Architecture of an Autonomous Robot at the IT Level

Submitted: 17th October 2014; accepted: 28th December 2014

Stanisław Ambroszkiewicz, Waldemar Bartyna, Kamil Skarżyński, Marcin Stępniak, Maciej Szymczakowski

DOI: 10.14313/JAMRIS_1-2015/5

Abstract:
The paper proposes an architecture for a control system of an autonomous robot as well as an architecture for a multi-robot system in which the robots cooperate in order to accomplish client's tasks. The solution is based on the SOA paradigm and an ontology as a way of representing an environment, and is specified at the Information Technology level. This approach is focused on intuitive cooperation with a human and automation of a task execution, as well as automation of handling exceptions and changes in the environment. For the purpose of the test scenarios, a sample ontology was created, which allows the human user to define tasks to be performed by a robot. Additionally, a simulation environment was designed and implemented in Unity. It allows for automatic generation of the visual representation of information defined in the ontology, and for testing the effectiveness of the proposed architecture in different types of scenarios with variable sets of services (devices).

Keywords: *ontology, multi-robot system, environment representation*

1. Introduction

The aim of the RobREx project is to develop a set of technologies and appropriate architecture necessary for the production of autonomous service and terrain robots, in particular rescue and exploration robots. Test tasks should include: penetration, inspection and intervention. They require the creation of an appropriate ontology (a priori knowledge about the structure of the environment) to build the environment models and their particular instances, i.e. maps. For example, a robot should "know" that it is located in a building and not in a forest, i.e. it should be familiar with the overall structure of the building. Then the robot perception (associated with map updating) will be focused on recognizing specific types of objects such as corridors, stairs, doors, and furniture.

Common (for humans and robots) representation of the environment enables, on the one hand, easy definition of tasks to be performed by a robot, and on the other hand, processing and execution of these tasks in the environment. The formal structure of the representation is crucial because it allows for automatic processing of its instances (i.e. maps), so they can be *understood* by the robots. The representation is based on the general idea proposed in [13,14] as a formal approach to object-oriented programming language for robots. It has been extended by an additional hierarchy between objects and abstract objects (such as *a space* with its attribute being the air *temperature* in this space). The separation of the process of creating maps from the representation of the environment itself is crucial. Naturally, there is a priori knowledge about the overall structure of the representation; people have this knowledge and they can enter it into the system by means of friendly graphical interfaces. The robot receives knowledge about the environment, usually in the form of a partial map (the overall structure of its surroundings), which can be updated and refined by the robot itself depending on the requirements associated with the performed tasks. A general and universal structure of representation for a wide class of possible environments is needed. The concept of such a structure has a close relationship with the notion of ontology found in Computer Science. The classical definition of ontology [5] is as follows: an ontology is an explicit specification of a conceptualization; It is a formal description of concepts and of relations between these concepts. Conceptualization should be understood as an abstract, simplified model of the world needed to achieve complex goals. This model is created by identifying the important concepts related to these goals and may include, for example, objects, their attributes and relations that occur between objects. Model specification must be *formal* (i.e. definitions of terms and relation must be unambiguous) to allow for its automatic processing, and must be *common*, because it has to support cooperation in a multi-robot environment.

The ontology is a set of concepts and relations that occur between them, i.e. objects and their attributes, and the relations between objects. Each object is of a certain, pre-defined type. The object type is defined by attributes that objects of this type have, and by the internal (hierarchical) structure of such objects, which may consist of sub-objects and relations between these sub-objects. An elementary type has no internal structure (so is defined only by a set of attributes), and a complex type has such a structure. Ontology is defined as a hierarchical collection of types of objects (see [1]). Primary attributes and primary relations are the key elements of which object types are constructed. The object itself, as an instance of its type, is defined by assigning specific values to its attributes and by specifying relations. Primary attributes and primary relations must be *measurable* and *recognizable* by sensory devices in the system. In addition to classical approaches to the representation of an environment in robotics (i.e. metric, topologi-

Fig. 1. Visualization of the test environment

cal and hybrid) object-oriented approaches are also proposed (e.g. [3, 11]). They focus on creating maps of simple geometric shapes or objects, such as lines or wall. Cognitive approaches (e.g. [2, 4, 6, 7, 12]), in turn, attempt to mimic human perception. There are other, (but not directly for applications in robotics), more complex approaches, e.g. BIM (Building Information Modeling) and SIS (Spatial Information Systems). These approaches are based on automatic mapping with very little (or no) a priori knowledge about the environment.

The proposed approach assumes that the robot has a priori knowledge in the form of an ontology, which defines a general representation of the environment through the subordination relations between object types (e.g. overall structure of an estate or a building), but its instance, a map, specifies details, i.e., how many floors the building has, what are the distributions and dimensions of the rooms on the floors, and so on. Robot has (usually) a partial map, which it has to update and refine (learn about) through the recognition of objects based on their characteristics and relations.

A similar approach to providing knowledge for a robot is represented by the KnowRob [9] project. It allows for the acquisition and merging of knowledge from multiple heterogeneous sources. In both approaches, relations can be inferred from the values of object attributes, but in the proposed solution one can also define them in a more general way, e.g., only needs to specify that a jar is in a cabinet. It is not necessary to know the exact position. It can be detected by the robot during the task realization. Relations can also be define and store in this form in the ontology. This allows for easier and more intuitive task defining by people by declaring the desired situation through those relations.

The combination of object-oriented concepts and cognitive maps [6, 11], context learning [8] and a priori knowledge in the form of ontologies, can be the basis for the construction of new algorithms for object recognition.

Updating and refinement of maps (contextual learning) should be done automatically through recognition of objects based on their attributes and relations. The learning process involves determining the context dynamically and determining the probability of occurrence of specific types of objects in this context. The context is understood as the general structure of the environment (a subset of the possible types of objects), in which the robot is currently located. It is reasonable to assume that the robot knows the probability distribution (determined on the basis of recent experience) of several possible types. For example, the robot located in a building (an object of *building* type), has only a probability distribution (for rooms, corridors, elevators, etc.) suggesting where exactly in the building it is currently located. In addition, the recognition of objects (e.g. objects in a room: desks, chairs, etc.), the robot should have also a probability distribution (conditional, depending on previously processed data from the sensors) for the occurrence of objects of these types. For example, when the robot moves in the direction of a strong light source, it should expect, with a high probability, that the data from the camera will create an image of a window. Consequently, it should segment the image in order to find attributes and relations that describe an object of *window* type. There is extensive literature on recognition and context learning (e.g. [8, 10]). In our innovative approach an ontology (a priori knowledge) is applied as a basis for the context construction which results in a new quality in the design of efficient algorithms for recognition and learning.

The basis of the method used by a robot to learn how to recognize objects, is a dynamic context as the current state of the robot environment. Thus, the learning process involves estimating of:

(1) the probability distributions for the types of environments in which the robot can currently be, and

(2) the probability distributions of possible types of objects that the robot can currently recognize.

These estimations depend on the a priori knowledge, i.e. the ontology, and robot experience acquired in the past. The learning process should be integrated with the robot architecture.

2. The Environment Representation

Representation of the environment is an abstract, simplified model of the world. From many features of physical objects only those relevant to the realization of pre-defined classes of tasks are selected. This applies to the attributes of objects and the relations between them. This approach refers to a representation that human uses. The representation is common for people and devices (robots) because it has to provide means for automated and reliable cooperation. User (human) via a graphical interface defines the tasks to be performed by a robot, that receives the knowledge about the environment in the form of partial maps (which it can update and refine).

The hierarchical and object-oriented ontology as well as maps for indoor and outdoor environments are based on attributes and relations between objects forming the environment. The ontology formalism uses technologies similar to these in OWL (Web Ontology Language http://www.w3.org/TR/owl-features/) and is based on XML documents created according to the pre-defined XSD schemas.

Concept Glossary and Map Repository are development tools used, respectively, to define and manage types of elementary and complex objects, or to define and manage object maps (mostly partial that may require additional refinement depending on the capabilities of available devices).

Similarly to OWL, Semantic Web and Web Services, separate namespaces for attributes, types, and relationships are created. On the basis of structures defined in Glossary, XSD schemas are generated in order to validate the data when creating instances of types in Map Repository. In the prototype implementation of the system, Glossary is a module of Map Repository

Object types are defined by specifying a set its attributes. An attribute consists of the following data:

- a brief human-understandable name, e.g. length;
- namespace;
- basic type and range of values that can be assigned to the attribute (e.g. string, integer, floating point, etc.);
- the type of measurement unit (e.g. length, weight, money, time);
- optionally, a human-understandable description.

The second basic concept used while constructing ontology is a relation between objects. The relationship is defined by specifying:

- a human-understandable name, e.g. "*is on*";
- namespace;
- arguments of the relation and their types;
- evaluation of relation between objects is done by evaluating relations between attributes of these objects;
- optionally, a human-understandable description.

Based on the defined attributes and relations one can construct new complex structures, i.e., types and objects – instances of these types.

Object type consists of a set of attributes and, optionally, other object types (so it contains sub-types) and also inherit after a specified base type. Inheritance means that, in addition to its own attributes and subtypes, it also has attributes and sub-types of the inherited type.

In the test scenario (see Fig. 1), a closed jar filled with granules is placed in a cabinet. The goal of the robot task is to transfer the granules form the jar to a bowl on a table. The necessary steps include locating the cabinet, opening its door, removing the jar and unscrewing it, locating a bowl, pouring the content of the jar into the bowl and then closing the jar, putting the jar back into the cabinet and closing its door.

For the purpose of the scenario, complex types (shown in Fig. 2 in the form of data structures) were created. All objects that can be positioned in a global coordinate system must inherit the type *PlacedMapObject*, which has three attributes (*PositionX*, *PositionY*, *PositionZ*) corresponding to the position in three-dimensional coordinate system. Object orientation is described by the remaining attributes (*AngleX*, *AngleY*, *AngleZ*). Two basic geometric shapes, *Cuboid* and *Cylinder*, were defined. *Jar* and *JarClousure* inherit the *Cylinder* type. *Table* contains (aggregates) five objects of type *Cuboid* representing its legs and its tabletop. *Cabinet* consists of six objects of type *Cuboid* (left, right, bottom, top and back side of a cabinet with a single shelf), *Open* attribute (indicating whether the cabinet is open), and an object of *Door* type. The *Door* object inherits the Cuboid type and aggregates an object of *Handle* type. This type includes information necessary to infer the type of grip, which a robot has to use in order to open the door. *Room* is the type of the highest object in the object map hierarchy in the test scenario. It has (in addition to its own structure) references to all previously mentioned objects defined as relations of type *IsIn*. In the object map there are also examples of other relations such as: *IsEdgeGlued* (e.g. two walls of the room), *IsOn* (e.g. a bowl is on the table), *IsAttachedTo* (the handle is attached to the door).

3. The System Architecture

The proposed system architecture (see Fig. 3) was designed according to SOA (*Service Oriented Architecture*) paradigm. The capabilities of devices (here, autonomous robots) are treated as services (they also include tasks such as searching and recognizing).

From the system point of view, it is important to know how to invoke a service (i.e. what kind of input parameters are required and what kind of output parameters are produced), and not how the service is executed.

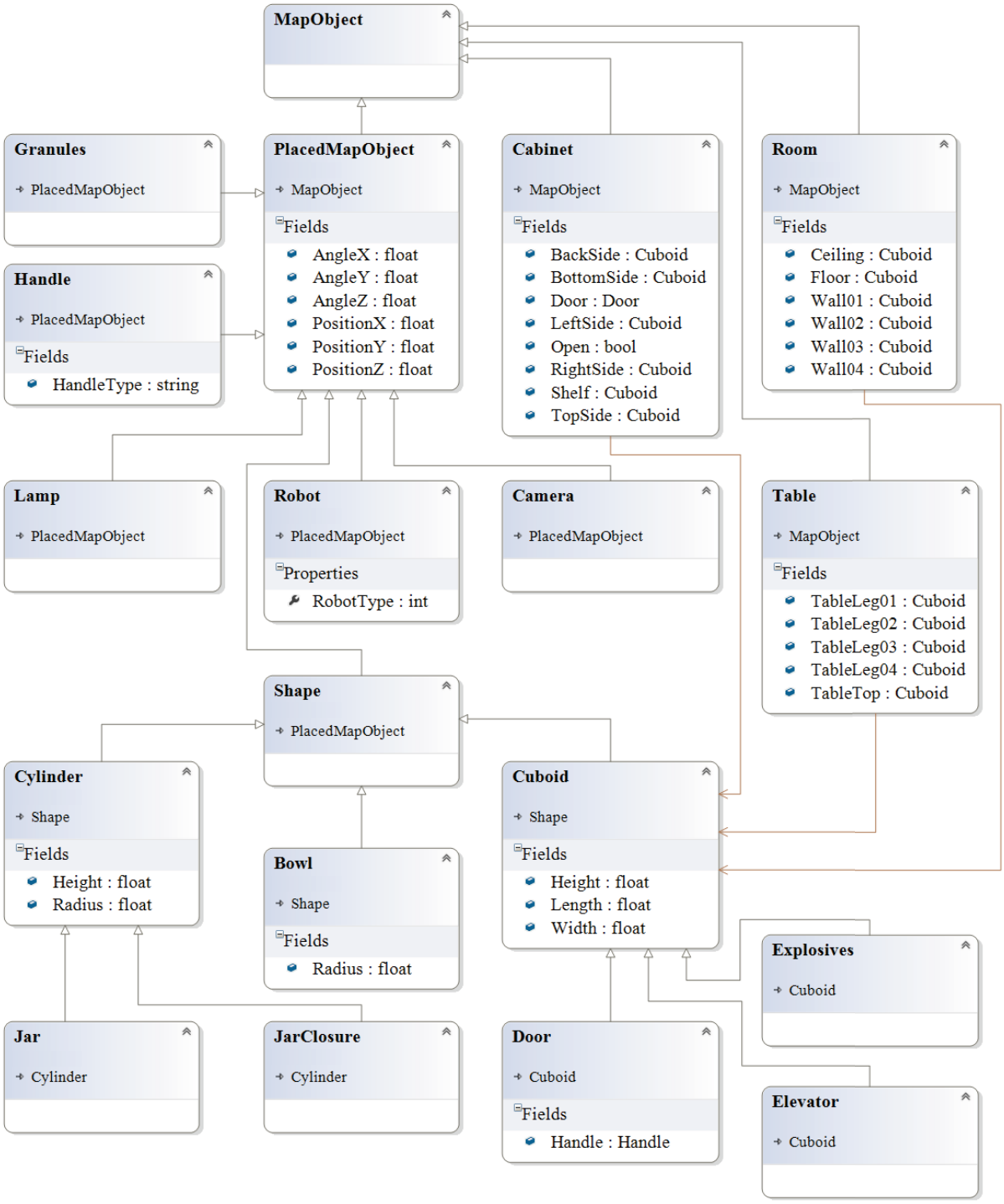

Fig. 2. The data structure of the environment representation. The black arrows represent inheritance, and the orange arrows represent aggregation

Services are registered in *Service Registry* in the form of their interfaces which include information such as service type, network address, range of work, input / output parameters, validity period of the entry. The data stored in Registry is used by *Task Manager*, which allows the customer to define, in a declarative way, a task by describing the initial situation (optional) and the final situation. Situations are described by specifying relations between relevant objects defined in *Repository*. Then an appropriate abstract plan is chosen from a pool of existing plans or is generated automatically. The abstract plan is defined as a series of types of services.

A service is executed by appropriate *Service Manager*, which can be seen as a broker between a device and the system. Each device provides a set of functions (behaviors) which are being invoked during service execution. The plan of the service execution (a series of invocations of device functions, conditions, and queries to Repository) is being assigned while defining the service. Service Manager, after receiving a task, accomplishes it by executing an appropriate plan, and

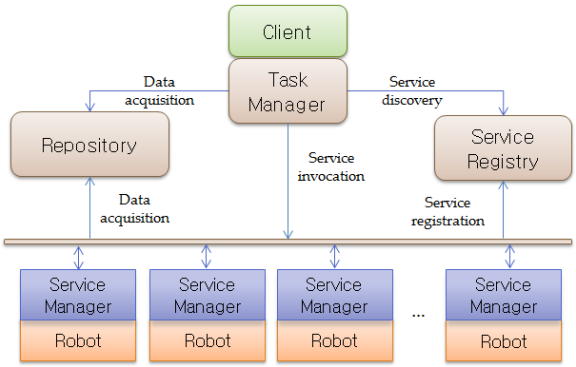

Fig. 3. The system architecture

updates the object map (stored in Repository), if there were changes in the environment caused by the device. If the service cannot be realized (e.g., the relation described in the task initial situation is no longer true), an appropriate message is sent to Task Manager. Then it can attempt to reconfigure the abstract plan by using different types of available services, and continue with the task execution. The tasks are defined by users in a declarative manner. The user describes the situation she/he wants to become true after completing the task. In addition to defining the final situation, she/he can also specify the initial situation. A sample task may look as follows:

– *initial situation*: granules are in the jar,

– *final situation*: granules are in the bowl.

The user, while defining the task, chooses the appropriate objects from the object map. The objects are identified by their *Name* attribute. Task Manager chooses an abstract plan that can *convert* the initial situation to the final situation. The plan may consist of one or more steps in the form of service types. The sample task can be realized by one service of type *move_content*. In the arrangement phase, Task Manager queries the Service Registry for a set of services of that specific type that can realize this particular task (the single step of the plan). Then, TM sends arrangement requests to each of the services found. Based on their responses, it chooses the optimal one for the task. The selection criteria may include, for example, the estimated time of task execution, or amount of required resources. Task Manager invokes the selected service, which in turn, replies with description of situation after executing the task. If the situation is not equivalent to the final situation of the task, Task Manager may try to invoke another service of the appropriate type, or use an alternative plan. Actions taken by Task manager in these situations depend on its policies of handling exceptions and transactions.

Service Manager, after agreeing to execute the task in the arrangement phase and invocation, realizes given task by proceeding according to a service execution plan. The plan is defined as a series of queries to Repository and robot functions (its skills) and for the *move_content* service may look as follows (see Fig. 4):

1) query Repository for information about object specified in the initial situation (the object from which the content has to be moved),
2) set the robot arm in the default position,
3) move to the location of the initial object,
4) grab initial object with the arm,
5) query Repository for information about object specified in the final situation (the object to which the content has to be moved),
6) move to the location of the final object,
7) turn the arm over the final object.

The plan also includes conditional statements (realized as an invocation of robot perception) which allow for checking if a particular situation (described by relations between objects) is true in the robot surroundings. Depending on its result another step is taken or an exception is returned (describing the current situation related to the task being executed). Invoking the above functions with appropriate parameters leads to achieving the final situation of the task, but the service execution plan may also include returning the initial object to its original location (see Fig. 5).

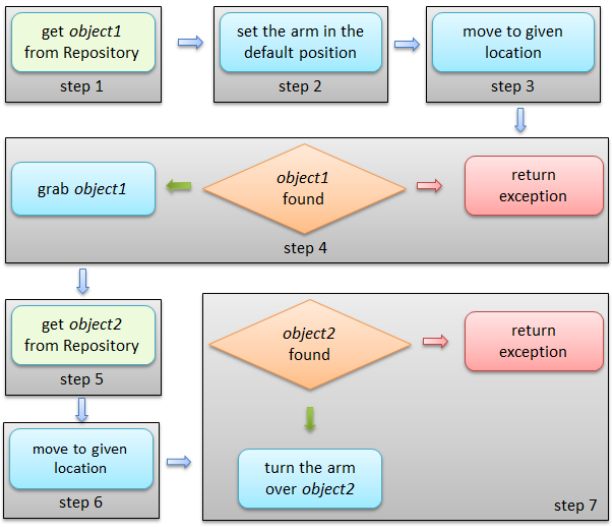

Fig. 4. Diagram representing a sample plan of a move_content service execution. Green squares represent queries to Repository, blue squares – invocation of a robot function, orange diamonds – invocation of a robot perception

Functions of the robots may be implemented in various technologies and on various hardware, therefore dedicated modules are required to run them. Once developed, a module can be used later to run other functions created in the same technology or for the same type of hardware.

A service execution plan should also handle exceptions, i.e. reactions to failures. An example of an exception may be a situation in which there is no jar in the cabinet, or the jar is empty. Service Manager should update the object map in Repository and send information about the current situation to Task Manager. In turn, it decides what the robot should do next, and how to proceed with the task. In this context, the learning process (i.e. estimating the most likely situation) on

Fig. 5. Realization of the sample task in the simulation environment; a) grabbing the initial object, b)turning the robot arm over the final object, c)driving to the original location of the initial object, d) returning the initial object to its original position.

the basis of previous and current observation is particularly important.

A simulation environment is developed in Unity 3D. It will serve as a platform for testing and validating proposed specifications of the environment representation (ontology), the architecture of a multi-robot system at the level of IT, and learning algorithms.

The simulation environment will be developed in order to implement more complex scenarios involving multiple services (devices). They will also include cognitive services (such as searching and map updating) which allow for observing and adjusting to changes in a dynamic environment. Ultimately, experiments will be carried out in a real environment and on real devices represented by theirs Service Managers.

4. Conclusions

The paper presents the problems associated with the representation of the robot's environment and possible solution in the form of an ontology. Along with the architecture of the multi-robot system, it supports automation of accomplishing complex tasks defined by a human in an environment of cooperating devices.

In comparison with the existing approaches, it is more focused on intuitive cooperation with human and automation of a task execution, as well as automation of handling exceptions and changes in the environment during the task execution.

ACKNOWLEDGEMENTS

This work was carried out within the project "RobREx - Autonomy for rescue and exploration robots", grant NRDC no PBS1 / A3 / 8/2012. Marcin Stępniak is partially supported by the Foundation for Polish Science under International PhD Projects in Intelligent Computing. Project financed from The European Union within the Innovative Economy Operational Programme 2007-2013 and European Regional Development Fund.

AUTHORS

Stanisław Ambroszkiewicz – Institute of Computer Science, Polish Academy of Sciences, ul. Jana Kazimierza 5, 01-248 Warsaw, Poland, e-mail: sambrosz@ipipan.waw.pl, www: http://www.ipipan.waw.pl/mas/stan/.

Waldemar Bartyna – Institute of Computer Science, University of Natural Sciences and Humanities, Siedlce, Poland, e-mail: wbartyna@gmail.com.

Kamil Skarżyński – Institute of Computer Science, University of Natural Sciences and Humanities, Siedlce, Poland, e-mail: kamilskar@gmail.com.

Marcin Stępniak* – Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland, e-mail: martinus.st@gmail.com.

Maciej Szymczakowski – Institute of Computer Science, University of Natural Sciences and Humanities, Siedlce, Poland, e-mail: maciek.szymczakowski@gmail.com.

*Corresponding author

REFERENCES

[1] S. Ambroszkiewicz, W. Bartyna, M. Faderewski, and G. Terlikowski, "Multirobot system architecture: environment representation and protocols", *Bulletin of the Polish Academy of Sciences: Technical Sciences*, vol. 58, no. 1, 2010, 3–13.

[2] D. Anguelov, R. Biswas, D. Koller, B. Limketkai, S. Sanner, and S. Thrun, "Learning hierarchical object maps of non-stationary environments with mobile robots". In: *Proc. of the 17th Annual Conference on Uncertainty in AI (UAI)*, 2002.

[3] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun, "Towards object mapping in dynamic environments with mobile robots". In: *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland, 2002.

[4] C. Galindo, A. Saffiotti, S. Coradeschi, P. Buschka, J.-A. Fernandez-Madrigal, and J. González, "Multi-hierarchical semantic maps for mobile robotics". In: *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2005, 3492–3497.

[5] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?", *International Journal of Human-Computer Studies*, vol. 43, no. 5–6, 1995, 907 – 928.

[6] M. E. Jefferies and W.-K. Yeap, eds., *Robotics and cognitive approaches to spatial mapping*, volume 38 of *Springer Tracts in Advanced Robotics*, Springer, 2008.

[7] B. Kuipers, "The spatial semantic hierarchy", *Artif. Intell.*, vol. 119, no. 1-2, 2000, 191–233.

[8] A. Oliva and A. Torralba, "The role of context in object recognition", *Trends in Cognitive Sciences*, vol. 11, no. 12, 2007, 520 – 527.

[9] M. Tenorth and M. Beetz, "Knowrob – knowledge processing for autonomous personal robots". In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 2009, 4261–4266.

[10] A. Torralba, "Contextual priming for object detection", *International Journal of Computer Vision*, vol. 53, no. 2, 2003, 169–191.

[11] S. Vasudevan, S. Gächter, V. Nguyen, and R. Siegwart, "Cognitive maps for mobile robots-an object based approach", *Robotics and Autonomous Systems*, vol. 55, no. 5, 2007, 359–371.

[12] W. Yeap and M. Jefferies, "On early cognitive mapping", *Spatial Cognition and Computation*, vol. 2, no. 2, 2000, 85–116.

[13] C. Zieliński, "Torbol: an object level robot programming language", *Mechatronics*, vol. 1, no. 4, 1991, 469 – 485.

[14] C. Zieliński, "Description of semantics of robot programming languages", *Mechatronics*, vol. 2, no. 2, 1992, 171 – 198.