

---

# CRAM<sub>m</sub> — Memories for Robots Performing Everyday Manipulation Activities

---

**Jan Winkler**

**Moritz Tenorth**

**Asil Kaan Bozcuoğlu**

**Michael Beetz**

WINKLER@CS.UNI-BREMEN.DE

TENORTH@CS.UNI-BREMEN.DE

ASIL@CS.UNI-BREMEN.DE

BEETZ@CS.UNI-BREMEN.DE

Institute for Artificial Intelligence, Universität Bremen, 28359 Bremen, Germany

## Abstract

Agents that learn from experience can profit immensely from memorizing what they have done, why, how, and what happened. For autonomous robots performing complex manipulation tasks, these memories include low level data, such as perceptual snapshots of relevant scenes that influenced decision making, detailed complex motions the robot performed, and effects of these motions. They also include high level representations of the intended actions and the belief-dependent decisions that led to the chosen course of action. In this paper, we propose CRAM<sub>m</sub>, a memory management system that can record very comprehensive and informative memories without slowing down the operation of the robot. CRAM<sub>m</sub> offers a query interface that allows the robot to retrieve the kinds of information stated above. This is done using a first-order logical language that provides predicates concerning the beliefs and intentions of the robot, its physical state, perceptual information, and action effects and their relations at various different levels of abstraction.

## 1. Introduction

Consider a robot that is supposed to prepare meals, set the table, clean up, load and unload the dishwasher, and so on. Such activities are commonly called “everyday activities”. Anderson (1995) defines an everyday activity as “a) a complex task that is both common and mundane to the agent performing it; b) one about which an agent has a great deal of knowledge, which comes as a result of the activity being common, and is the primary contributor to its mundane nature; and c) one at which adequate or satisficing performance rather than expert or optimal performance is required.” In this article, we investigate how robotic agents can be equipped with memories of previous activity episodes in order to build up the “great deal of knowledge” for competently performing everyday activities and to learn from their experience.

We propose CRAM<sub>m</sub>, a software infrastructure which equips robotic agents with a comprehensive memory of their experiences that allows a-posteriori reasoning, diagnosis and reconstruction of the believed world states at different points in time. CRAM<sub>m</sub> is an extension of CRAM (Cognitive Robot Abstract Machine), a framework for the implementation of cognition-enabled robot control systems (Beetz, Mösenlechner, & Tenorth, 2010). In the context of CRAM<sub>m</sub> we consider memories to be the information gained from past experience, i.e. from everyday manipulation episodes, that

robotic agents can access and use to improve their future activities (Wood, Baxter, & Belpaeme, 2012). As such, the information content of the memories can be measured in terms of the queries that can be answered based on the information contained in the memory.

CRAM<sub>m</sub> enables the robot to answer queries such as the following ones: *Which fetch tasks failed because the object could not be found?*, *Which kinds of failures could the “place” sub-plan not recover from?*, *Which items in the refrigerator often stand at the same position?* and *Did the robot block its view of the pot with its own arm when it put down the mug?* The questions above require the robot to memorize its poses, the images it has taken, its beliefs and intentions, information about objects in the world when executing its plan and the relation between these pieces of information. A robot capable of answering these questions is a robot that knows what it has done, how, and why, and what the results of its activities were (Brachman, 2002). The capability of answering such queries can inform robots to make better execution-time decisions and to revise plans to improve their expected performance.

In cognitive psychology, memories are categorized into short-term (STM) and long-term memory (LTM), where the STM is a small-capacity memory that provides the context for accomplishing the current task. The LTM is a high-volume memory that provides comprehensive information for all kinds of tasks. Wood et al. (2012) further categorize artificial memory types along other dimensions, for instance procedural versus declarative memories, where the declarative memory is often considered as the consciously accessible information, while the procedural memory is seen as compiled or subconscious information. Episodic memories are experienced event information that is temporally and spatially organized and combined with context information. *In this paper, we focus on declarative, episodic, long-term memories for robot manipulation episodes.*

Functionally, memorization can be divided into three distinct processes: encoding, storage, and retrieval. The encoding is concerned with observing the state of plan execution and the data streams that are sent between the different components, and mapping this data into memory structures that allow the answering of the queries above. The storage is concerned with accumulating the encoded data in a long-term memory while not affecting the overall system performance. Retrieval is concerned with how to answer queries using the data stored in the memory.

The contributions of this paper are (1) expressive memory representations that combine symbolic plan events with subsymbolic sensor data, (2) methods for temporal, spatial, diagnostic and causal reasoning operating on the symbolic and subsymbolic memory structures, and (3) efficient and scalable logging mechanisms that can build up these structures during task execution without negatively affecting the robot’s performance. We evaluate the system on log data collected in three different tasks (object perception, picking and placing an object, continuous arm movements) that pose different challenges to the sensor- and plan logs. To measure the information contained in the memories, we present a set of queries that cover a range of inference capabilities.

## 2. Cognition-enabled Robot Control and the CRAM system

Before introducing CRAM<sub>m</sub>, we would like to briefly explain the particularities of CRAM and our robot control systems and the consequences and opportunities of artificial memory design. The CRAM Plan Language CPL (Beetz, Mösenlechner, & Tenorth, 2010; Beetz, 2000) is a concurrent

reactive programming language that provides all the comfort of typical high level programming languages including a rich set of control structures that help to make the program robust and flexible as well as modular and transparent. The control program takes control decisions based on (possibly complex) inference processes. To this end, control decisions are often formulated as logical queries that evaluate to *true* or *false* or that compute values for parameterizing actions. To execute a task on the robot, the plans activate, parameterize, and deactivate modules of the robot’s distributed control system that provide different kinds of functionality such as object perception, robot navigation and localization, etc.

An important concept of the language are descriptions of entities such as objects, motion, grasps, or poses, which are called *designators* and which are first-class elements of the language. In the beginning of the plan execution, these descriptions are often vague, such as “the cup on the table”, leaving out situational context or detailed properties of what exactly is described. Plans are parameterized using such vague information and refine it when necessary. This way, plans can have qualitative parameters that allow much flexibility in how the task is executed, which are only quantified when the information they provide is really necessary for execution. Designators are refined whenever more information becomes available, for example when an object has been perceived. Since this information is not necessarily correct nor complete, designators can be revised with newer, more correct information as the result of reasoning processes or failures. When a designator is extended with further information, a new designator is created, holding the new, possibly more specific, description. Those two stages of description are then *equated*, i.e. linked together, to track the change of parameterization over time. The robot’s current *belief* about the world is described in terms of such designators. Especially the poses of objects in the environment and the robot’s own position are described in exactly this way.

The execution of a plan generates a tree of tasks, which are interpretation records of subplans and very similar to stack frames in program execution. When robot agents are assigned goals to achieve during plan execution, they must perform one or more tasks in order to do so. An arc from task  $t$  to task  $t_{sub}$  roughly represents that  $t$  called  $t_{sub}$  as a subplan. The data structures of the tasks include the local variables and their time-stamped value changes (see (McDermott, 1993) for a detailed description of this mechanism). Using these data structures we can define what the robot “believes”. For example, we can specify as logical rules that the robot believes to pick up a blue object if the plan parameter for the object acted on has an object description as its value where the color attribute has the value “blue”.

### 3. An Overview on CRAM<sub>m</sub>

The power of this plan language imposes requirements on the memory apparatus to be provided by CRAM<sub>m</sub>. CRAM<sub>m</sub> has to remember the relationship between plans and their subplans, it has to be able to reconstruct how a particular entity description looked like at a given state of plan execution, and it has to be able to reconstruct why the robot made a particular decision during plan execution. It equips robots with a comprehensive memory of their experiences that allows a-posteriori reasoning, diagnosis and reconstruction of the believed world states at different points in time. To enable such functionality, robots using CRAM<sub>m</sub> record

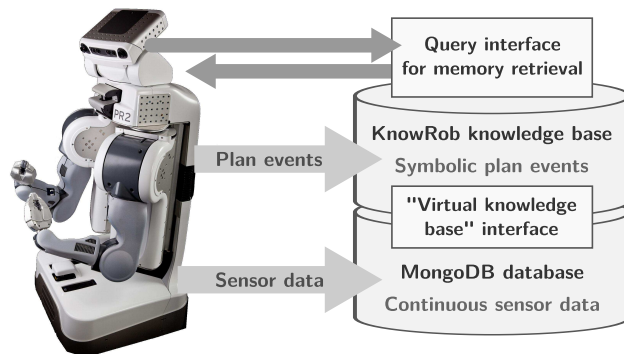


Figure 1: The proposed system architecture for recording live robot data includes logging mechanisms for continuous sensor data into a MongoDB database, and symbolic plan events into a KNOWROB knowledge base. A virtual knowledge base interface integrates continuous data with the symbolic knowledge base. A specialized query interface allows reasoning on the stored information.

1. their symbolic beliefs and intentions, the intended course of action (i.e., the task tree of the plan execution),
2. events and data from the perception system, and lower-level information like their position in the environment, their poses, etc, and
3. the relations between them. This includes the temporal synchronization using global time stamps, how sensor data and changes in data structures cause changes in the beliefs and intentions of the robot, and how the interpretation of plans causes changes in the world.

This information, coming from different sources, is combined to a timeline of events in the robot’s knowledge base that allows ontological, teleological, causal and temporal reasoning. Figure 1 depicts the main components of the system which will be explained in more detail in the following sections. During task execution, the robot records comprehensive action logs. Symbolic plan events are directly asserted to the knowledge base (Section 5.1), including the task tree of the robot’s control program, described as instances of the respective action classes, information about start and end times, references to manipulated objects, success and failure states, etc. Continuous data like sensor data or the frequently updated robot pose are stored in an efficient and scalable database which supports high-volume and high-frequency data recording without slowing down the robot (Section 5.2). Both data sources are described using the same representation language, explained in Section 4. The sensor data is integrated as a “virtual knowledge base” that provides an abstract query interface similar to the rest of the knowledge base. The representation forms the basis for sophisticated inference methods that can help the robot to take control decisions or diagnose plan failures which are described in Section 6.

The  $\text{CRAM}_m$  system builds upon the functionality of existing components like the CRAM executive, the KNOWROB knowledge base, the robot self-model in the SRDL language (Kunze, Roehm, & Beetz, 2011), and a tool for logging sensor data into a database. In this work, we have integrated these components and have added new modules for logging high level plan events and designators from the CRAM executive, for accessing the logged sensor data from the KNOWROB knowledge base, for computing spatial transformations based on the logged data, as well as predicates to reason about the combination of all this information.

#### 4. Formal representation of experiences

The representation of logged actions builds upon the action ontology of the KNOWROB robot knowledge processing system (Tenorth & Beetz, 2013) that provides structures to represent tasks as well as their spatial and temporal context, including events, objects, environment maps, and robot components. KNOWROB is implemented in PROLOG and represents knowledge using the Web Ontology Language OWL (W3C, 2009). As mentioned earlier, our memory consists of a knowledge base of logged plan events (stored in terms of OWL statements in the KNOWROB knowledge base) and a large-volume database with continuously-valued sensor data. To integrate both in a coherent representation that the robot can reason about, we use a special feature of the KNOWROB system that allows the definition of “virtual knowledge bases” on top of sub-symbolic data. Conceptually and from a query point of view, they appear like any other information stored in the knowledge base. However, instead of storing the information in preprocessed symbolic form, it is extracted on demand at query time from the stored data. This has several advantages: The same data can be used to compute different relations that do not have to be selected at recording time, the extracted symbols are inherently grounded, and the large-volume data can be recorded and stored using optimized databases, processing only what is needed to answer a query.

The KNOWROB ontology provides a conceptualization of the robotics domain as well as formalized background knowledge about the relation between actions, agents, goals, etc. For example, the “action” branch of the ontology contains about 130 action classes that form the building blocks for describing robot tasks. In addition to existing classes in the ontology that have so far been focused on the robot’s behavior in the outer world, we have added classes for describing control structures during task execution in order to be able to also reason about these aspects. The memory consists of assertions about occurrences of actions, represented as instances of these action classes, and assertions about the task context. The transitive *subAction* predicate links actions in the task hierarchy; references to objects and locations can be described using properties from KNOWROB such as *objectActedOn*, *fromLocation*, *toLocation*, etc. Due to the class–instance relationship between the robot’s plans and its logged experiences, it becomes very easy to retrieve examples of previous executions of an action from the memory.

Actions are represented as special kinds of events initiated by agents to achieve a desired effect. This makes it possible to describe these endogenous events using the same structures as exogenous events like sensor readings or utterances of a dialog partner. Figure 2 visualizes the representation of actions and external events using a pick-up task as example. The overall task *PickingUpAnObject* had the goal to bring object *Cup93* into the robot’s gripper. This task started at time point  $T_1$  and ended at time point  $T_{12}$ . Intermediate subtasks for perceiving, reaching, grasping, and lifting the object are described as *subAction* within the task tree and are therefore directly associated with the overall goal. Each event is characterized by its *startTime* and, if its duration is finite, its *endTime*. The KNOWROB system provides methods for reasoning about the timelines using Allen’s interval algebra (Allen, 1983). Events that are produced by other components of the robot’s distributed control system are usually not synchronized (e.g. the exogenous events in the lower part of Figure 2), but can be associated with the logged actions using temporal reasoning on the time stamps.

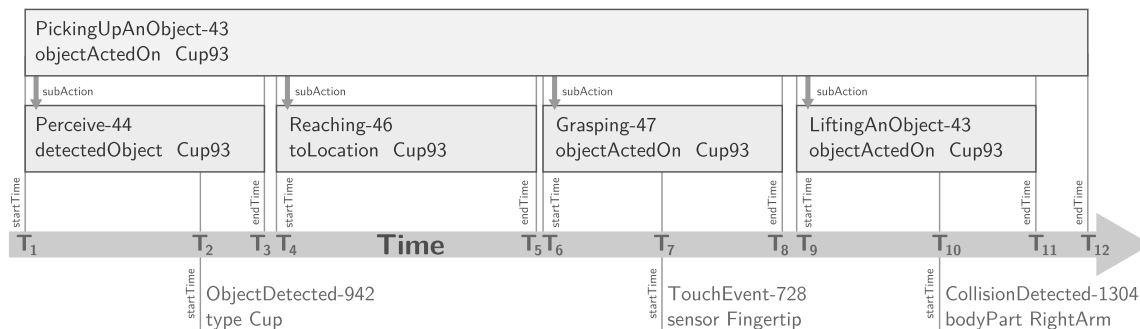


Figure 2: Example timeline of events for a pick-up task including its subtasks and a few external, instantaneous events. Temporal relations can be computed based on the start- and end times of the actions.

## 5. Encoding and Storage of Memory Contents

We distinguish between symbolic plan events and continuous-valued sensor signals which are logged using different mechanisms. Section 6 explains how information from both kinds of storage structures can be retrieved and combined in queries.

### 5.1 Logging Plan Events

As a modern robot plan language, the CPL allows splitting up complex goals into subgoals and plan primitives. CRAM provides mechanisms for defining goals, implementing the reasoning processes necessary for their parameterization, and ultimately performing these parameterized tasks. High level goals correspond to the *intentions* of the current plan execution while the sub-actions executed to achieve these goal reflect the progress and the dynamically inferred parameterization of the task at hand. This approach allows the distinction between different contexts in which each component is executed, for example which goals are currently active at different levels of the hierarchy.

CRAM<sub>m</sub> records the task tree including the task parameterizations, failures that arose during execution, the start and end times, success states of single subgoals, and the reported progress feedback from intermediate tasks. In addition, it stores when a designator is created (e.g. an object’s occurrence in the world is first mentioned) and when its information is updated, resulting in a change of belief about the world. This information is stored in the OWL representation language in the knowledge base.

Figure 3 shows a simplified example of a *perceive and pick* action, depicting several hierarchically connected tasks. The original task tree comprises roughly 250 actions and events, and 34 designators at 44 different timepoints, which we have pruned to improve readability. The top-level task to achieve the *object-in-hand* goal is decomposed into a task for perceiving the object and another one for actually grasping it. Task parameters are described by designators, as well as the perception results. The white box in the upper left visualizes the contents of the designator describing the detection of an object of type CONTAINER. This symbolic log is linked to sensor data recorded during the task, for example camera images, which are stored at important times during the task execution, or the robot’s pose.

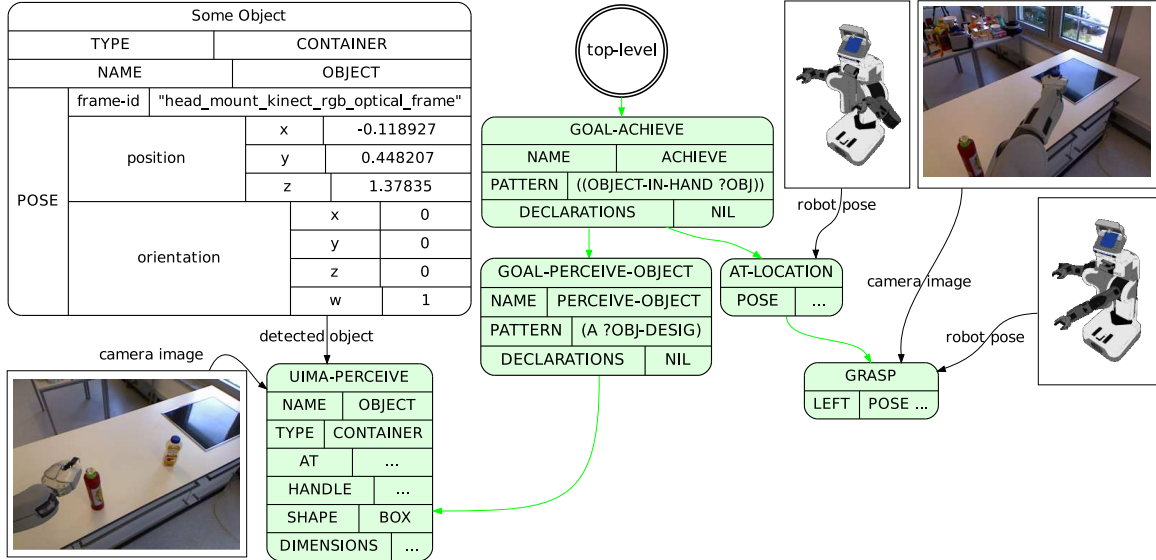


Figure 3: Simplified plan event log for an object-in-hand goal achievement. A perception algorithm is employed to find the correct pose for the object in question, the robot agent navigates towards that pose, and grasps the object.

## 5.2 Logging Sensor and Robot Pose Data

The abstract information from the plan logs is complemented with recorded data from sensors, information about the robot’s position in the environment, its pose, etc. in order to be able to reconstruct the world from the viewpoint of the robot as accurately as possible at a later point in time. This can lead to quite a significant amount of data that needs to be recorded without slowing down the task execution.

Our robots are running the ROS communication middle ware (Quigley et al., 2009) in which sensor data and robot pose information are broadcast on so-called “topics” – an asynchronous communication channel that other components (such as the logger) can listen to. This gives the logger access to virtually all pieces of information that are sent around in the robot’s system. For recording this information, we use a modified version of the *mongodb\_log* software (Niemueller, Lakemeyer, & Srinivasa, 2012) that stores the data in a MongoDB database. While this “NoSQL” database does not support sophisticated SQL queries, it is a fast and scalable storage solution that allows recording robot data with little overhead.

The extensions developed for this logging software include an interface for logging designator communication between different components, as well as methods for limiting the amount of data that is recorded. The former enables exact reconstruction of the high level communication between the plan execution system and for example the perception system (requests, results), storing the designators as nested key-value lists in the MongoDB database. The second kind of extensions is necessary to keep the log databases in a manageable size and consist of different methods. First, sensor data like images are only stored for particular points in time, like the beginning and end of a grasping action. Point clouds are stored as depth images, which contain the same information, but consume much less memory. In addition, the *tf* transformations, which represent positions of objects in the world and especially the position and orientation of every joint of the robot, are only

Meta-Predicates (belief state or ground truth)		Reasoning about events	
<i>holds(occ, T<sub>i</sub>)</i>	Occasions in the real world	<i>loc_change(Obj)</i>	Object changed its location
<i>belief_at(event, T<sub>i</sub>)</i>	Occasions in the belief state	<i>object_perceived(Obj)</i>	Object has been perceived
<i>occurs(event, T<sub>i</sub>)</i>	Events in the belief state	Reasoning about occasions	
Reasoning about the logged task tree		<i>loc(obj, Loc)</i>	Location of an object
<i>task(Task)</i>	Tasks on interpretation stack	<i>object_visible(Obj)</i>	Object is visible to the robot
<i>task_goal(Task, Goal)</i>	Goal of task	<i>object_placed_at(Obj, loc)</i>	Object was placed at location
<i>task_start(task, T)</i>	Start time of task	Reasoning about logged poses and designators	
<i>task_end(Task, T)</i>	End time of task	<i>desig_type(Desig, Type)</i>	Type of designator
<i>task_status(Task, Status)</i>	Status of task (not started, ongoing or finalized)	<i>desig_prop(Desig, Prop, Val)</i>	Property values of designator
<i>subtask(Task, Subtask)</i>	Task is a parent of Subtask	<i>obj_pose_by_desig(Obj, Pose)</i>	Object pose from perceived designator
<i>subtask<sup>+</sup>(Task, Subtask)</i>	Task is an ancestor of Subtask	<i>lookup_transform(Src, Tgt, T, Tf)</i>	Logged transform from Src to Tgt at time T
<i>returned_value(Task, Result)</i>	Result of task (success or fail)	<i>transform_pose(P<sub>i</sub>, Src, Tgt, T, P<sub>o</sub>)</i>	Transform P <sub>i</sub> from frame Src to frame Tgt at time T
<i>failure_task(Error, Class)</i>	Failure of task	<i>visible_in_cam(Obj, Cam, T)</i>	At time T, Obj was in the field of view of Cam
<i>failure_class(Error, Class)</i>	Class of failures	<i>blocked_by_in_cam(Obj, Blk, Cam, T)</i>	At time T, Blk was blocking the view of Cam on Obj
<i>failure_attribute(Err, Name, Val)</i>	Attribute of failure		

Table 1: Predicates for reasoning about the memorized experiences.

logged when the data has changed. These transformations are updated very frequently (at around 30-40 Hz), which is needed for motion control, but not necessarily to reconstruct the approximate motions from the log files. We therefore introduce a threshold and only store transformations which have changed more than this value with respect to the previously logged version. This reduces the resulting *tf* file size from around 200 MB to around 30 MB for a regular pick and place task since only actual movement data is recorded. The thresholds have been chosen as 0.005m euclidean and 0.005rad angular distance. In addition, we log each transformation at least once a second to facilitate the retrieval of the last transformation before a given time point from the database.

## 6. Retrieval of Information from stored Memory Data

The vaguely structured plan event logs recorded in the memory can hold substantial amounts of information about the tasks and the events that happened during their execution. Taking a seemingly simple *pick and place* task as example, questions such as “How long did the pick and place task take?” and “How many tries did the agent need to find a suitable pose to stand at when grasping?” become answerable.

These queries can be formulated using the predicates listed in Table 1. The first set of *meta-predicates* is used to ask for information at a given time and to distinguish between the robot’s uncertain belief and ground truth data about the state of the world. While all information in the memory originates from sensor data, some is much more reliable than others. The proprioceptive sensors measuring the robot’s joint angles and thus producing information about its pose, for example, are very accurate and reliable and are thus considered as ground truth. Visual object recognition and pose estimation, in contrast, is a comparatively brittle and unreliable source of information that only updates the robot’s belief about the world. The other predicates can be used as arguments to the meta-predicates to reason about the logged task tree, recorded events, occasions (similar to situations), designator values and robot poses over time.

The task tree is logged directly to the knowledge base, i.e. the respective predicates can be implemented by normal PROLOG queries. In contrast, the predicates for reasoning about events, occasions, designators and robot pose information are evaluated on the data logged in the MongoDB database. To the user, they span a kind of “virtual knowledge base” that is computed on demand



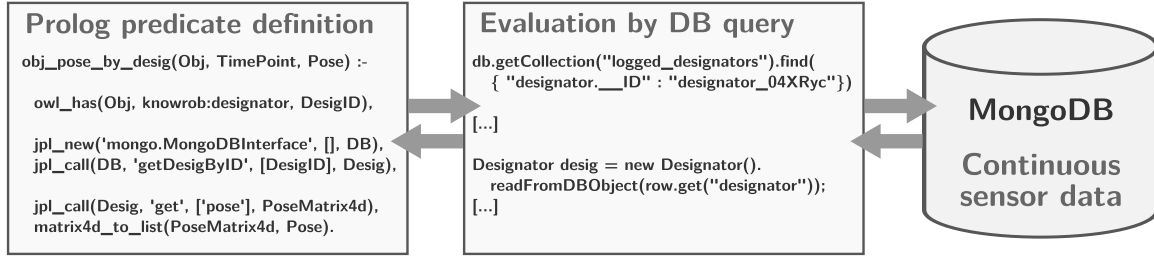


Figure 4: Simplified illustration of the implementation of reasoning predicates that are evaluated based on logged perception data. The set of these predicates spans a “virtual knowledge base” over the recorded memory data.

at query time. Figure 4 explains how the PROLOG predicate `obj_pose_by_desig` is implemented, which computes the pose of an object at a given time based on detections of that object described as designators. The PROLOG implementation reads the designator attached to the object at hand, and calls a Java method using the Java Prolog Interface (JPL) to read its pose information. This Java method translates the call into a query to the database and returns the results to the PROLOG predicate.

## 7. Experiments

The presented techniques are applied to a *pick-and-place* scenario, featuring a PR2 robot that transports an object from one arbitrary position on a counter to another. The experimental setup is as follows. A cylindrical object is placed on a kitchen counter. The robotic agent is equipped with only the information that the object is somewhere on this counter and that it should pick it up and transport it to a random new position on the counter. The top-level plan structure

```

1 (let* ((loc-desig (a location '((on Cupboard) (name kitchen_island))))
2       (obj-desig (an object '((type container) (at ,loc-desig))))
3       (achieve `(loc ,obj-desig ,loc-desig)))

```

supplies information about the object itself, but leaves out situational data. The `(achieve `(loc ,obj-desig ,loc-desig))` call ultimately starts the plan performance, ordering the robotic agent to move the object `obj-desig` from its current location (on the counter) to the new location `loc-desig`, also being on the kitchen counter. The designator `loc-desig` used herein has a two fold use. It generally describes all locations on the counter, without stating an explicit pose. It is applied to the current object location, which is somewhere on the kitchen counter, making all explicit poses on the counter valid search regions for this object. Also, it is used as the target location for putting down the object, which in turn makes all (free) poses on the table valid putdown poses during object placement. At no point, the high level structure `log-desig` is replaced in the high level plan, but rather resolved to actual 6D poses in the lower level modules.

Figure 5 shows images automatically taken during plan execution as part of the plan log. Three situations are depicted – detecting, approaching, and grasping the object in question. Several more situations were encountered in which the agent failed to perceive the object, and had to try out several positions to stand at before being able to grasp or place the object. We elaborate on this scenario using different queries we developed in order to gain knowledge from recorded experimental data. The information acquired this way spans over all kinds of data the robot is recording: plan events, motion and pose data, communication with other components, and images taken. The knowledge

resulting from inquiries like why certain tasks mostly fail or if certain positions to stand at are bad for grasping nearby objects, are key information for enabling cognitive abilities like reflection about how a task is performed.

### 7.1 Recorded Experimental Data

The experimental evaluation for the presented techniques covers the examination of three distinctly different logging subjects. In a low level sense, we record the motion data of the robot. For this purpose, we let it perform a mundane movement sequence over a long period of time that shows the efficiency of low level data logging and storage. Another type of sensor event to record during many experiments is the image stream from the robot’s cameras. To comprehend each respective situation throughout an experimental trial and to be able to post-process imagery from such an experiment, visual evidence from key moments (grasping, navigation, perception) is collected. In order to find a feasible mechanism for this, we conducted a multitude of experiments for table top inspection (i.e. finding all objects on a table) to validate that the data recording mechanism can handle such data streams. The most complex type of data stream to collect is the actual task and parameter description of any high level plan the robot is performing. In order to enable the proposed system to reliably assemble this information, we ran several large pick and place experiments.

The resulting data that is recorded during the execution of a robot plan includes information about performed high level tasks, low level motion, and images taken in key moments. Such key moments are triggered before and after travelling to a new position, before and after grasping an object, and when running perception attempts. The images taken during this consist of single JPG encoded, compressed image files, which take up around 45kb per file. This keeps the amount of drive space used for visual evidence during the plan logging in reasonable ranges. The symbolic reasoning data recorded during a complete pick and place task as examined in this work, covering symbolic high level reasoning data such as a task description, as well as object, action, and location definitions, sums up to about 200–250kb.

The most drive space intense part of logged experiment data is actual low level motion information ( $\tau f$  link transformation data). Figure 6 shows the amount of data recorded for a reference motion. During this motion, the robot moves one arm from one position to another 25 times with different inverse kinematics solutions. This way, differences in inverse kinematic solutions can be neglected and different filter settings for the  $\tau f$  throttling can be compared. The figure shows that throttling greatly decreases the data to store, at the cost of accuracy. During the conducted pick and place tasks, a threshold of  $0.005m$  and  $0.005rad$  was used. These thresholds still allow for qualitative reasoning as noise of perception systems and the actual robot base localization introduce



Figure 5: Camera images taken during execution of a pick and place task.

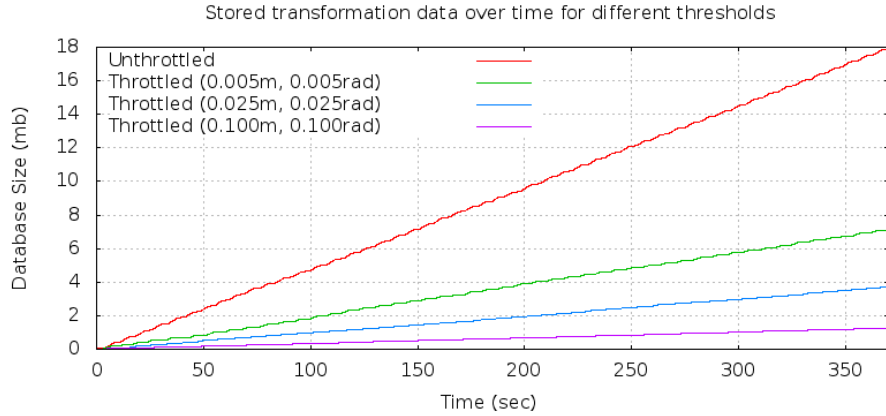


Figure 6: Stored transformation data over time. The different lines represent different throttling thresholds, as shown in the Figure. The experiments were conducted over a timespan of 370s.

similar uncertainties. Therefore, the slightly lower accuracy can be neglected in favor of a smaller storage size.

## 7.2 Queries on the Recorded Data

We assess the performance of the system by the range and diversity of queries it is able to answer based on the memorized information. The following queries are exemplary for different kinds of reasoning problems that occur when reasoning about logged execution data: Durations of tasks, types and probabilities of failures to occur, spatial reasoning to compute relations between objects and between objects and the robot’s pose at different points in time, as well as the use of these inferences for diagnostic purposes. For being able to answer these queries, the system has to combine information from the high level task tree, low level data like the robot’s pose over time, detected objects, and background knowledge like the robot’s self model. While some of these queries could directly be integrated into the robot’s decision making procedures, their main purpose will probably be to retrieve training data and annotations for learning statistical models of the robot’s plans and its performance in different situations.

### 7.2.1 How long does a pick-up task take on average?

The average duration of certain tasks can be important when analyzing time requirements of plans. For example, the following query returns the average time needed for a pick-up action by counting how many tasks with the goal *OBJECT-IN-HAND ?OBJ* have existed and how many seconds each of these tasks took.

```
?- bagof(Dur, (task_goal(Tsk, 'GOAL-PERCEIVE-OBJECT'),
              task_start(Tsk, StT),
              task_end(Tsk, EndT),
              Dur is EndT - StT), Durs),
   sumlist(Durs, Sum),
   length(Durs, Num),
   Avg is Sum / Num.

Durs = [7,7,9,7,9,7],
Sum = 46,
```

```
Num = 6,
Avg = 7.6667
```

### 7.2.2 Tasks failed due to an undetected object

The robot can investigate which tasks have failed due to *ObjectNotFound* failures using the query below which can be answered based on the recorded task tree. CRAM<sub>m</sub> can also return all images captured by the robot in the context of perception tasks that did not detect matching objects (return value `nil`). These images serve programmers as diagnostic material or, in an autonomous learning context, can be used by a robot to test alternative perception methods offline.

```
?- task(Task),
   failure_class(Error, kr:'ObjectNotFound'),
   failure_task(Error, Task).

Task = log:'node_E3dONaOC',
Error = log:'node_E3dONaOC_failure_0'
```

### 7.2.3 How likely are instances of a task class to fail due to a certain failure?

Using the logged memories, robots can compute success probabilities for their tasks. The probability that a task fails due to a given reason can be computed by the number of failed tasks divided by the total number of tasks of this kind. This probability can help to model the expected behavior of the plans and to determine whether refinements are necessary.

```
?- bagof(Err, (task_class(Task, kr:'ResolveActionDesignator'),
              failure_class(Err, kr:'ManipulationPoseUnreachable'),
              failure_task(Err, Task)), Errors),
   length(Errors, NumErr),

   bagof(Task, task_class(Task, kr:'ResolveActionDesignator'), Tasks),
   length(Tasks, NumT),

   Probability_of_failure is NumErr/NumT.

NumErr = 2,
NumT = 36,
Probability_of_failure = 0.0556.
```

### 7.2.4 Which objects did the robot believe to be on the counter top at a certain time?

Remembering which objects were at a certain location in the past can save robots from carrying out additional perception tasks. An example would be to query what objects the robot believed to be on the table before it tried to grasp an object:

```
?- task_goal(T, achieve('(OBJECT-IN-HAND_?OBJ)')),
   task_start(T, S),
   belief_at(loc(O,L), S),
   on_Physical(O, kr:'CounterTop208').
T = log:'CRAMAchieve_4aOJNJBZ',
S = 1378119171,
O = log:'VisualPerception_WbrSG11j_object_0',
L = kr:'RotationMatrix3D_vUXiHJMj'
```

The result of such a query is obtained from which objects were in the belief state at that time instance. Moreover, the last predicate checks whether this object is on top of the counter by check-

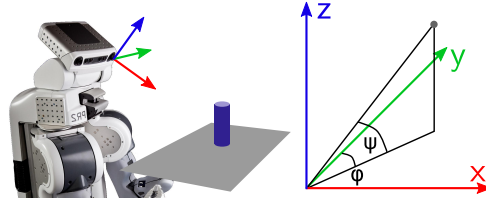


Figure 7: Left: Outside view of the scene with robot camera coordinate frame. Right: In camera-local coordinates, the computation of the bearing towards the objects can be decomposed into two two-dimensional problems.

ing the location of the island with the semantic map of the environment. This query integrates the recorded designators (perception results), the symbolic task tree and prior knowledge from the robot’s environment model.

### 7.2.5 Check if camera was facing the object to be detected

If an object cannot be detected, it may be that the robot did not look at the right location. If a subsequent detection succeeds, we can analyze if this was the problem by computing whether the position of the object was in the robot camera’s field of view before. To compute what the camera was looking at during some point in time, we need to know where the camera is positioned and how large its field of view is.

The former information can be obtained from the recorded robot pose data. In the context of the ROS robot software system, the *tf* library facilitates the management of 3D coordinates by offering methods for transforming any pose into any coordinate frame at a given time. While the original *tf* only keeps data from the past 10 seconds, we have extended the system to operate on the full memory of poses such that it allows arbitrary transformations between all coordinate frames at all times for which data is available. The latter information can be obtained from the robot model in the Semantic Robot Description Language (SRDL, (Kunze, Roehm, & Beetz, 2011)). SRDL describes the geometry of robot parts, their kinematic structure and, for special components like sensors, semantic properties like their resolution or field of view of a camera.

Being able to transform poses into other coordinate frames at arbitrary times makes the problem of computing the camera’s view very simple. Using the logged pose data, we can transform the object pose, which is stored with respect to the robot’s environment map, into the local camera coordinates (Figure 7). Instead of having to solve a three-dimensional problem, we can now decompose the problem into the computation of the bearing towards the object in horizontal and vertical direction and compare the angle to the camera’s field of view.

$$\phi = \text{atan}\left(\frac{y_{obj}}{x_{obj}}\right) < HFOV \quad \psi = \text{atan}\left(\frac{z_{obj}}{x_{obj}}\right) < VFOV$$

This computation is implemented in the *obj\_visible\_in\_camera* predicate that can be used to ask whether an object was visible for some specific camera at a given time (e.g. the beginning of a perception action), or in which cameras it has been visible.

```
?- task_start(log:'CRAMPerceive_uocvmivw', _St),
    owl_individual_of(pr2:pr2_head_mount_kinect_rgb_link, srdl2comp:'Camera'),
    obj_visible_in_camera(log:'VisualPerception_Z9fXhEae_object_0',
```

```

true .                                pr2:pr2_head_mount_kinect_rgb_link , _St).

?- task_start(log:'CRAMPerceive_uocvmivw', _St),
   owl_individual_of(Cam, srdl2comp:'Camera'),
   obj_visible_in_camera(log:'VisualPerception_Z9fXhEae_object_0', Cam, _St).

Cam = pr2:pr2_high_def_frame ;
Cam = pr2:pr2_head_mount_kinect_ir_link ;
Cam = pr2:pr2_head_mount_kinect_rgb_link ;
[...]
false .

```

### 7.2.6 Check if the camera's view of an object was blocked by a robot part

A common problem in object manipulation tasks is that the robot cannot see an object because one of its arms is blocking the view. This problem could be avoided by retracting both arms out of the scene, but this is very inefficient. To analyze if a perception failed because a robot part was in the view, we can again use the logged pose and object position data, but instead of computing if the bearing towards the object is smaller than the camera's field of view, we compute whether the bearings to the object and some robot part are close enough together. This exploits the hierarchical nature of the SRDL model by backtracking over all *sub\_components* of the robot's arm and checking for each of them if they block the view.

```

?- task_start(log:'CRAMPerceive_uocvmivw', _St),
   sub_component(pr2:pr2_right_arm, Part),
   obj_blocked_by_in_camera(log:'VisualPerception_Z9fXhEae_object_0',
                           Part,
                           pr2:pr2_head_mount_kinect_rgb_link, _St).

Part = pr2:pr2_r_wrist_roll_link ;
Part = pr2:pr2_r_forearm_cam_optical_frame ;
Part = pr2:pr2_r_gripper_palm_link ;
[...]

```

This query is an approximation of the object's visibility since it neither takes the volume of the robot's arm nor the object into account. We are working on methods for geometrically reconstructing the recorded scenes so that we can apply more sophisticated techniques like off-screen rendering of the scene (Mösenlechner & Beetz, 2013).

## 8. Related Work

Episodic memories similar to the ones recorded by CRAM<sub>m</sub> have been investigated in the area of cognitive architectures, though often with a focus on modeling human cognitive processes rather than implementing a scalable architecture for robots systems. One of the earlier cognitive architecture that mimics humans' working memory (WM) is Soar (Laird, Newell, & Rosenbloom, 1987). Soar's working memory contains procedural, declarative and episodic knowledge. Namely, it contextualizes *i*) a *context stack*, which specifies active goals, problem spaces, states and operators of the embodied agent; *ii*) *objects*, which are denoted with attributes called *values*; and *iii*) preferences that give the procedural search-control knowledge. ACT-R (Anderson et al., 2004) is another cognitive mechanism that is built upon a memory concept. In contrast to Soar, it has two different memories for declarative and procedural knowledge which contain facts and things respectively. It

was adapted for different cognitive applications such as choosing among the competing associations of a concept (Anderson & Reder, 1999), *list memory paradigm* (Anderson et al., 1998) and creating a memory based on the theory of *serial memory* in psychology (Anderson & Matessa, 1997). ICARUS (Langley & Cummings, 2004), an integrated cognitive architecture for physical agents, has two different memory hierarchies. On the one hand, the conceptual memory contains knowledge about general features of things and their relationships. On the other hand, the skill memory stores knowledge about how to accomplish goals. Each of these hierarchies has a long-term memory (LTM) and a short-term memory (STM).

In the context of robotics, a memory system needs to consider the properties of physical robotic agents, scalability issues due to storage constraints, and processing speeds of the memorized data. Prior work in this field, which is extended in the presented approach, was done by Beetz (2000) using a simulated robot in a simpler environment performing navigational tasks. Our work contributes by extending the domain of application to *mobile manipulation*, which covers much more complex manipulative and perceptive tasks, and applying the principles to an actual, real robot. The mechanisms shown are deeply anchored in the robot’s control system and can handle high volume low level data without disturbing the plan execution. For such low level logging, Niemueller et al. (2012) have presented a comprehensive, dynamic logging system for low level sensor signals into a MongoDB database. We build upon this work and have extended it with methods for logging plan events and designators and have integrated it with our knowledge processing system to allow semantic reasoning about the data. To keep the amount of logged data in a manageable size, we implemented techniques for throttling high-frequency data like the stream of robot pose information before recording. Hilbert and Redmiles (2000) describe the benefits of event logging and event stream transformation into streams of interest by selecting, abstracting, and storing them according to current requirements. They make use of this technique to summarize sequences of actions into tasks and to characterize sequences based on probability matrices. As Coad (1992) points out, such an “*Event Logging Pattern*” consists of a “*device*” triggering an event remembering message which adds a certain sensor event to a database of events with historical values when surpassing a given threshold value. In our case, these messages might be generated when a plan event *starts* and when it *ends*, putting everything in-between into its context. Our assumption is that a certain context is *active* as long as it is not revoked by an active trigger or by the absence of a previously active trigger signal. In the case of plan-logging, a task-context is started at the beginning of its subroutine and ends when the subroutine is left again. Subtasks of this task may show the same behavior, making them hierarchical children.

On the basis of such high level information, Brachman (2002) describes the necessity of systems that can reflect on their current task and their own performance. Benefits gained from more reflective systems would be the ability to take a step back from the current situation and getting out of a mental box, but also to be able to explain why a certain task is being performed in the way it is done by a cognitive agent. Our proposed approach aims at gaining this kind of knowledge from observing the (internal) state of the agent and the state of the surrounding environment, thus being able to reconstruct any situation during the performance, as well as build up the causal connection between events and their consequences. With this information at hand, Kaelbling and Lozano-Pérez (2012) elaborated on the point of having a complete belief state available for replanning and rea-

soning purposes. They harness this information as basis for dynamic decision making. Planning actions based on this, and taking a robot’s possible courses of actions into account, they plan ahead based on the current situation in order to get an impression about the nature of future situations. The process they perform in a live scenario relies on quick processing times and on inexpensive computation mechanisms to not slow down the ongoing task. We build upon this principle by making a complete belief state available offline. This enables our approach to use more computationally expensive algorithms to generate more insights about the respective situation and perform a-posteriori reasoning about the characteristics of the robotic behavior.

## 9. Discussion and Conclusions

In this paper, we presented a comprehensive memory system for cognitive agents acting in the real world. Within this context, we made efforts to clearly distinguish between the agent’s current *belief* about the state of the world and the *actual* state, as well as the robot’s *intentions* that led to this state. When performing an action, the agent expects a certain outcome – this makes up its *believed* world state. Sensor readings, such as camera images, can result in contradictory information, and can yield knowledge about how well a task was performed and even what went wrong when comparing the expected and the actual outcomes. Taking the intentions of the current task into account, the agent can answer questions about *why* it performed a certain task in a certain way, and can store information about possible failures and common pitfalls during this type of action, making improvement of future executions of the same or similar tasks feasible.

The memory of the robotic agent is filled from two streams of events. The first, being of low volume, consists of symbolic event data describing the hierarchical task structure of the executed robot plans. This data also includes symbolic, qualitative parameterizations of tasks described by designators, allowing for logical reasoning. These designators can be extended over time and made more precise when more information becomes available or old information gets retracted and replaced. The second stream holds quantitative data from the robot’s sensors, including camera images and the robot pose and position. Both streams are synchronized using time stamps of the start and end times of plan events. This approach allows to reason about what information was gained through which measures. Perception tasks that have limited *a priori* information about the objects they are looking for yield their requesting and resulting designators. Comparing both may conclude that an object somewhere on a table is at a specific 3D coordinate, and therefore changing vague information into more specific details.

The proposed representation forms the basis for the definition of higher-level concepts like which action *causes* which effects and what the current *beliefs* are. For example, taking an arm movement of the robot into account, this action might be signaled by a plan event *VoluntaryBody-Movement* as it is part of a grasping action. On the basis of this high level concept, low level data about the robot’s pose and the actual reaching motion of the arm can now be connected and reasoned about. Assuming all movements to be connected to the current plan event is sufficient here as the plan triggering the motion takes exclusive control over the arm through a semaphore mechanism. Coming to the agent’s belief, information about real-world entities is available to internal reasoning processes through the designator representation as well. Taking two designators embodying objects



in the real world, their positions might be concluded to be near each other as the designators indicate physical proximity – on a quantitative level, e.g. being near to each other, or on a qualitative level, residing on the same supporting surface.

We presented a comprehensive robot memory system, featuring an extensive encoding scheme for symbolic and subsymbolic experience data collected from real-world robot plan executions. An integrated storage approach for both kinds of data was introduced and logical queries for information retrieval from this memory were developed to make use of the resulting knowledge possible for plan improvement mechanisms. The queries presented up to now picture the conceptual setup of the system, forming the base for more elaborate reasoning mechanisms and query types. Taking more information into account and running exhaustive analysis algorithms on the collected experience data offers much potential when it comes to a-posteriori reasoning and analysis, especially in terms of life long learning concepts. Collecting large amounts of data over many trials can form the basis for substantial improvements during planning and plan execution. Possible results of such learning processes include heatmaps that reflect the utility of robot and object poses for certain tasks, and appropriate failure handling for failed tasks under specific situational context. Experiences collected by robotic agents in different situations can not only hold information about single task types currently performed, but open up possibilities to reason about general knowledge that applies to many situations and tasks. We are developing an open source software framework<sup>1</sup> around the presented techniques that implement plan logging capabilities. For sensor data storage, we rely on the *mongodb\_log* ROS package, which is available as open source, and the reasoning capabilities shown in our example queries for knowledge acquisition are implemented into the KNOWROB knowledge base system, which is being developed as open source as well.

## Acknowledgements

This work is supported in part by the DFG Project BayCogRob within the DFG Priority Programme 1527 for Autonomous Learning and the EU FP7 Projects *RoboHow* (grant number 288533) and *SAPHARI* (grant number 287513).

## References

- Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26, 832–843.
- Anderson, J. E. (1995). *Constraint-directed improvisation for everyday activities*. Doctoral dissertation, University of Manitoba.
- Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *PSYCHOLOGICAL REVIEW*, 111, 1036–1060.
- Anderson, J. R., Bothell, D., Lebiere, C., & Matessa, M. (1998). An integrated theory of list memory. *Journal of Memory and Language*, 38, 341–380.
- Anderson, J. R., & Matessa, M. (1997). A production system theory of serial memory. *Psychological Review*, 104, 728–748.

---

1. <http://www.github.com/code-iai/planlogging>

- Anderson, J. R., & Reder, L. M. (1999). The fan effect: new results and new theories. *Journal of Experimental Psychology: General*, 128, 186–197.
- Beetz, M. (2000). *Concurrent reactive plans: Anticipating and forestalling execution failures*, Vol. LNAI 1772 of *Lecture Notes in Artificial Intelligence*. Springer Publishers.
- Beetz, M., Mösenlechner, L., & Tenorth, M. (2010). CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1012–1017). Taipei, Taiwan.
- Brachman, R. (2002). Systems that know what they’re doing. *IEEE Intelligent Systems*, 67–71.
- Coad, P. (1992). Object-oriented patterns. *Commun. ACM*, 35, 152–159.
- Hilbert, D. M., & Redmiles, D. F. (2000). Extracting usability information from user interface events. *ACM Comput. Surv.*, 32, 384–421.
- Kaelbling, L. P., & Lozano-Pérez, T. (2012). Integrated task and motion planning in belief space. *Submitted. Draft at <http://people.csail.mit.edu/lpk/papers/HPNBelDraft.pdf>.*
- Kunze, L., Roehm, T., & Beetz, M. (2011). Towards semantic robot description languages. *IEEE International Conference on Robotics and Automation (ICRA)* (pp. 5589–5595). Shanghai, China.
- Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: an architecture for general intelligence. *Artif. Intell.*, 33, 1–64.
- Langley, P., & Cummings, K. (2004). Hierarchical skills and cognitive architectures. *Proceedings of the Twenty-Sixth Annual Conference of the Cognitive Science Society* (pp. 779–784 (pp. 779–784)).
- McDermott, D. (1993). *A reactive plan language* (Technical Report). Yale University, Computer Science Dept.
- Mösenlechner, L., & Beetz, M. (2013). Fast temporal projection using accurate physics-based geometric reasoning. *IEEE International Conference on Robotics and Automation (ICRA)*. Karlsruhe, Germany.
- Niemueller, T., Lakemeyer, G., & Srinivasa, S. S. (2012). A Generic Robot Database and its Application in Fault Analysis and Performance Evaluation. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 2012*. Vilamoura, Algarve, Portugal: IEEE/RAS.
- Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (2009). ROS: an open-source Robot Operating System. *In IEEE International Conference on Robotics and Automation (ICRA)*. Kobe, Japan.
- Tenorth, M., & Beetz, M. (2013). KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. Part 1: The KnowRob System. *International Journal of Robotics Research (IJRR)*, 32, 566–590.
- W3C (2009). *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax*. World Wide Web Consortium. <http://www.w3.org/TR/2009/REC-owl2-syntax-20091027>.
- Wood, R., Baxter, P., & Belpaeme, T. (2012). A review of long-term memory in natural and synthetic systems. *Adaptive Behavior*, 20, 81–103.