLY-PGE-14S2C-CS camera, attached to the robot wrist (however the camera was not used in the experiments). Additionally, there is a Kinect sensor mounted above the manipulator workspace. The system calibration procedure is presented in [78].

The presented control system was implemented using the Robot Operating System (ROS) [23, 45]. The virtual effectors constituting the low-level robot controllers were implemented as OROCOS (Open Robot Control Software [16]) components. Detailed description of the components can be found in [79]. Here it is only indicated that along with the library of components for control of the IRp-6 robot a high-level API was developed. It facilitates programming of diverse tasks. The library and the API together form the IRPOS controller. The virtual receptor was implemented using DisCODe (Distributed Component Oriented Data Processing [71]) – a component-based framework that can also work as a ROS node. DisCODe components rely on two libraries: OpenCV [12, 46] for 2D vision (e.g. extraction of SIFT features) and PCL (Point Cloud Library) [61] for 3D perception (processing of point clouds, filtering, matching etc.). The mapping of subsystems to ROS/OROCOS/DisCODe is presented in Fig. 26.

We validated the system making it pick specific objects out of several different, randomly placed ones. Fig. 25 presents images acquired during one of such experiments. In the middle row RViz visualization of the first three steps is presented, with

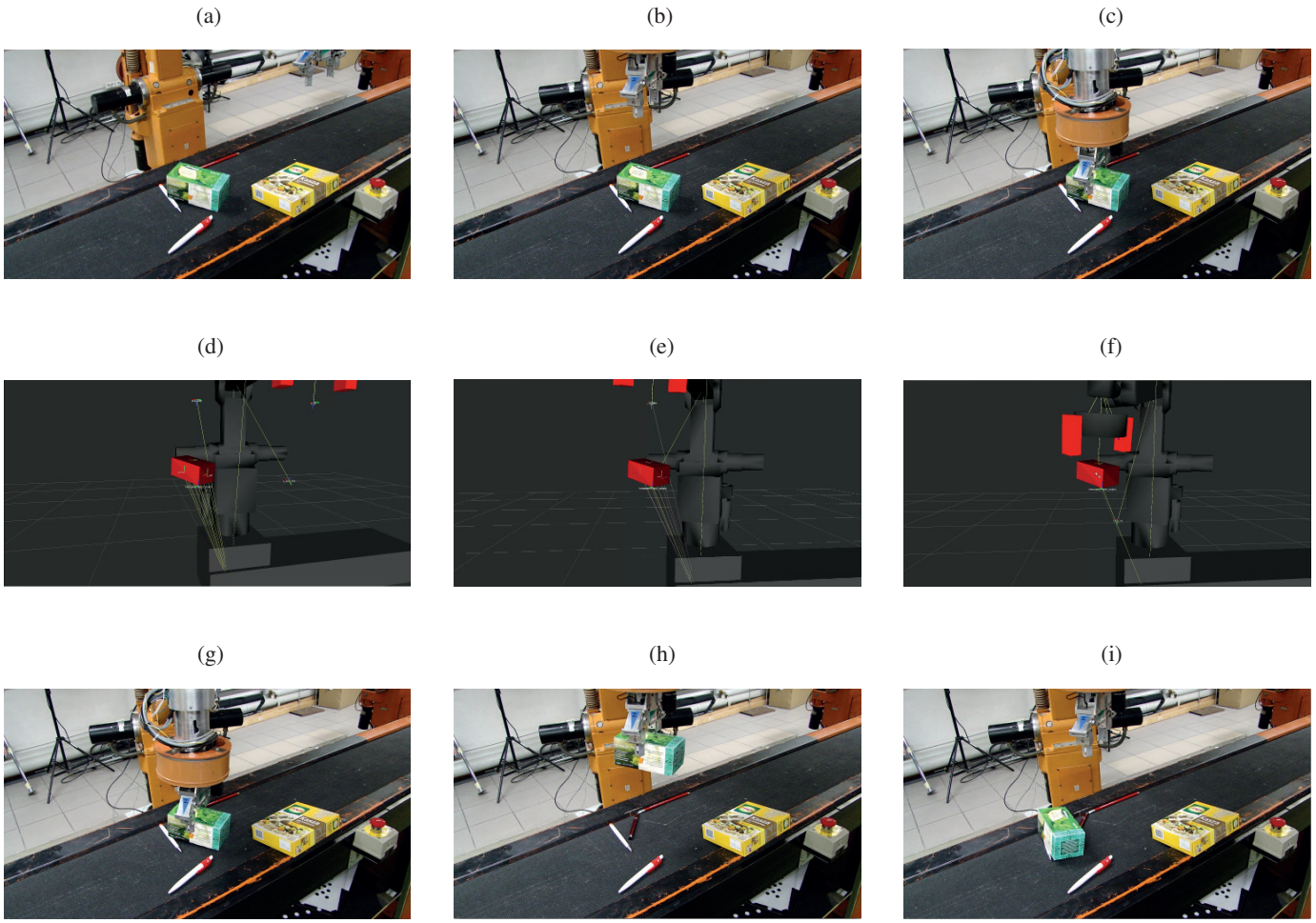

Fig. 25. Images from one of the experiments. Top row: (a) initial pose and recognition of objects on the scene, (b) pre-grasp pose reached, (c) grasp pose reached. Middle row: visualization of (a–c) in RViz. Bottom row: (g) griper closed, (e) object picked, (f) gripper opened and thus the object dropped

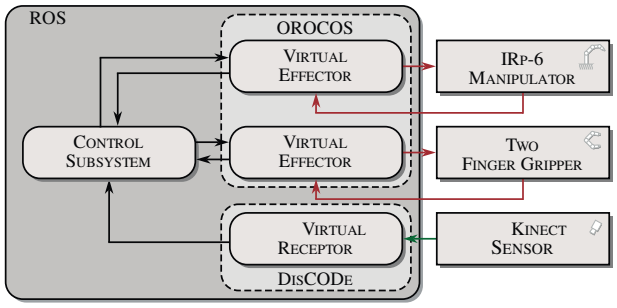

Fig. 26. Mapping of agent subsystems onto control software modules implemented in ROS, OROCOS and DisCODe

all the relevant reference frames indicated, such as: gripper reference frame, pre-grasp, grasp, grasping pose etc. The film picturing this experiment, along with the visualization of consecutive phases of object recognition, generation of grasp parameters etc. can be seen at[1].

[1]https://vimeo.com/139905134

5. Related work and discussion

The survey [43] of robot control system architectures pointed out that the majority of designed systems uses informal methods of specification usually relying on boxes and arrows approach. This observation did not change in the recently extended and updated version [44] of that paper. Although some work has been done on formal specification of robot control software (e.g. [52, 51, 83, 87, 5]) and its formal verification (e.g. [53, 8]), unfortunately it has not gained widespread adoption. One of the few systems that employs formal specification and generation of code for the implementation of the lowest layer of robotic system controller is G^{en}oM (Generator of Modules) [26, 1]. It is integrated with BIP (Behaviour Interaction Priorities) [9] framework and toolset, which is used to formally specify and verify the correctness of the produced system, treated as real-time component based complex structure.

As this paper not only fosters a formalised approach to robotic system specification, but also promotes a specific architecture, a fundamental question is whether the proposed architecture is flexible enough to accommodate other architectures

that have been proposed by other researchers, i.e. can the functionality of other architectures be reproduced within the proposed architecture? To show that this is the case, some popular architectures will be outlined using the proposed approach.

Behavioural systems [4, 57] based on reactivity have gained widespread attention. In order to satisfy the requirement of the fastest possible response, reactivity should be implemented at the lowest level. The shortest possible loop starts with proprioceptors and directly influences the real effector. As the activities of the reactive layer rely mainly on sensor input and to a very limited degree on internal state (i.e. memory) it is very easy to reproduce this layer within an embodied agent, where the quickest responses are organized within the virtual effector and are based on proprioceptive input. An example of this approach is the utilisation of force/torque sensing directly within the virtual effector (e.g. [92]). If a subsumption architecture [13, 14] is to be reproduced, its modules can be represented by blocks within the DFD definition of a transition function by using inhibiting and suppressing links. Otherwise, the modules can deliver the results of their computations to a composition unit, which will compute the final outcome, subsequently dispatched to a real effector for execution. If a slower reaction is permissible then exteroceptive input from the virtual receptors can be used by the control subsystem, which will use the same scheme of operation as the virtual effector. Thus the same pattern will be reproduced at different levels of the control hierarchy. This can be continued at even higher levels of the hierarchy by using extra CT type agents (Section 2.2 or [93]).

Many systems have been designed utilising a muti-tiered architecture – usually employing either three-tier (3T) [28] or two-tier (2T) structures, e.g. NASREM (NASA Standard Reference Model) [50]. Components of each layer conform to the function of each layer. The lowest layer is behavioural. It is tightly coupled with actuators and sensors. The middle layer is the sequencing layer (sometimes termed as executive layer [44]), which is responsible for selecting behaviours that will eventually realise the task. The uppermost tier usually involves deliberation. In robotics deliberation is associated with planning. Planning itself can be devoted to different aspects of robot's activities, e.g. path planning, trajectory planning, task planning. However the most demanding form is task planning, as it requires symbolic representations of the mission as well as the environment and the robot itself – all of those representations should enable reasoning leading to control decisions (hence this layer usually requires the definition of an adequate ontology, e.g. [76, 74, 20]). This influences the architecture of the robot control system. On the one hand, the perception system has to transform the sensor readings into symbolic representations (e.g. K-CoPMan [59]), i.e. solve the anchoring problem [22], and on the other hand the sequencing layer has to act as demanded by the produced plan. The sequencing layer has to have access to the plan treated as data (e.g. CRAM_m [35]) or the planning layer can invoke the sequencing layer to produce the effects required by the plan (e.g. LAAS architecture [9])). Some systems, especially set in industrial environment, rely on one-time generation of plans, which are subsequently executed without an on-line planning service, e.g. [73]. As in non-industrial environments fixed plans rarely succeed, replanning has do be done. Moreover, planning requires handling of inconsistent or missing data [10]. The classical SPA architecture usually involves a heavy computational load associated with planning. Thus those computations should be delegated to the supporting computational agents (agents without receptors and effectors), which can either reside in the robot's control computer or be offloaded to a cloud [89].

From the above it is evident that planning and sequencing layers act together to accomplish the plan and in some systems are tightly bound together (e.g. CRAM [7], CLARAty [58]), hence 2T instead of 3T architecture results. The proposed embodied agent based architecture can accommodate all of those possibilities. By design an embodied agent forms a two tier architecture: virtual entities controlling the real devices form the first tier and the control subsystem forms the second one.

A computationally demanding planning task can be decomposed into several parts implemented in several interacting CT agents. In the simplest case the CERT agent is supplemented by one computational agents (CT type) producing plans based on the internal imperative to solve a certain problem or on demand from the user. The planning process can be described in terms of transition functions, taking in percepts from CERT agents and the goal embedded in the internal memory of the control subsystem of a given CT agent, and producing a plan in the required form. Such a plan as a whole can be transmitted to a CERT agent, that subsequently interprets that plan, or the CERT agent obtains the plan as a sequence of consecutive commands executed one by one. In both cases, if the plan fails the CERT agent informs the CT agent to replan the activities based on new evidence obtained from the perception subsystem (i.e. virtual receptors). This reflects the 3T architecture.

The Syndicate architecture [63] extends the 3T architecture onto multi-robot systems. In this case the layers are not only interconnected hierarchically, but also horizontally with components of each layer interconnected between themselves, enabling direct communication at each level of abstraction. The architecture based on the concept of an embodied agent does not allow the virtual entities to communicate with each other, both internally to the agent and between the agents. However, if the embodied agents controlling the hardware are structured appropriately (i.e. the CERT agents), direct communication between virtual entities is not necessary. Even if it would be required it can be implemented indirectly through the control subsystem of the agent. Nevertheless, the CERT agents can be connected between themselves forming the bottom layer. Above this lowest layer, layers composed of mutually interconnected CT agents can be constructed, as postulated by the Syndicate architecture. Moreover, this structure of layers can be reproduced creating multi-tier systems. Both in robotics and artificial intelligence architectures based on blackboards are used. In that case agents cooperate with each other sharing and updating the data in the blackboard (e.g. [68]). Such an architecture is readily reproducible using the postulated approach. In this case the blackboard becomes a CT type agent catering to the needs of all other agents. Blackboards can also be utilised by the communication inter-subsystem layer, that has not been discussed here,

but is presented in [85]. Service Oriented Architecture (SOA) fosters another approach presenting devices as a collection of capabilities which are exposed as services [3]. Those capabilities can be represented as behaviours of the control subsystem of a device, including a robot. CT type agents can then organise publishing, discovering and arranging composite services. Thus the embodied agent-based approach seems to be flexible enough to fully support all of the above mentioned architectures. Moreover, focusing on the modelling of a robotics system aspect of the presented approach, it is clear that the derived model can also be used as a system pattern in Model Driven Engineering (MDE), e.g. [47, 15] and thus subsequently for automatic code generation.

6. Conclusions

The paper focused on the methodology of designing robot control systems. The proposed method is based on a general architecture (universal model) of an embodied agent which is subsequently tailored to the required hardware of the system and the task it has to execute. It requires the definition of the structure of the communication buffers and internal memory of the subsystems of the agent. Those buffers are used as arguments of transition functions that define the behaviours of subsystems. All behaviours are based on the same pattern, parametrised by transition functions and terminal and error conditions. The thus defined behaviours are assembled into the overall task of a subsystem by defining an adequate FSM, with the general task of a given embodied agent being realised by its control subsystem. As a result the approach leads to definition of both the system structure and activities.

We also provide an associated design procedure, leading to a detailed specification of the considered robotic system. As the design process of any complex system is inherently iterative, we describe the steps that the system architect should follow iteratively in order to refine the specification from general description to satisfactory level of details. Once the specification is detailed enough it is used for the implementation of the system. It should be noted that the transformation of an FSM, transition functions and conditions into code is straightforward in any programming language, so such a specification can be readily used in the implementation of the control system.

The resulting systematic, formalised and domain-specific approach, conveys in a synthetic form years of experience of the research team in the design of robot control systems. Thus we briefly discuss the evolution of our approach and indicate the most important robotic systems and turning points that influenced both the architectural model and the specification method, enabling them to mature. The diversity of those systems, both in terms of the hardware utilized and the executed tasks enables the formulation of the conclusion that this design method is general enough to be applied to any robotic system and is not constrained with respect to a hardware type, control paradigm or task. To support that statement we additionally discuss how several different robotic architectures present in the literature can be expressed through the prism of our approach. We provide examples showing how specific systems following reactive or deliberative as well as hybrid approaches can be described as agents and deliberate how multi-tier architectures can be composed by assembling agents into layers.

Only a full specification of a realistic system performing useful tasks can show the merits of the proposed design procedure and convince the reader of its utility. For this reason we present an example of a specification of a robotic system performing picking operation relying on visual perception combined with position-force control. Both the comprehensive presentation of the methodology and the exemplary specification resulting from it are novel, not presented earlier.

Acknowledgements. This project was supported by the National Science Centre according to the decision no. DEC-2012/05/D/ST6/03097, the National Centre for Research and Development grant no. PBS1/A3/8/2012 and grant of the Dean of Faculty of Electronics and Information Technology of Warsaw University of Technology no. 504/01446/1031/42.

References

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

[2] A. Aldoma, F. Tombari, R.B. Rusu, and M. Vincze, *OUR-CVFH–Oriented, Unique and Repeatable Clustered Viewpoint Feature Histogram for Object Recognition and 6DOF Pose Estimation*, Springer, 2012.

[3] S. Ambroszkiewicz, W. Bartyna, M. Faderewski, and G. Terlikowski, "Multirobot system architecture: environment representation and protocols", *Bull. Pol. Ac.: Tech.* 58 (1), 3–13 (2010).

[4] R. C. Arkin, *Behavior-Based Robotics*, MIT Press, 1998.

[5] C. Armbrust, L. Kiekbusch, T. Ropertz, and K. Berns, Soft robot control with a behaviour-based architecture, In *Soft Robotics*, pages 81–91. Springer, 2015.

[6] V. Azizi, A. Kimmel, K. Bekris, and M. Kapadia, Geometric reachability analysis for grasp planning in cluttered scenes for varying end-effectors, In *Automation Science and Engineering (CASE), 2017 13th IEEE Conference on*, pages 764–769. IEEE, 2017.

[7] M. Beetz, L. Mösenlechner, and M. Tenorth, CRAM_m – a cognitive robot abstract machine for everyday manipulation in human environments, In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, October 18–22, 2010, Taipei, Taiwan*, pages 1012–1017. IEEE, 2010.

[8] M. O. Ben Salem, O. Mosbahi, M. Khalgui, Z. Jlalia, G. Frey, and M. Smida, "Brometh: Methodology to design safe reconfigurable medical robotic systems", *The International Journal of Medical Robotics and Computer Assisted Surgery* 13 (3), e1786 (2017).

[9] S. Bensalem, L. de Silva, F. Ingrand, and R. Yan, "A verifiable and correct-by-construction controller for robot functional levels", *Journal of Software Engineering for Robotics*, 2 (1), 1–19 (2011).

[10] Ł. Białek, M. Borkowska, A. Borkowski, B. Dunin-Kęplicz, M. Gnatowski, A. Szałas, and J. Szklarski, "Coordinating multiple rescue robots", *Prace Naukowe Politechniki Warszawskiej. Elektronika*, (194), 185–196 (2014).

[11] F. Bonsignorio, "A new kind of article for reproducible research in intelligent robotics [from the field]", *IEEE Robotics & Automation Magazine* 24 (3), 178–182 (2017).

[12] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly, first edition, September 2008.

[13] R.A. Brooks, "A robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automation* 2 (1), 14–23 (1986).

[14] R.A. Brooks, "Intelligence without representation", *Artificial Intelligence* 47 (1-3), 139–159, January (1991).

[15] D. Brugali, "Model-driven software engineering in robotics", *IEEE Robotics Automation Magazine* 22 (3), 155–166, Sept (2015).

[16] H. Bruyninckx, The real-time motion control core of the OROCOS project, In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2766–2771, IEEE, September 2003.

[17] 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, August (1996).

[18] E. Cervera, "Try to start it! the challenge of reusing code in robotics research", *IEEE Robotics and Automation Letters* 4 (1), 49–56 (2019).

[19] H. Chen and B. Bhanu, "3d free-form object recognition in range images using local surface patches", *Pattern Recognition Letters* 28 (10), 1252–1262 (2007).

[20] D. Choiński and M. Senik, "Distributed control systems integration and management with an ontology-based multi-agent system", *Bull. Pol. Ac.: Tech.* 66 (5), 613–620 (2018).

[21] A. Collet, M. Martinez, and S.S. Srinivasa, "The MOPED framework: Object Recognition and Pose Estimation for Manipulation", *The International Journal of Robotics Research* 30 (10), 1284–1306 (2011).

[22] S. Coradeschi and A. Saffiotti, "An introduction to the anchoring problem", *Robotics and Autonomous Systems* 43 (2), 85–96 (2003).

[23] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O.C. Jenkins, Rosbridge: Ros for non-ros users, In *Robotics Research*, pages 493–504, Springer, 2017.

[24] W. Dudek, W. Szynkiewicz, and T. Winiarski, "Cloud computing support for the multi-agent robot navigation system", *Journal of Automation Mobile Robotics and Intelligent Systems* 11 (2), 67–74 (2017).

[25] M. Figat, C. Zieliński, and R. Hexel, Fsm based specification of robot control system activities, In *11th International Workshop on Robot Motion and Control (RoMoCo)*, pages 193–198, July 2017.

[26] S. Fleury, M. Herrb, and R. Chatila, "GenoM: A tool for the specification and the implementation of operating modules in a distributed robot architecture", *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'97)*, 2, 842–849, September (1997).

[27] E. Gamma, J. Vlissides, R. Johnson, and R. Helm, *Design patterns: elements of reusable object-oriented software*, Addison-Wesley, 1994.

[28] E. Gat, On three-layer architectures, In D. Kortenkamp, R.P. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210, AAAI Press Cambridge, MA, 1998.

[29] H. Golnabi and A. Asadpour, "Design and application of industrial machine vision systems" *Robotics and Computer-Integrated Manufacturing* 23 (6), 630–637 (2007).

[30] A.R. Graves and C. Czarnecki, "Design patterns for behavior-based robotics", *IEEE Transactions on Systems, Man, and Cybernetics – part A: Systems and Humans* 30 (1), 36–41 (2000).

[31] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes, In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 858–865, IEEE, 2011.

[32] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. GlentBuch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, et al., Bop: benchmark for 6d object pose estimation, In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[33] M. Hutson, "Artificial intelligence faces reproducibility crisis", *Science* 359 (6377), 725–726 (2018).

[34] M. Janiak and C. Zieliński, "Control system architecture for the investigation of motion control algorithms on an example of the mobile platform Rex", *Bull. Pol. Ac.: Tech.* 63 (3), 667–678 (2015).

[35] J. Winkler, M. Tenorth, A.K. Bozcuoglu, and M. Beetz, CRAM$_m$ – memories for robots performing everyday manipulation activities, In *2nd Annual Conference on Advances in Cognitive Systems*, pages 91–108, 2013.

[36] S. Kaisler, *Software Paradigms*, Wiley Interscience, 2005.

[37] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again, In *Proceedings of the International Conference on Computer Vision (ICCV 2017), Venice, Italy*, pages 22–29, 2017.

[38] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation", *IEEE Journal on Robotics and Automation* 3 (1), 43–53 (1987).

[39] M. Kopicki, R. Detry, M. Adjigble, R. Stolkin, A. Leonardis, and J.L. Wyatt, "One-shot learning and generation of dexterous grasps for novel objects", *The International Journal of Robotics Research*, 2015.

[40] T. Kornuta and M. Stefańczyk, "Modreg: a modular framework for rgb-d image acquisition and 3d object model registration", *Foundations of Computing and Decision Sciences* 42 (3), 183–201 (2017).

[41] T. Kornuta, T. Winiarski, and C. Zieliński, Specification of abstract robot skills in terms of control system behaviours, In R. Szewczyk, C. Zieliński, and M. Kaliczyńska, editors, *Progress in Automation, Robotics and Measuring Techniques. Vol. 2. Robotics*, volume 351 of *Advances in Intelligent Systems and Computing (AISC)*, pages 139–152, Springer, 2015.

[42] T. Kornuta and C. Zieliński, "Robot control system design exemplified by multi-camera visual servoing", *Journal of Intelligent & Robotic Systems* 77 (3–4), 499–524 (2015).

[43] D. Kortenkamp and R. Simmons, Robotic systems architectures and programming, In O. Khatib and B. Siciliano, editors, *Springer Handbook of Robotics*, pages 187–206, Springer, 2008.

[44] D. Kortenkamp, R. Simmons, and D. Brugali, Robotic systems architectures and programming, In B. Siciliano and O. Khatib, editors, *Springer Handbook of Robotics, 2nd Edition*, pages 283–306, Springer, 2016.

[45] A. Koubâa, *Robot operating system (ros): The complete reference*, volume 2, Springer, 2017.

[46] R. Laganiere, *OpenCV 3 Computer Vision Application Programming Cookbook*, Packt Publishing Ltd, 2017.

[47] M. Lauder, M. Schlereth, S. Rose, and A. Schürr, "Model-driven systems engineering: state-of-the-art and research challenges", *Bull. Pol. Ac.: Tech.* 58 (3), 409–421 (2010).

[48] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A.C. Berg, Ssd: Single shot multibox detector, In *European conference on computer vision*, pages 21–37, Springer, 2016.

[49] D.G. Lowe, "Distinctive image features from scale-invariant keypoints", *International Journal of Computer Vision* 60 (2), 91–110 (2004).

[50] R. Lumia, J. Fiala, and A. Wavering, "The nasrem robot control system standard", *Robotics and Computer-Integrated Manufacturing* 6 (4), 303–308 (1989). Special Issue Robots in Manufacturing.

[51] D.M. Lyons, *Prerational intelligence*, volume 2: Adaptive behavior and intelligent systems without symbols and logic of *Studies in cognitive systems*, chapter A Schema-Theory Approach to Specifying and Analysing the Behavior of Robotic Systems, pages 51–70, Kluwer Academic, 2001.

[52] 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, June (1989).

[53] D.M. Lyons, R.C. Arkin, S. Jiang, M. O'Brien, F. Tang, and P. Tang, "Performance verification for robot missions in uncertain environments", *Robotics and Autonomous Systems* 98, 89–104 (2017).

[54] M. Mason, "Compliance and force control for computer controlled manipulators", *IEEE Transactions on Systems, Man, and Cybernetics* (11), 418–432 (1981).

[55] A.T. Miller and P.K. Allen, "Graspit! a versatile simulator for robotic grasping", *Robotics & Automation Magazine, IEEE* 11 (4), 110–122 (2004).

[56] M. Muja and D.G. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, In *VISAPP (1)*, pages 331–340, 2009.

[57] S. Müller, P. Wolf, K. Berns, and P. Liggesmeyer, Combining behavior-based and contract-based control architectures for behavior optimization of networked autonomous vehicles in unstructured environments, In *Commercial Vehicle Technology 2018*, pages 324–335, Springer, 2018.

[58] I. Nesnas, The CLARAty project: Coping with hardware and software heterogenity, In D. Brugali, editor, *Software Engineering for Experimental Robotics*, pages 9–30, Springer–Verlag, 2007.

[59] D. Pangercic, M. Tenorth, D. Jain, and M. Beetz, Combining perception and knowledge processing for everyday manipulation, In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan*, pages 1065–1071, 2010.

[60] C.R. Qi, H. Su, K. Mo, and L.J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation", *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE* 1 (2), 4 (2017).

[61] R.B. Rusu and S. Cousins, 3D is here: Point Cloud Library (PCL), In *International Conference on Robotics and Automation*, Shanghai, China, 2011.

[62] S. Sabour, N. Frosst, and G.E. Hinton, Dynamic routing between capsules, In *Advances in Neural Information Processing Systems*, pages 3856–3866, 2017.

[63] B. Sellner, F.W. Heger, L.M. Hiatt, R. Simmons, and S. Singh, "Coordinated multiagent teams and sliding autonomy for large-scale assembly", *Proceedings of the IEEE – Special Issue on Multi-Robot Systems* 94 (7), 1425–1444, July (2006).

[64] D. Seredyński and W. Szynkiewicz, Fast Grasp Learning for Novel Objects, In R. Szewczyk, C. Zieliński, and M. Kaliczyńska, editors, *Recent Advances in Automation, Robotics and Measuring Techniques*, volume 440 of *Advances in Intelligent Systems and Computing (AISC)*, pages 681–692. Springer, 2016.

[65] D. Seredyński, T. Winiarski, K. Banachowicz, and C. Zieliński, Grasp planning taking into account the external wrenches acting on the grasped object, In *Robot Motion and Control (RoMoCo), 10th International Workshop on*, pages 40–45. IEEE, 2015.

[66] B.R. Shetty and M.H. Ang, Active compliance control of a puma 560 robot, In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 4, pages 3720–3725, IEEE, 1996.

[67] K. Shoemake, Animating rotation with quaternion curves, In *ACM SIGGRAPH computer graphics*, volume 19, pages 245–254, ACM, 1985.

[68] P. Skrzypczyński, "Multi-agent software architecture for autonomous robots: A practical approach", *Management and Production Engineering Review* 1 (4), 55–66, December (2010).

[69] M. Staniak, T. Winiarski, and C. Zieliński, Parallel visual-force control, In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[70] M. Staniak and C. Zieliński, "Structures of visual servos", *Robotics and Autonomous Systems* 58 (8), 940–954 (2010).

[71] M. Stefańczyk and T. Kornuta, Handling of asynchronous data flow in robot perception subsystems, In *Simulation, Modeling, and Programming for Autonomous Robots*, volume 8810 of *Lecture Notes in Computer Science*, pages 509–520, Springer, 2014.

[72] M. Stefańczyk, M. Laszkowski, and T. Kornuta, WUT Visual Perception Dataset – a dataset for registration and recognition of objects, In *Challenges in Automation, Robotics and Measurement Techniques*, volume 440 (2) of *Advances in Intelligent Systems and Computing (AISC)*, pages 635–645, Springer, 2016.

[73] M. Stenmark and J. Malec, "Knowledge-Based Instruction of Manipulation Tasks for Industrial Robotics", *Robotics and Computer Integrated Manufacturing*, 33, 56–67, June (2015).

[74] M. Stenmark, J. Malec, K. Nilsson, and A. Robertsson, "On distributed knowledge bases for robotized small-batch assembly", *IEEE Transactions on Automation Science and Engineering* 12 (2), 519–528 (2015).

[75] C. Szegedy, S. Ioffe, V. Vanhoucke, and A.A. Alemi, Inception-v4, inception-resnet and the impact of residual connections on learning, In *AAAI*, volume 4, page 12, 2017.

[76] M. Tenorth and M. Beetz, "KnowRob: a knowledge processing infrastructure for cognition-enabled robots", *International Journal of Robotics Research* 32 (5), 566–590, May (2013).

[77] M. Walęcki, K. Banachowicz, and T. Winiarski, Research oriented motor controllers for robotic applications, In K. Kozłowski, editor, *Robot Motion and Control 2011 (LNCiS) Lecture Notes in Control & Information Sciences*, volume 422, pages 193–203, Springer Verlag London Limited, 2012.

[78] T. Winiarski and K. Banachowicz, Automated generation of component system for the calibration of the service robot kinematic parameters, In *20th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR*, pages 1098–1103, IEEE, 2015.

[79] T. Winiarski, K. Banachowicz, M. Walęcki, and J. Bohren, Multibehavioral position-force manipulator controller, In *21th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR*, pages 651–656, IEEE, 2016.

[80] T. Winiarski and A. Woźniak, "Indirect force control development procedure", *Robotica* 31 (3), 465–478 (2013).

[81] Y. Zhao, T. Birdal, H. Deng, and F. Tombari, "3d point-capsule networks", *arXiv preprint arXiv:1812.10775*, 2018.

[82] C. Zieliński, The MRROC++ system, In *Proceedings of the First Workshop on Robot Motion and Control, RoMoCo'99*, pages 147–152, June 1999.

[83] C. Zieliński, A Quasi-Formal Approach to Structuring Multi-Robot System Controllers, In *Second International Workshop on Robot Motion and Control, RoMoCo'01*, pages 121–128, 18–20 October 2001.

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

[85] C. Zieliński, M. Figat, and R. Hexel, "Communication within multi-fsm based robotic systems", *Journal of Intelligent & Robotic Systems* 93 (3-4), 787–805 (2019).

[86] C. Zieliński and T. Kornuta, Generation of linear Cartesian trajectories for robots using industrial motion-controllers, In *16th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR*, pages 62–67, August 2011.

[87] C. Zieliński and T. Kornuta, Diagnostic requirements in multi-robot systems, In *Intelligent Systems in Technical and Medical Diagnostics*, volume 230 of *Advances in Intelligent Systems and Computing (AISC)*, pages 345–356, Springer, 2014.

[88] C. Zieliński, T. Kornuta, and T. Winiarski, A systematic method of designing control systems for service and field robots, In *19-th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR*, pages 1–14. IEEE, 2014.

[89] C. Zieliński, M. Stefańczyk, T. Kornuta, M. Figat, W. Dudek, W. Szynkiewicz, W. Kasprzak, J. Figat, M. Szlenk, T. Winiarski, K. Banachowicz, T. Zielińska, E. G. Tsardoulias, A. L. Symeonidis, F.E. Psomopoulos, A.M. Kintsakis, P.A. Mitkas, A. Thallas, S.E. Reppou, G.T. Karagiannis, K. Panayiotou, V. Prunet, M. Serrano, J.-P. Merlet, S. Arampatzis, A. Giokas, L. Penteridis, I. Trochidis, D. Daney, and M. Iturburu, "Variable structure robot control systems: The RAPP approach", *Robotics and Autonomous Systems* 94, 226–244 (2017).

[90] C. Zieliński, W. Szynkiewicz, T. Winiarski, M. Staniak, W. Czajewski, and T. Kornuta, "Rubik's cube as a benchmark validating MRROC++ as an implementation tool for service robot control systems", *Industrial Robot: An International Journal* 34 (5), 368–375 (2007).

[91] C. Zieliński and T. Winiarski, "General specification of multi-robot control system structures", *Bull. Pol. Ac.: Tech.* 58 (1), 15–28 (2010).

[92] C. Zieliński and T. Winiarski, "Motion generation in the MRROC++ robot programming framework", *International Journal of Robotics Research* 29 (4), 386–413 (2010).

[93] C. Zieliński, T. Winiarski, and T. Kornuta, Agent-based structures of robot systems, In J. Kacprzyk and et al, editors, *Trends in Advanced Intelligent Control, Optimization and Automation*, volume 577 of *Advances in Intelligent Systems and Computing*, pages 493–502, 2017.