The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

CRAM — A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments

Michael Beetz, Lorenz Mösenlechner, Moritz Tenorth
Intelligent Autonomous Systems Group
Department of Informatics, Technische Universität München
Boltzmannstr. 3, D-85748 Garching
{beetz,moesenle,tenorth}@cs.tum.edu

Abstract— **This paper describes CRAM (Cognitive Robot Abstract Machine) as a software toolbox for the design, the implementation, and the deployment of cognition-enabled autonomous robots performing everyday manipulation activities. CRAM equips autonomous robots with lightweight reasoning mechanisms that can infer control decisions rather than requiring the decisions to be preprogrammed. This way CRAM-programmed autonomous robots are much more flexible, reliable, and general than control programs that lack such cognitive capabilities. CRAM does not require the whole domain to be stated explicitly in an abstract knowledge base. Rather, it grounds symbolic expressions in the knowledge representation into the perception and actuation routines and into the essential data structures of the control programs. In the accompanying video, we show complex mobile manipulation tasks performed by our household robot that were realized using the CRAM infrastructure.**

I. Introduction

We investigate autonomous robot control systems that enable robots to perform complex everyday manipulation activities in human living environments. Such control systems are, for instance, needed for autonomous household robots. The design, implementation, and deployment of robot control systems for such complex applications is a challenging and intense programming task that requires powerful software tools. To respond to these needs, a number of middle-ware software libraries that support the development of distributed modular control systems have been developed. These middleware systems include ROS [1], Player [2], Yarp [3], and Orocos [4]. There is, however, a lack of powerful software tools that enable programmers to effectively and efficiently implement higher-level capabilities such as learning, knowledge processing, and action planning into the robot control programs to produce more flexible, reliable and efficient behavior.

Such cognitive mechanisms are needed because a robot which performs everyday manipulation tasks must continually decide which actions to perform and how to perform them. Even the seemingly simplest tasks such as picking up an object from a table require complex decision making. The robot must decide where to stand in order to reach the object, which hand(s) to use, how to reach for it, which grasp type to apply, where to grasp, how much grasp force to apply, how to lift the object, how much force to apply to lift it, where to hold the object, and how to hold it. The decision problems are even more complex because many decisions depend on the task context, which requires the robot to take many more factors into account to achieve the best performance, or at least a performance that is good enough.

Cognitive architectures [5] have been proposed as an answer to this need. However, general-purpose cognitive architectures such as the 3T architecture [6] or Icarus [7] have not yet demonstrated to be capable of improving the robot problem-solving performance. More recent cognitive architectures including the RobotCub [8] and the Paco+ [9] architectures have been successfully applied to autonomous robot control but are tailored to specific paradigms such as developmental robotics or the acquisition and use of object-action complexes.

In this paper, we propose CRAM (*Cognitive Robot Abstract Machine*) as a software toolbox for the design, the implementation, and the deployment of cognition-enabled autonomous robots performing everyday manipulation activities. CRAM provides a language for programming cognition-enabled control systems. This language includes data structures, primitive control structures, tools and libraries that are specifically designed to enable and support mobile manipulation as well as cognition-enabled control. The CRAM toolbox facilitates the implementation of complex control programs that make decisions based on world percepts as well as acquired knowledge. It provides tools for integrating first-order reasoning in the control programs and, on the other hand, reason about the control programs.

The backbones of CRAM are the CRAM Plan Language (CPL) and the knowledge processing system KnowRob [10]. CPL is a very expressive behavior specification language for autonomous robots that enables them to not only execute their control programs, but also to reason about and manipulate them — even during their execution. This becomes possible since CPL provides key aspects of the robot's control program as persistent *first-class objects*. That is, these aspects (like object specifications, failure descriptions, or decisions the robot needs to take) are not only compiled pieces of code, but exists as objects that can be inspected, queried and reasoned about. That allows the control program to analyze automatically why it failed to achieve a goal, or when it had wrong beliefs about the world.

KnowRob [10], the second backbone of CRAM is a knowledge processing system particularly designed for autonomous personal robots that provides CPL with the knowl-

978-1-4244-6676-4/10/$25.00 ©2010 IEEE

edge required for taking decisions. KNOWROB is a first-order knowledge representation based on description logics and provides specific mechanisms and tools for action-centric representation, for the automated acquisition of grounded concepts through observation and experience, for reasoning about and managing uncertainty, and for fast inference — knowledge processing features that are particularly necessary for autonomous robot control.

While CRAM is a research endeavor that is still in its early stages, there are already a number of contributions it makes to cognition-enabled control of autonomous robots. Some of the concepts of the plan language CPL were already proposed in the context of its predecessor language RPL [11], but CPL is particularly designed to control real physical robots. Therefore, the system has been revised and improved to support feedback loops of at least 10Hz.[1] Another key feature is the tight coupling between CPL and the data structures generated, updated, and used by the lower-level control modules. That means that in contrast to layered architectures that need to abstract away from these low-level data structures, in CPL they are still available to the complete program and can be used for decision making. This way the plan-based controller can use much more detailed, accurate, and realistic models of robot control where needed.

Similar properties hold for KNOWROB. KNOWROB can include information from data structures used by lower-level control modules into its belief state. To this end, programmers define how the truth value of particular predicates can be computed on demand from the respective data structures. We call this kind of predicates *computable predicates*.

CRAM is designed to be *lightweight*, *standard*, and *configurable*. *Lightweight implementation* means that the code for the implementation should be very concise. We achieve this by using and extending the Common Lisp language and by using existing Lisp compilers rather than building a plan language with its own interpreter, as it has been done for RPL. As a consequence, the kernel of CPL is roughly only 3000 lines of code. CRAM is based on Common Lisp and Prolog, both *standardized* languages for which free compilers and a big variety of text books, tutorials and other documentation exist. Thirdly, we have realized CRAM such that only a small kernel is needed and the system can easily be *configured* and extended with additional functionality if needed. This way, a programmer has to satisfy restrictions and coding conventions only for those components that are needed. For example, plans can be written in a simpler way and do not have to contain annotations that would allow a reasoner to reconstruct goals, the structure of sub-plans and the belief state if the robot is not supposed to reason about the course of action anyways.

The remainder of this paper is organized as follows. We start with discussing an example plan realized in CRAM. The following section describes the CRAM system, including the plan language, the knowledge processing system, and the high-level reasoning module. Afterwards, we introduce some extensions to CRAM for realizing robots performing everyday manipulation tasks and list some further extension modules that provide additional cognitive capabilities.

[1] Voluntary human action control supposedly runs in feedback loops of about 10Hz.

II. CRAM PLANS

Before we come to the different components of the CRAM system, let us first show how a system engineer can program cognition-enabled control programs using CRAM. Below is an example plan for picking up the object ?obj.

```
(def-goal (achieve (object-in-hand ?obj))
  (with-designators
        (pickup-place ...)
        (grasp-type ...)
        (pickup-reaching-traj ...)
        (lift-trajectory ...)
    (when (and (holds-bel (object-in-hand ?curr-obj) now)
               (obj-equal ?curr-obj ?obj))
      (succeed (object-in-hand ?obj)))
    (at-location pickup-place
      (achieve (arm-at pickup-reaching-traj))
      (achieve (grasped grasp-type))
      (achieve (arm-at lift-trajectory))
      (succeed (object-in-hand ?obj))
```

The first thing to notice is that we specify an achievement goal with the desired world state (object-in-hand ?obj) as the argument, instead of writing a procedure declaration of the form pick-up (?obj). Explicitly representing the desired world state has several advantages: A state like (object-in-hand ?obj) cannot only have achievement routines, but also routines for perception (perceive (object-in-hand ?obj)), belief (holds-bel (object-in-hand ?obj)), or other goals like communication, prevention or maintenance of the respective state. With this mechanism, plans can check whether the robot already believes a goal to hold before it tries to achieve it. The robot can also validate that a routine was successful by perceiving the state after the achievement was completed.

A second important aspect is that not only the goal, but also many control decisions, including complex reasoning, are turned into first-class objects, i.e. entities that parametrize the plan and that can be queried, analyzed, and reasoned upon. In the code, these decisions are specified in the with-designators construct. The decisions are described by the a set of constraints that need to be met, essentially a set of attribute-value pairs. These descriptions are only resolved at the very last moment when the decision actually needs to be taken. This allows the system to base the decision on all information that can be gathered until the code that needs to take the decision gets executed. For example, the action parameter pickup-reaching-traj is initially specified as:

```
(a hand-trajectory
   (purpose      (pick-up ?obj))
   (type         motion-plan)
   (objective    (minimize torque)))
```

which refers to a motion trajectory computed by the motion planner with the optimization criterion *minimum torque*. If the robot, at a later point in time, sees the object and notices that it is a glass filled with a fluid, it adds an additional motion constraint (motion-constraint keep-obj-upright). Then, right before the robot intends to reach for the object, it passes this abstract trajectory specification to the motion planner in order to get an executable specification for the

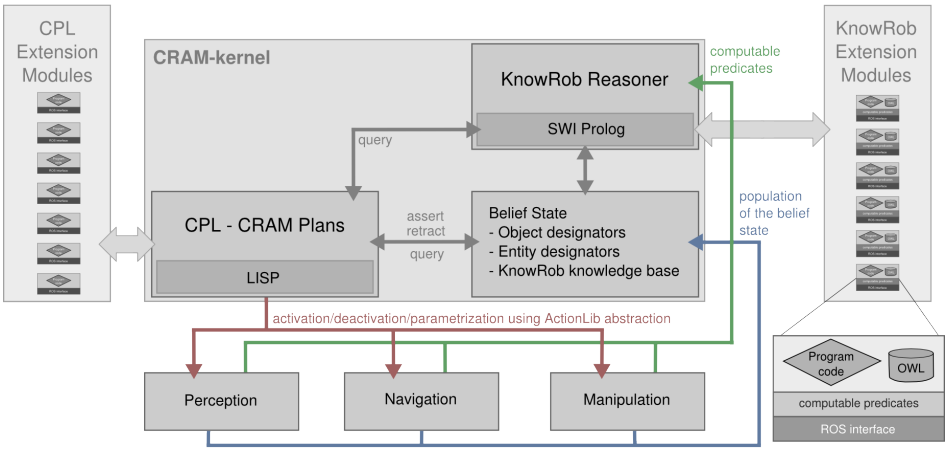

Fig. 1. Overview of the CRAM system. The CRAM kernel consists of the CPLplan language and the KNOWROB knowledge processing system. Both are tightly coupled to the perception and actuation components. CRAM is realized in a highly modular way and can be extended with plug-ins providing additional cognitive capabilities.

reach. Please note that the resolution of a designator is not heuristics-based but tightly integrated the KNOWROB knowledge processing kernel. For instance, to get a three-dimensional pose for the location (a location (on table)) we query the semantic environment map to find all objects that are a table and calculate a location that matches the current context. Realizing control decisions as persistent first-class objects enables the robot to collect context information that affect the respective decision, and then make the decision in a maximally informed manner right before it is needed. Besides trajectories, we represent objects and locations as similar first-class objects.

Another interesting property of this plan is that it is "universal" [12]. It specifies how to achieve the object being at its destination for various contexts: If the object is grasped, the plan only signals success; if the robot does not already have the object in its gripper, it has to pick it up first, etc.

III. CRAM SYSTEM ARCHITECTURE

Figure 1 gives an overview of the CRAM system. CRAM is built using the ROS middle-ware [1], i.e. the different building blocks are realized as ROS nodes and available as open-source software from the public tum-ros-pkg repository (`http://tum-ros-pkg.sourceforge.net/`).

Plans written in the CRAM plan language CPL (Section III-A) specify concurrent, sensor-guided, reactive behavior: They specify how the robot is to respond to sensory events, changes in the belief state and detected plan failures. The belief state is continually updated with information on performed actions and with data from passive perception mechanisms. The plan can also query the KNOWROB reasoner III-B to infer the knowledge necessary for informed control decisions and to determine parameterizations for context-dependent action execution. The CRAM kernel can be extended with optional modules providing additional cognitive capabilities. Extensions can add to both CPL and KNOWROB and contribute additional knowledge, program code for specialized inference, or both. On a more abstract level, the CPL plans themselves become entities the robot can perform reasoning on, which is done in the COGITO layer explained in Section IV.

A. CPL — *the* CRAM *Plan Language*

The way programs are written looks at a first glance very similar to classical architectures such as BDI architectures. There are, however, many modifications that are tailored to autonomous manipulation robots, as well as design decisions that aim at CRAM being a very pragmatic software infrastructure for reliable, sensor-guided feedback control.

Maybe the biggest difference is the plan language. In most architectural approaches where plan languages are designed to be reasoned about, these languages are typically restricted to a very limited set of control structures to avoid that the reasoners have to automatically understand, reason about, and revise general robot control programs. As a consequence, most plan languages only consider plans as partially ordered sets of atomic actions. Concurrency, action synchronization, failure handling, loops and reactiveness exist only in rudimentary forms, if at all. This is most obvious in the so-called 3T architectures, in which software engineers introduce a middle layer into the architecture that is responsible for translating partially ordered plans into flexible and reliable execution routines, for translating the feedback back into the abstract plan language, trigger replanning and handle errors locally. The biggest problem with these architectures is that it is hard to find abstractions that are suitable for planning in time: On the one hand, they must reduce complexity enough to make planning feasible, but on the other hand, they should not abstract away information that is necessary for decision making.

In contrast, CRAM provides CPL, a plan language that is designed to specify flexible, reliable, sensor-guided robot behavior. It merges the features found in the upper two layers of 3T architectures by providing control structures for parallel execution and the partial ordering of sub-plans, support for sophisticated failure handling and the semantic annotation of the control program to allow for reasoning about it without the need to completely understand the program.

TABLE I

SOME CPL CONTROL STRUCTURES AND THEIR USAGE.

Control Structure	Example of its usage
in parallel do p_1 ...p_n	**in parallel do** *navigate*(⟨235,468⟩) *build-grid-map*()
try in parallel p_1 ...p_n	**try in parallel** *detect-door-with-laser*() *detect-door-with-camera*()
with constraining plan p b	**with constraining plan** *relocalize-if-nec*() *deliver-mail*()
plan with name N_1 p_1 ... **with name** N_n p_n **order** $n_i \prec n_j$	**plan with name** S_1 *put-ontable*(C) **with name** S_2 *put-on*(A,B) **with name** S_3 *put-on*(B,C) **order** $S_1 \prec S_3$, $S_3 \prec S_2$

The abstract machine provides the same control structures as RPL [11] for reacting to asynchronous events, for coordinating concurrent control processes, and for using feedback from control processes to make the behavior robust and efficient. Table I lists several low-level control structures that are provided by CPL and can be used to specify the interactions between concurrent control sub-plans. The control structures differ in how they synchronize sub-plans and how they deal with failures. What makes them different from a multi-threading library is that these control structures generalize the concept of a hierarchical (sequential) program to sub-routines that may run in parallel and still form a tree. This allows for complex synchronization, partial orderings, blocking and in particular for clean failure handling that propagate from a sub-plan to its parent.

The **in parallel do**-construct executes a set of sub-plans in parallel. The sub-plans are activated immediately after the **in parallel do** plan is started. The **in parallel do** plan succeeds when all of his sub-plans have succeeded. It fails after one of its sub-plans has failed. An example use of **in parallel do** is mapping the robot's environment by navigating through the environment and recording the range sensor data. The second construct, **try in parallel**, can be used to run alternative methods in parallel. The compound statement succeeds if one of the sub-plans succeeds. Upon success, the remaining sub-plans are terminated. **with constraining plan** *P B*, the third control structure, means "execute the primary activity *B* such that the execution satisfies the constraining plan *P*." Policies are concurrent sub-plans that run while the primary activity is active and interrupt it if necessary. Finally, the **plan**-statement has the form **plan** STEPS CONSTRAINTS, where CONSTRAINTs have the form **order** $S_1 \prec S_2$. STEPS are executed in parallel except when they are constrained otherwise. Additional concepts for the synchronization of concurrent sub-plans include semaphores and priorities.

B. KNOWROB

Knowledge processing and reasoning services in CRAM are provided by KNOWROB [10], a highly modular knowledge processing system especially developed for the needs of mobile robotics. It consists of a small core system and a large set of optional modules.

The KNOWROB core system is based on Prolog and contains methods for loading knowledge bases, for reasoning on their content, and for interfacing the system with the CPL plan language. KNOWROB provides functions that accept queries and return a set of bindings that satisfy the query. From the plan implementor's point of view, queries to KNOWROB are most often needed to make an if-then-else like decision, or to use the result of a query, bound to a variable, as a parameter. For the former case, we define the method knowrob-query(q) which returns a boolean value depending on whether or not q is implied by the knowledge base. The latter case is realized by the method knowrob-query-var(var, q) that returns the bindings of the query variable *var* which render the logical expressions of the query true.

Knowledge inside KNOWROB is represented in description logic in OWL. This declarative knowledge defines in some way the terms the robot describes the world in, for example the type of an object, its properties, and possible relations to other objects.

This rather static knowledge is extended by the so-called *computable predicates* which integrate external sources of information into the knowledge base. *Computable predicates* are evaluated by calling external procedures, for instance by querying the vision system, by calling specialized reasoning procedures, or by executing a small function that just computes if one object stands on top of another one. The result of the computation is linked to the semantic relation, e.g. the method *computeObjectOnTop* is linked to the OWL relation *on-Physical*.

There may be none, one, or several computable predicates attached to one relation, which facilitates adding and removing modules from the system. If a module is loaded, it just provides an additional way of computing some semantic relations. Modules in KNOWROB can contain extensions to the KNOWROB ontology, e.g. provide additional encyclopedic or common-sense knowledge, they can contribute specialized reasoning methods, or provide interfaces to external data sources. Some of the main modules in KNOWROB are:

- Knowledge imported from the Cyc ontology [13] and the OpenMind Indoor Common Sense database [14] that equips the robot with a detailed ontology of objects, actions, and events, as well as a reasonable amount of common-sense knowledge.
- The semantic map [15] represents objects in the environment in terms of OWL instances and allows for reasoning on their properties.
- An interface to the ProbCog system [16] integrates probabilistic reasoning into the system, so that relations can be computed using learned statistical relational models. As a result, KNOWROB receives either a probability distribution over different alternatives, or simply the most likely result.

The KNOWROB distribution provides many more modules for spatial and temporal reasoning, for loading information from the perception system and other ROS nodes, for clustering and classification, or for calculating the results of an

action by running simulations in a physical simulator. A graphical visualization module accepts object instances or observed action sequences and displays them.

IV. COGITO

The COGITO system adds a reasoning layer on top of the CRAM kernel. While the components of the CRAM kernel realize robot plans that include reasoning about the belief state and observations of the environment, the COGITO system performs reasoning about *these plans* in order to examine how well they perform, where they encounter problems, or when the robot's belief state was wrong. This information is then used to find flaws in plan execution, explain the outcome of plans and transform these plans in order to fix the detected flaws and improve the plan [17], [18]. For enabling this kind of "meta-reasoning", CPL represents all aspects of a plan, like tasks, failures, descriptions of objects and abstract entities like locations or trajectories, as first-class objects. That means that a plan is not only a piece of compiled code that is executed, but also a data structure that can be accessed during run-time or recorded to create a persistent execution trace. Information that is stored in such an execution trace includes the values of plan parameters at any point in time, results of sub-plans, their status, failures etc.

Plan execution dynamically creates a tree of task objects, each containing the information mentioned above. This information is stored as a first-order representation that allows to make inferences and answer queries, e.g. for variable values at a particular plan state. This explicit representation of goals, perception tasks, and beliefs enables the robot to reason about the plan and the roles of sub-plans. It can, for example, infer how it achieved a goal by extracting the subgoal hierarchy out of the plan, or it can infer why tasks are performed, namely to achieve certain super-tasks.

V. CRAM$-EM$

The CRAM kernel contains only the very basic, general functionality. Extension modules equip the system with application-specific cognitive capabilities. CRAM$-EM$ (CRAM for Everyday Manipulation) is a set of extension modules for robots that perform everyday manipulation tasks, providing methods for recognizing and localizing objects, for reaching towards and grasping them, and for determining a good robot base location for manipulation.

Designators: When acting in the real world, the robot needs to resolve the abstract, symbolic object descriptions used in its high-level planning system to physical objects in the environment. For this task, it needs information about their properties, e.g. the object recognition algorithm that can best identify this kind of object. The robot also needs to determine if two entities, perceived at different points in time, are the same object. On the action generation side, the robot needs information about action properties, e.g. parameters of trajectories or places where objects are stored.

Such information is stored using *designators* for objects, locations, or other entities like trajectories. Which physical object a designator corresponds to, or which trajectory should be chosen for an action, can change over time when additional information becomes available. For instance, when the robot is to grasp a mug and the perception routines detect that the mug is filled with coffee, the ongoing manipulation action needs to be reconfigured to take the additional constraint into account that the mug is to be kept upright. CRAM uses an implementation of the Rete [19] algorithm to propagate updates in designator properties. The Rete algorithm implements a network of designators that connects input tokens to productions. Whenever an input token is asserted or retracted, it propagates through the network, and matching productions (combinations of tokens) are executed.

Object perception: When interacting with objects, a robot needs a powerful perception system that can recognize these objects and determine their poses in space. CRAM uses the K-COPMAN [20] cognitive perception system that comprises techniques for active perception of the task-relevant objects, as well as passive perception of the robot's environment that builds a memory of perceived objects.

Prediction: Being able to predict the effects of actions is important for verifying that they lead to the desired state, for checking their safety and optimizing their execution. CRAM includes a powerful projection module that uses detailed, realistic, physical simulation to execute a plan and record log data [18]. The recorded execution traces can be accessed through a first-order representation that is computed on demand, similar to computables, and enable the system to both reason on past experiences and predict future situations.

VI. COGNITIVE EXTENSIONS

CRAM can be equipped with a set of extension modules that provide additional cognitive capabilities for improved perception, learning, and adaptation.

Action awareness: Tools for the observation, interpretation and analysis of human actions are provided by the AMEVA [21] system. It contains a marker-less human motion capture system, techniques for segmenting human motions and building hierarchical action models, and algorithms for learning statistical models of complete activities.

Learning and adaptation: The Robot Learning Language [22] provides methods for logging experience data and performing learning on these data for improving the performance of the planning system.

Transformational planning: The transformational planning system TRANER [23] extends CPL with methods for transforming plans. Abstract transformation rules can be applied to plans to re-order actions, to insert or remove actions, and to change their parameters. Examples of such transformations are to stack plates or to use a container for transporting objects. Observations of humans can be used for determining which combination of transformations is likely to lead to good performance [24].

Web-enabled robot control: Plans [25] and object models [26] can be imported from the World Wide Web in order to extend the task repertoire and the set of objects the robot can recognize.

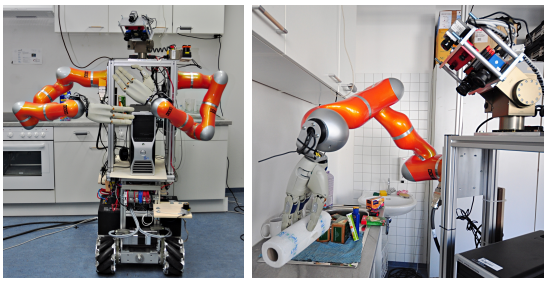

Fig. 2. The TUM Rosie robot that is controlled by the CRAM architecture presented in this work.

VII. Experiments

The CRAM system is being developed on the TUM Rosie robot platform shown in Figure 2. TUM Rosie is a mobile robot designed to perform everyday manipulation tasks in a kitchen environment. While the whole system is still under development, CRAM has already shown to be useful for the control of a complex robotic system performing object manipulation activities. Please have a look at the following video that shows the robot manipulating household objects: `http://www.youtube.com/watch?v=MFVBoIzHcjo`.

VIII. Conclusions

With CRAM, we presented a toolkit and language for specifying complex behavior of cognitive mobile robots. The CRAM kernel provides methods for synchronizing parallel behavior, handling execution failures and reacting to changes in the world. The plan language is tightly integrated with a knowledge representation system that provides the commonsense knowledge required for successful operation in human environments. CRAM is designed in a very modular way: There is a stack of extension modules called CRAM$-EM$ for everyday manipulation tasks, and several other modules for resolving abstract entity descriptions, for cognitive perception, for interpreting human actions, transforming the robot's plans, or for retrieving task descriptions from the web. The system is available as open-source software and can be downloaded for free. We demonstrate its capabilities in the accompanying video that shows our robot performing everyday manipulation tasks controlled by CRAM.

IX. Acknowledgments

This work is supported in part within the DFG excellence initiative research cluster *Cognition for Technical Systems – CoTeSys*, see also `www.cotesys.org`.

References

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *In IEEE International Conference on Robotics and Automation (ICRA 2009)*, 2009.

[2] B. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*, 2003, pp. 317–323.

[3] P. Fitzpatrick, G. Metta, and L. Natale, "Towards long-lived robot genes," *Robotics and Autonomous Systems*, vol. 56, no. 1, pp. 29–45, 2008.

[4] R. Smits, T. D. Laet, K. Claes, P. Soetens, J. D. Schutter, and H. Bruyninckx, "Orocos: A software framework for complex sensor-driven robot tasks," *IEEE Robotics and Automation Magazine*, 2008.

[5] D. Vernon, G. Metta, and G. Sandini, "A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 2, pp. 151–180, 2007.

[6] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack, "Experiences with an architecture for intelligent, reactive agents," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 1, 1997.

[7] P. Langley and D. Choi, "A unified cognitive architecture for physical agents," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 21, no. 2. 2006, p. 1469.

[8] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori, "The iCub humanoid robot: an open platform for research in embodied cognition," in *PerMIS: Performance Metrics for Intelligent Systems Workshop. Aug*, 2008, pp. 19–21.

[9] "Perception, Action and Cognition through learning of Object-action complexes," http://www.paco-plus.org/, 2005.

[10] M. Tenorth and M. Beetz, "KnowRob — Knowledge Processing for Autonomous Personal Robots," in *IEEE/RSJ International Conference on Intelligent RObots and Systems.*, 2009.

[11] D. McDermott, "A Reactive Plan Language," Yale University," Research Report YALEU/DCS/RR-864, 1991.

[12] M. J. Schoppers, "Universal plans for reactive robots in unpredictable environments," in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, J. McDermott, Ed. Milan, Italy: Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1987, pp. 1039–1046.

[13] C. Matuszek, J. Cabral, M. Witbrock, and J. DeOliveira, "An introduction to the syntax and content of Cyc," *Proceedings of the 2006 AAAI Spring Symposium*, pp. 44–49, 2006.

[14] R. Gupta and M. J. Kochenderfer, "Common sense data acquisition for indoor mobile robots," in *In Nineteenth National Conference on Artificial Intelligence (AAAI-04*, 2004, pp. 605–610.

[15] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3D Point Cloud Based Object Maps for Household Environments," *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge)*, 2008.

[16] D. Jain, L. Mösenlechner, and M. Beetz, "Equipping Robot Control Programs with First-Order Probabilistic Reasoning Capabilities," in *International Conference on Robotics and Automation (ICRA)*, 2009.

[17] A. Müller, "Transformational planning for autonomous household robots using libraries of robust and flexible plans," Ph.D. dissertation, Technische Universität München, 2008.

[18] L. Mösenlechner and M. Beetz, "Using physics- and sensor-based simulation for high-fidelity temporal projection of realistic robot behavior," in *19th International Conference on Automated Planning and Scheduling (ICAPS'09).*, 2009.

[19] C. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence*, vol. 19, no. 1, pp. 17–37, 1982.

[20] 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.*, Taipei, Taiwan, 2010.

[21] M. Beetz, M. Tenorth, D. Jain, and J. Bandouch, "Towards Automated Models of Activities of Daily Life," *Technology and Disability*, vol. 22, 2010.

[22] A. Kirsch, "Robot learning language — integrating programming and learning for cognitive systems," *Robotics and Autonomous Systems Journal*, vol. 57, no. 9, pp. 943–954, 2009.

[23] A. Müller, A. Kirsch, and M. Beetz, "Transformational planning for everyday activity," in *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, Providence, USA, September 2007, pp. 248–255.

[24] M. Tenorth and M. Beetz, "Priming Transformational Planning with Observations of Human Activities." in *IEEE International Conference on Robotics and Automation (ICRA).*, 2010.

[25] M. Tenorth, D. Nyga, and M. Beetz, "Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web." in *IEEE International Conference on Robotics and Automation (ICRA).*, 2010.

[26] U. Klank, M. Z. Zia, and M. Beetz, "3D Model Selection from an Internet Database for Robotic Vision," in *International Conference on Robotics and Automation (ICRA)*, 2009.