FP6-004381-MACS

MACS

Multi-sensory Autonomous Cognitive Systems Interacting with Dynamic Environments for Perceiving and Using Affordances

Instrument: Specifically Targeted Research Project (STReP)

Thematic Priority: 2.3.2.4 Cognitive Systems

D2.2.2 Delevopment of an Affordance-based Control Architecture

Due date of deliverable: June 30, 2006
Actual submission date: July 13, 2006

Start date of project: September 1, 2004 Duration: 36 months

Fraunhofer Institute für Intelligene Analyse- und Informationssysteme (FhG/AIS)

Revision: Version 1

Project co-funded by the European Commission within the Sixth Framework Programme (2002–2006)		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

EU Project MACS

Deliverable D2.2.2

Development of an Affordance-based Control Architecture

Erich Rome, Erol Şahin, Ralph Breithaupt, Jörg Irran, Florian Kintzler, Lucas Paletta, Maya Çakmak, Emre Uğur, Göktürk Üçoluk, Mehmet R. Doğar, Piotr Rudol, Gerald Fritz, Georg Dorffner, Patrick Doherty, Mariusz Wzorek, Hartmut Surmann, Christopher Lörken

Number: **MACS/2/2.2**
WP: *2.2*
Status: *draft, version 1*
Created at: *June 22, 2006*
Revised at:
Internal rev: *v14 – July 13, 2006*

FhG/AIS	Fraunhofer Institut für Intelligente Analyse- und Informationssysteme, Sankt Augustin, D
JR_DIB	Joanneum Research Graz, A
LiU-IDA	Linköpings Universitet, Linköping, S
METU-KOVAN	Middle East Technical University, Ankara, T
OFAI	Österreichische Studiengesellschaft für Kybernetik, Vienna, A

This research was partly funded by the European Commission's 6th Framework Programme IST Project MACS under contract/grant number FP6-004381. The Commission's support is gratefully acknowledged.

Corresponding author's address:

Dr.-Ing. Erich Rome
Fraunhofer Institut für Intelligente
Analyse- und Informationssysteme
Schloß Birlinghoven
D-53754 Sankt Augustin, Germany

Fraunhofer IAIS Institut Intelligente Analyse- und Informationssysteme

Fraunhofer Institut für Intelligente
Analyse- und Informationssysteme
Schloss Birlinghoven
D-53754 Sankt Augustin
Germany

Tel.: +49 (0) 2241 14-2683
(Co-ordinator)

Contact:
Dr.-Ing. Erich Rome

JOANNEUM RESEARCH

Joanneum Research
Institute of Digital Image Processing
Computational Perception (CAPE)
Steyrergasse 9
A-8010 Graz
Austria

Tel.: +43 (0) 316 876-1769

Contact:
Dr. Lucas Paletta

LINKÖPINGS UNIVERSITET

Linköpings Universitet
Dept. of Computer and Info. Science
Linköping 581 83
Sweden

Tel.: +46 13 24 26 28

Contact:
Prof. Dr. Patrick Doherty

Middle East Technical University
Dept. of Computer Engineering
Inonu Bulvari
TR-06531 Ankara
Turkey

Tel.: +90 312 210 5539

Contact:
Prof. Dr. Erol Şahin

Ö F A I

Österreichische Studiengesellschaft
für Kybernetik (ÖSGK)
Freyung 6
A-1010 Vienna
Austria

Tel.: +43 1 5336112 0

Contact:
Prof. Dr. Georg Dorffner

Contents

1 Introduction

This document describes the first complete version of a robot control architecture that is designed to explicitly deal with *agent affordances* (cf. Sec. 3). The description encompasses the functions of the architecture and some new details that are related to the implementation. Wherever possible and meaningful, we refer to existing documents on implementation details and specifications. A list of these documents is provided in the next section 2.

This document is structured as follows. First, we provide the basic definitions that are required to understand the document. Many of these definitions are taken from other MACS documents, some of them are new. As an appendix, a glossary of terms is also provided, since it turned out to be essential that all readers of this document, be it MACS staff, reviewers, or external readers, get a clear and common understanding of the used terms. The definitions of the terms are taken partly from other MACS documents or from other sources, many are new.

Architectural components and control and data flow between them are illustrated by an appropriate diagram. The main section then specifies the architectural components and their mutual relations. Each component's description comprises a general, functional description, descriptions of the interfaces to other components, and, wherever possible, more or less detailed descriptions of the used data structures. Since the implementation is proceeding in parallel, some of the data structures are not yet defined, and some are already defined in great level of detail.

The document is concluded by an outlook on the next implementation activities, summarizes some open questions and lists some suggestions for future research. In the appendix, more detailed information on some of the architectural components are provided.

2 Related documents

D1.1.2 "Specification of Module Interfaces" [1]
D2.1.1 "Identification of architectural requirements of an affordance-based control" [2]
D3.1.2 "Affordance recognition from visual cues" [3]
D4.2.1+4.3.1: "Tentative Proposal for a Formal Theory of Affordances
Tentative Proposal for an Affordance Support Architecture
Prototype: Affordance-Based Motion Planner" [4]
D5.3.1 "Robotic learning architecture that can be taught by manually putting the robot through action sequences" [5]
D6.1.1 "Specification of final demonstrator" [6]
D6.4.1 "Report on experiment design" [7]
Technical report: "Specification of a Prototype Behavior System" [8]

3 Definitions

In order to understand the concepts presented in this document, it is important to know that these concepts are used in two different contexts. First, we need to be able to describe the robot's situation and its environment from an oberver's point of view. For instance, when we describe the demonstrator scenario and the experiments performed with the robot inside that scenario. Second, we need to be able to describe the implementation of the affordance-inspired robot control architecture. There, we talk about representations, control software routines etc. that are not directly related to the phenomena in the real world.

It is crucial to know that we distinguish between an *agent affordance* and its representation. Examples for the differences between the two are given within the next two definition subsections.

3.1 Definitions related to the robot's environment

`Begin of quotations from draft deliverable` "D4.2.1+4.3.1" [4].

Definition 1 (Entity)**.** *An entity is any thing, object or individual with cohesive structure.*

One can distinguish between physical entities or mental entities. These will be called external and internal entities, respectively.

An entity consists of aspects.

Our robot's internal entities are called *affordance representations.*

Definition 2 (Aspect)**.** *An aspect is a property or characteristic of an entity, or a relation an entity shares with other entities.*

Events occur in the environment and they most often represent encapsulated change relative to some part of the environment. There is a similarity with entity trajectories, but an ontological distinction should be made between entity trajectories and events:

Definition 3 (Event)**.** *An event is a spatial/temporal configuration of aspects and dependencies between them. Events have spatial and temporal boundaries. (Definition modified from D4.2.1+4.3.1).*

Definition 4 ((Agent) Affordance)**.** *An affordance is a relation between an agent and its environment which affords a capability. The agent/environment relation affords a capability if the agent*

1. *has the capacity to recognize that it is in such a relation between itself and its environment, and it*
2. *has the ability to act to bring about that capability.*

`End of quotations from draft deliverable` "D4.2.1+4.3.1" [4].

The agent affordance definition is used whenever we are referring to or describing the robot's situation in its environment, e.g. in examples of the robot's behavior or in descriptions of experiments. For this purpose, we use the notions of entity, (observed) behavior, (observed) outcome. Example: "The robot has successfully lifted the blue can.", where the blue can is the entity, lifting is the observed behavior, and the successful execution resulting in the can attached to the robot's electromagnet is the observed outcome.

3.2 Definitions related to robot internals

`Begin of quotations from draft deliverable` "D4.2.1+4.3.1" [4].

Definition 5 (Attribute)**.** *An attribute is a data type representing an aspect of an entity. Attributes have types.*

Definition 6 (Values)**.** *A value is a quantity or quality of an attribute. The type of the value set a value belongs to is the value's type.*

An uncertainty measure can be associated with any attribute/value pair. The type of uncertainty measure used may be probabilistic, fuzzy, rough, etc.
In the robot system, entity frames will be used to represent cohesive structures:

Definition 7 (Entity Frame)**.** *An entity frame is a data structure representing an entity. It consists of a name and a set of attribute/value pairs.*

Definition 8 (Entity Trajectory)**.** *An entity trajectory is a data structure representing a set of aspects of an entity through time. It consists of a name, a subset of attribute/value pairs associated with the entity definition, a temporal interval and a sequence or continuum of entity (sub)frames ordered relative to a suitable temporal structure.*

An entity trajectory represents the temporal/spatial progression of an entitie's aspects during a temporal interval. It can be viewed as a bundle of signals or time series.

Definition 9 (Aspect Trajectory)**.** *An aspect trajectory is an entity trajectory with a single attribute.*

An aspect trajectory represents the temporal progression of an single aspect of an entity. It can be viewed as a signal or time series.
`End of quotations from draft deliverable` "D4.2.1+4.3.1" [4].

Definition 10 (Event Type Structure)**.** *An event type structure is a data structure describing an event type and includes:*

1. *An event name parameter ename.*
2. *A set of attribute variables, $\mathcal{F}$, associated with the event type.*
3. *A set of entity variables, $\mathcal{E}$, associated with the event type.*
4. *A temporal index $\bar{t}$ parameter associated with the event type.*[1]
5. *A spatial index $\bar{s}$ parameter associated with the event type.*[2]
6. *A set of temporally ($\bar{t}' \in \bar{t}$) and (possibly) spatially ($\bar{s}' \in \bar{s}$) parameterized logical formulas representing constraints and dependencies between feature values of the feature variables in $\mathcal{F}$. This set of constraints and dependencies is denoted by $\mathcal{D}$.*

A simple temporal network (STN) is a data structure that will be used as an internal representation for event types expressible in this form.

[1] If required, event start and termination temporal indexes can be derived form $\bar{t}$.
[2] It should be possible to derive the spatial and temporal indexes, $\bar{s}, \bar{t}$ from $\mathcal{D}$.

Definition 11 (Entity Structure)**.** *An entity structure is a data structure that is either a recursive entity frame, that is, an entity frame that contains at least one attribute value pair where the value part is a pointer to another entity structure, or a plain entity frame where no values point to other entity structures.*

Definition 12 (Affordance Representation)**.** *An affordance representation or affordance triple is a data structure:*

$$(\textit{outcome descriptor}, \textit{cue descriptor}, \textit{behavior descriptor}).$$

where

- *outcome descriptor is an entity trajectory, containing pairs of attributes and associated value ranges, as well as their spatial and temporal development,*
- *cue descriptor is an entity trajectory, containing pairs of attributes and associated value ranges, as well as their spatial and temporal development.*
- *behavior descriptor is a reference to a robot behavior—reactive or high-level—, plus an optional set of behavior parameters.*

The cue descriptor contains sensory data—filtered or raw—that support the existence of an affordance. The outcome descriptor contains sensory data—filtered or raw—that describe the outcome as *perceived by the robot* of a prior application of a single or complex behavior. On repeated execution of a behavior, the outcome descriptor can be used to *verify* (→affordance hypothesis verification) the "success" of the execution, that is, whether the actual result matches the expected result.

When we refer to implementation of affordance-based control, that is, when we are talking about the control architecture, we use the notion of affordance representations. Then we talk about sensed or perceived cues, executable actions or behaviors, and the sensed outcome of the robot's actions. For the above example, the sensed cue could be a flat rectangular "top" region, the executable behavior would be the high-level behavior routine "lifting", and the sensed outcome could be the coinciding vertical movement of the flat rectangular "top" region and changes in the sensor values of crane arm sensor monitoring.

4 Overall Architectural Considerations

The architecture described here is based on earlier considerations and the requirements of an affordance-based control architecture as specified in the deliverable D2.1.1 "Identification of architectural requirements of an affordance-based control" [2]. In this section, we will describe approaches for some of the central subtopics. We will explicitly point out deviations from the original requirements, where applicable.

4.1 Introduction

We start this introduction by quoting a part of the latest Executive Summary of the MACS *Periodic Activity Report Year 1* ([9, p. 5–9], available at the MACS web site [10]). It contains some of Gibson's most quoted characterizations of the affordance concept, and our interpretation of these statements.

> Experimental psychologist James J. Gibson coined the notion of affordances in his *Ecological Approach to Visual Perception*:
>
> > "The affordances of the environment are what it offers the animal, what it provides or furnishes, either for good or ill. The verb to afford is found in the dictionary, but the noun affordance is not. I have made it up. I mean by it something that refers both to the environment and the animal in a way that no existing term does. It implies the complementarity of the animal and the environment." [11, p. 127].
>
> The animal (or agent), on the other hand, must possess the capabilities to perceive and act upon the affordances. An affordance in this sense is a very specific relation between an agent and an environment that provides a capability or potential for action. The potential is direct in the sense that perception is direct. Perceptual entities are not semantic abstractions but opportunities for action mediated by a specific relation between the agent and its environment. This implies a very tight loop between actions (or behaviors) and opportunities the environment affords (perceptual entities). The behaviors and perceptual entities are symbiotic.
>
> Another important aspect of Gibson's theory of affordances is that
>
> > "... to perceive an affordance is not to classify an object." [11, p. 134].
>
> Gibson goes on to state that
>
> > "... If you know what can be done with a graspable object, what it can be used for, you can call it whatever you please. ... The theory of affordances rescues us from the philosophical muddle of assuming fixed classes of objects, each defined by its common features and then given a name. ... But this does not mean you cannot learn how to use things and perceive their uses. You do not have to classify and label things in order to perceive what they afford." [11, p. 134].

> Thus, objects and affordances are complementary in the sense that one object class may offer a multitude of affordances, and one affordance may be offered by a multitude of object classes.

An example for the latter statement is the following. A beverage can affords to be picked up, to be opened, to be emptied, and to be thrown away, to name just a few. But each of these affordance is also offered by a multitude of other things that humans experience in their everyday environments. Affordances encompass a function-centered view on the environment.

How can one exploit this for robot control? First, we can state that a function-centered perception approach will realize a view of the environment that is orthogonal to object-centered perception. Such function-centered perception would potentially allow a robot to find more alternatives for acting in its environment. A robot mission that requires to find—based on appearance only—and use certain objects in the environment will fail if one or more of these objects cannot be found. But often the identity or appearance of an object may not be relevant for completing a task. A task could, for instance, also be completed if the robot finds an alternative object that offers the same functions as the original one. An affordance-inspired robot control with a function-centered perception would allow a robot more flexibility in plan execution and thus increase the likelihood of successfully completing a mission. Thus, it would enhance a robot's abilities to perceive and utilize the potential for action that the environment offers, i.e. enable a robot to make use of affordances. This is the central hypothesis of MACS.

What is new about this approach? Only very few robot perception approaches deal with recognition of functions that the environment offers. The vast majority of robot perception approaches are either close perception-action couplings for reactive behavior or oriented towards object recognition on higher control levels. Also, object recognition is in many cases based on general computer vision methods that do not account for the specifics of the robot at hand, i.e. its sensory system and its actuator system.

MACS aims at realizing affordance-inspired control in a hybrid architecture that allows goal-directed behavior based on function-centered perception. Affordance support in the sense sketched in the previous paragraph will be built into several levels of the architecture. In order to use affordance support for deliberate action, i.e. for planning, we will need an explicit representation of the potential for action or the functions that the enviroment offers, respectively. The formalization that is the basis for such representations is described in [4]. The authors are not aware of other robot control methods that make use of explicit, symbolic affordance representations.

How can the power of this approach be demonstrated? This was a central question in the beginning of the MACS project. Our hypothesis is that the power of this approach can best be demonstrated in experiments that involve manipulation tasks. Although preliminary experiments ([4]) showed that the approach can also be used for navigation tasks, we think that the results of manipulation experiments will stand out better.

We have described a demonstrator scenario and sketched a number of experiments that are suited to demonstrate the novelty of the approach [6]. For our robot at hand, a six-wheeled mobile robot with a simple crane arm manipulator, we are currently elaborating one of the sketched experiment families. We plan to conduct a number of experiments in

stack building, where we use a variety of test objects, from specifically made objects to everyday objects, that the robot shall use for building stacks. The robot shall learn the functions of stacking bases, middle stack elements and top stack elements, and, optionally, learn stability cues. The stack height should only be limited by the maximum height of the crane arm magnet above ground and the number and heights of stackable test objects. The robot will experiment with many of these test objects and learn cues for the presence of certain functions or affordances. Cues in this sense may be invariants across a wide range of test objects with different appearances. The robot shall learn how to use these test objects for the stacking task. The final challenge will be the use of new test objects that offer the same functions but have different appearances than the test objects that have been employed in the initial learning phase.

What are the research questions? The main research questions are listed below. In this introduction, we have already addressed many of them implicitly or explicitly and have supplied a few hypotheses and expectations. Since MACS is still ongoing research, we have to await results of experimental validation to provide substantiated answers to most of these questions.

- What is Gibson's ontological stance and is it consistent or inconsistent with the idea of affordances, not only as proposed by Gibson himself, but also in the context of computationally based artifacts such as robots?
- Is a functional ontology for a robotic's system a generalization of Gibson's ideas or is it something completely different?
- Should behavior-based robotics frameworks be viewed as an implementation of the idea of affordances? If so, is this the closest we can get in terms of abiding by Gibson's views on affordances? If not, what are the differences?
- How would affordance-based control go together with behavior-based and plan-based control? Is it complementary? Redundant? Inconsistent?
- What would be the benefit of adding affordance support to robot control?
- Traditional knowledge representation is inconsistent with Gibson's idea of affordances yet appears to be a pre-requisite for the implementation of higher cognitive processes in a robotic system. Does this observation rule out any generalization or integration of affordances with plan-based robotics systems?

  If not:

  - What about an affordance needs to be represented in a robot, and how?
  - Should (aspects of) affordances in a robot be programmed or learned? (Can they be programmed in the first place?)
  - How can (aspects of) affordances be used for reasoning and action?
  - How and where in the architecture would attention, intention, or other internal states filter (aspects of) affordances that were perceived on a low level?

What are the expected results? The main result will be a working, integrated robot system, based on the KURT3D robot, that serves as a proof of concept for the affordance-inspired robot control approach. Other results of the project will be a formal theory, a representational formalism for affordance support, a dedicated simulation environment, a specifically taylored learning approach for generating affordance representations, feature extractors and other software for function-centered perception, plus dissemination of the results.

4.2 Architectural building blocks

The main architectural building blocks are:

User Interface displays status information and allows a user both to guide a robot manually through an action sequence and to just specify a mission goal for the robot. The

Deliberation module converts a mission goal into an executable affordance-based mission plan which is passed to the

Execution module. This module executes the mission plan, monitors its execution, including successful or unsuccessful acting upon affordances. The Execution module's new *Event and Execution monitor* checks the existence of affordance support cues and compares expected outcome with actual outcome of an executed behavior control routine. The Execution control triggers behaviors of the

Behavior System. This module provides a number of pre-programmed behavior control routines that can be viewed as basic skills of the robot. Some behavior control routines are parametrizable and can be configured by other modules, if necessary. The behaviors make use of

Actuators that enable the robot to move about and to interact with its environment. They include the drive motors, the sensor servos, and the crane arm motors. The

Sensors enable the robot to perceive its environment and its internal states, Virtual sensors provide software state information, real sensors yield extero- and proprioceptive data. All sensory data are first handled by the

Perception module. It relays sensory data, extracted features and status information (like active behaviors and their parameters) to the Learning module, Execution module, Behavior System and Deliberation module. It can be configured to look just for certain features that relate to searched affordance support cues. Its *Entity Structure Generation Module* converts sensory data into appropriate data structures for architectural affordance support. The

Learning module takes input from the Perception module and generates affordance representations (affordance triples) to populate the new

Affordance Representation Repository. This repository is new and specific to our affordance-based approach. It provides affordance representations for use with the affordance-based Planner for goal-oriented, affordance-based mission planning.

4.3 Acquisition of affordance representations

In principle, we consider four ways in which a robot can acquire affordance representations:

1. through learning by self experience, that is, by experimentation in its environment,
2. through manual guidance (tele-operation) through action sequences,
3. through adding hand-coded affordance representations to the Affordance Representation Repository, and
4. through learning by imitation.

Ad 1) Learning by self experience will be the method that is in the focus of our investigations. The robot is equipped with a number of basic skills, that is, pre-programmed behaviors that allow it to explore its environment and perform some actions. The outcome of an action sequence is monitored, and a three phase learning process shall generate appropriate affordance representations.

Ad 2) Manual guidance is an aid in situations where the robot's basic skills are insufficient. The learning procedure is the same as in case 1), but here the robot is no longer acting autonomously, but is tele-operated by a human operator.

Ad 3) A means to accelerate the robot's bootstrapping might also be to equip the Affordance Representation Repository with an initial set of hand-coded basic affordance representations.

Ad 4) Learning by imitation of other robot's behaviors is the most demanding possibility and will be left as a future research topic. Successful implementation of learning by experimentation is a prerequisite for learning by imitation, and therefore the latter method has been chosen as the research focus for MACS, as far as learning is concerned.

Learning by experimentation In order to be able to learn the affordance representations there must be sufficient data concerning the behaviors. Thus during the learning process there could be the need to acquire more data for a special behavior to ensure that there is a sufficient amount of data for learning. To be able to request such data there must be the possibility to request the execution of a specific behavior. This is done by sending a (control flow) request from Learning to Deliberation. This could be interpreted as a "desire" to learn more about a certain behavior.

5 Architecture diagram

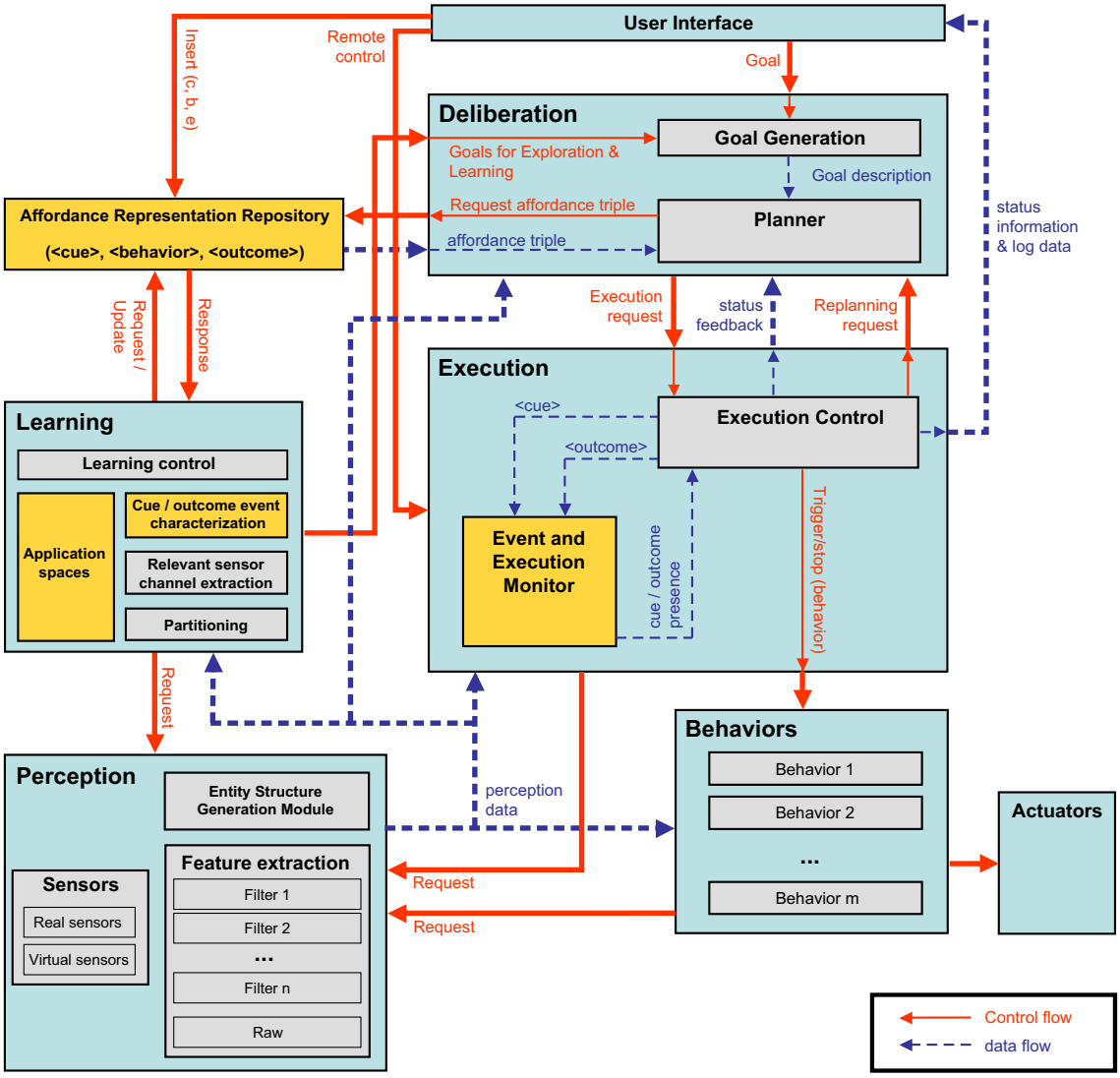

Figure 1: Modules, data and control flow of the MACS control architecture.

A red, solid arrow between components A and B in the diagram is of type control flow. The arrow indicates that the control is passed from A to B. The arrow does say nothing about the situations in which the control is passed, nor about the data that might be exchanged when passing control. The designations close to such an arrow indicate qualitatively the nature of the control flow, e.g. information request, configuration request etc.

A blue, dashed arrow between components A and B in the diagram is of type data flow. The data flow arrow does not say anything about the circumstances, that is, the current control states, under which the data are transferred. The designations close to such an arrow indicate qualitatively the types of data that are passed from A to B.

Bold arrows indicate flows between modules, thin arrows intra module flows. Data passed from module A to B are available to all components inside B. Orange colored boxes are specific affordance support oriented components that are usually not found in other control architectures.

6 Components

This section describes the functions of the main architecture building blocks, that is, the architectural components and their mutual relations.

6.1 Robot computing hardware

6.1.1 General description

The KURT3D robot is equipped with a minimum of two processors: A C167 microcontroller and an on-board notebook computer. The on-board notebook computer runs all the high level control software, e.g. Perception, Learning, Planning etc. The C167 microcontroller runs low level software (firmware) that controls the wheel drives and reads out attached sensor devices. It receives information and requests from the on-board notebook computer, and sends messages and sensor readings to the on-board notebook computer.

6.1.2 Interfaces

The C167 microcontroller communicates via CAN bus with the on-board notebook computer. The on-board notebook computer can communicate via radio transmission (WLAN) with a monitor station operated by a human.

6.1.3 Data structures

Details about the data structures used by the C167 can be found in the KURT3D firmware documentation [12] and, partly, in the appendix of this document.

6.2 Perception module

6.2.1 General description

This module should generate a perceptual view of the environment, using the raw data incoming from extero- and proprioceptive sensors. ESGM (Entity Structure Generation Module) packs the filter outputs into entity structures (be it entity or entity trajectory) to be used by the Learning module and the Event and Execution Monitor Module. The module should include a perceptual filter library and the ESGM which can be configured to detect and track only the relevant features needed for a task. Note that the output of this module is used both to detect the cues needed to trigger a behavior or to monitor the outcome of a behavior execution (through checking against `<cue>` and `<outcome>`).
Active Perception: When a `<cue>` is needed, the Execution Controller initiates an Active Perception behavior. The Behavior system executes this behavior, which may include using actuators to pan or tilt sensors, and sending perception requests to the Perception Module. The Event and Execution Monitor monitors the perception results and signals

a found `<cue>` to the Execution Control, which then terminates the Active Perception behavior.

6.2.2 Module interfaces

The *Learning Module* is able to configure the perception of the robot in order to get sufficient data out of the sensor stream. The configuration includes (i) the type of sensor information (e.g. images from the cameras, depth information from the laser scanner, etc.) and (ii) different levels of abstraction in the data provided by the sensors (e.g. from raw sensor data, via region classification up to object recognition results). The execution of particular behaviors like obstacle avoidance requires specific sensor information. In order to fulfill such tasks the *Behavior Module* requests for those data that are provided by the *Perception Module*. The *Execution Module* is able to configure the perception of the robot, because the observation of specific cues or outcomes is necessary to detect and recognize affordances present in the environment of the robot. The data flow provided by the *Perception Module* consists of structured entities as described in Section 6.4.

6.2.3 Data structures

For a more detailed discussion on how extero- and proprioceptive sensor information is structured see Section 6.4.

6.2.4 Feature extraction

6.2.4.1 General description Part of the Perception module. Uses filters and other routines to preprocess sensor input. Results of feature extraction are provided as a sensor/feature stream. For the purpose of uniform handling, raw sensory data are routed through the feature extraction component, but without application of dimensionality reducing filters (identity filter).

6.2.4.2 Interfaces The *Feature extraction* receives the data structure as provided by the *Sensors* (see sec. 6.3). The interface to the ESGM (6.4) is encapsulated within the *Perception Module* and needs no detailed specification for the overall architecture.

6.2.4.3 Data structures The ESGM performes a wrapping from data provided by the feature extraction mechanism into the entity structure format. The outcome of various filters at different levels of abstraction are represented as attribute/value pairs in order to simplify the dataflow between these two components of the *Perception Module*.

6.3 Sensors

6.3.1 General description

The robot's hardware sensors provide raw sensor readings, both extero- and proprioceptive. The robot accumulates the sensor data that can be processed by a robot's perception software, i.e. the *perception module* in general and specifically its *feature extraction part*. Furthermore, sensory data are used by the robot to realize several *reactive behaviors*. In

the following, we distinguish between internal, external and virtual sensors whereas the transition between these groups is sometimes smooth and depends on application specifics.

6.3.1.1 External sensors External sensors of the KURT3D platform are:

3D Laser scanner The KURT3D robot uses the indoor version of the SICK 2D Laser range scanner LMS 200. The scanner emits laser beams that are distributed by a permanently rotating mirror, yielding a planar 2D scan of the environment. Most of the Laser beams that hit surfaces in the environment are reflected back to the scanner. The device measures the time of flight of the returned laser pulses to determine the distances from the scanner to the surfaces that reflected the beams.

An additional motor/servo device (cf. Sec. 6.12) rotates this scanner around a horizontal axis (parallel to the robot's wheel axes). This "nodding" motion enables the scanner to perform 3D measurements of the environment.

The collected data can then be used to construct digital, three dimensional models that are used in a wide variety of applications. In MACS, the applications at hand are map building and navigation.

Digital Cameras Two webcams with a resolution of 640x480 pixels are mounted on pan and tilt units. They provide the perception module with digital image data at typical frame rates. The image data are tranferred via a USB interface to the robot's on-board notebook computer. The pan and tilt servos (cf. Sec. 6.12 need to be controlled for →*active perception*, e.g. for object finding and tracking.

6.3.1.2 Internal (proprioceptive) sensors The following sensors are employed by the robot KURT3D as proprioceptive sensors.

Wheel/Motor Encoders An encoder is a sensor for converting rotary motion of the robot motor or its wheels to a digital code of electronic pulses. The number of pulses per turn of a wheel, the wheel diameter and distance between to wheels can be used to calculated a rough robot pose. Since the linear pose estimation is better than orientation estimation (friction, skidding) the encoders are combined with gyroscopes whereas the wheel encoders are used to calculated the translation and the gyro to calculate the rotation.

Electrical current sensor The electrical current sensor measures the current state or the used power of the battery. It has to be used by the behavior system and/or planning component to determine the next battery charging cycle.

Kurt2-IMU The optional Kurt2 Inertial Measurement Unit is described in the appendix.

Crane arm sensors The crane arm's operation can be monitored by the following internal sensors:

- encoders to measure the speed of all the three motors
- potentiometer for the current direction of the boom
- limit switches for the horizontal and vertical movement
- weight sensor for the load currently lifted

6.3.1.3 Virtual sensors Virtual sensors are software components that provide (streams of) additional data. One example are data that the robot has computed internally, like the indices of the currently active behaviors and their current parameters.

6.3.2 Hardware interfaces

The different sensors of the robot are one hand directly connected to the on-board notebook computer or on the other hand to the microcontroller C167 board. The microcontroller board C167 is directly connected with the on-board notebook computer via an USB to CAN adapter.

Digital camera The cameras are connected via USB 1.1 to the on-board notebook computer.

3D Laser scanner The interface for the 3D scanner can be used at different levels. On a high level a 3D scan is acquired and delivered to the external caller as a 3D point cloud. On other levels 2D scans with timestamps of the laser data, the servo/motor and the gyro positions are delivered and can be used for navigation, obstacle avoidance or local planning.

Wheel/Motor encoders The encoders are directly connected to the C167 microcontroller board.

Electrical current sensor The electrical current sensor is also directly connected to the C167 microcontroller board.

Servos The servos are connected to the servo device S10. The servo device itself is connected via serial RS232 or via a USB to RS232 device to the on-board notebook computer.

6.3.3 Software interfaces

Real sensors The sensory data received by the on-board notebook computer are passed to the Perception module. Inside the Perception module, the sensory data may be filtered according to the module's current settings. Raw sensory data are provided without application of (dimensionality reducing) feature extractors. In both cases, the ESGM is used as an internal service for wrapping the data into entity structure format and providing them in specified sampling rates to other modules.

Virtual sensors The virtual sensors have only software interfaces. Their data are processed like any other sensor's data, see previous paragraph. All internal status information from other modules that are requested from the Learning module or other modules are provided via virtual sensors. This includes but is not limited to:

- Active behaviors and their parameters
- Information about termination of behaviors (normal, exceptions)

6.3.4 Data structures

The different sensors of the robot are directly connected to the on-board notebook computer or to the microcontroller C167 board. If they are connected to the microcontroller board the sensor readings or simple data processed values of the sensor readings can read via CAN messages to the on-board notebook computer. The data structure of the CAN messages is described in detail in the appendix Sec. B.2.1 for the actuators. The data structure of the CAN messages for the sensors is similar to the messages for the actuators.

Digital camera The data of the digital camera can be read on different levels. At the lowest level the picture data is a list of 8 bit RGB values i.e. a list of 640x480x3 bytes. It can be read via a CORBA function call or directly on the on-board noteook computer by a reference pointer to avoid unnecessary network traffic over CORBA. The pictures will be jpg compressed and stored. At higher levels, e.g. for learning, a list of jpg pictures can be read. Further details can be found in the implementation reference guide.

3D Laser scanner A Laser range scanner produces a list of 2D points or a list of point clouds (array of 3D points) of three dimensional points representing the scanned object or environment.

Additionally, the SICK scanner can be operated in a mode where it returns remission values, i.e. the percentage of light in the Laser's spectrum that is reflected back to the scanner.

Wheel/Motor encoders The motor and wheel encoders are connected to the C167 microcontroller board and the data can be read via a CAN messages CAN Id. 0x01 resp. 0x09 (as specified in the appendix Sec. B.2.1).

Electrical current sensor The sensor for measuring electrical current is also connected to the C167 microcontroller board and the data can be read via a CAN messages CAN Id. 0x01 resp. 0x08 (as specified in the appendix sec. B.2.1).

6.4 Entity Structure Generation Module (ESGM)

6.4.1 General description

Entity structure types specify structured collections of attribute/typed value pairs. Entity trajectory structures consist of sequences of instantiations of a particular entity structure type (sequences of entity structures of the same type). The *entity structure generation module* is intended as both a repository or library of static pre-defined entity structure types and a generation mechanism for both static (pre-defined) and dynamic (defined on-the-fly) entity trajectory structures. The ESGM can also store previously generated entity trajectory structures for later use and comparisons. In essence, the ESGM can collect, structure and process time series data from different data sources in the robotic architecture using user defined sampling rates and other information provided as a *policy* to the module.

6.4.2 Interfaces

The ESGM can receive sensory data from three sources:

1. from the Feature Extraction component,
2. raw sensory data without prior feature extraction, and
3. from the Virtual Sensors.

The ESGM can be called by other components in the robot architecture with a *policy*. A policy provides the ESGM with the appropriate information to begin an entity trajectory structure generation process. The policy may refer to a pre-defined entity structure type or provide the entity structure type information on-the-fly. The policy may also include information as to duration of the process of generation and whether the entity trajectory structure should be stored for later use or simply passed on to the caller.

6.4.3 Data structures

The data structures used by the ESGM are policies, entity structure types, entity structures, entity trajectory structures, data structures generated by the feature extraction, data structures generated by the raw sensor data sources and data structures generated by the virtual sensors.

6.5 Event and Execution Monitor (EEM; was: Affordance monitor)

6.5.1 General description

The event and execution monitor serves two purposes:

1. It can be used to recognize the occurrence of *events*; and
2. it can be used to monitor the execution of actions and/or behaviors.

Such functionality plays a central role in the affordance support module which consists of both the EEM and the ESGM.

Recall that an affordance representation consists of a cue descriptor, a behavior descriptor and an outcome descriptor. For the constraints associated with a cue or outcome descriptor to be satisfied, a cue event or an outcome event must be recognized by the EEM. The EEM's role is to receive an event occurrence request and to call the ESGM with the proper policy to extract state sequences of the appropriate set of attributes associated with the cue or the outcome to be used to verify the occurrence or non-occurrence of the event within a specified temporal duration.

An instantiation of a behavior descriptor or an action type requires execution monitoring to determine whether the execution of the behavior or action instance is successful or not after its invocation. In a similar manner, the EEM's role is to receive an execution monitor request and to call the ESGM with the proper policy to extract state sequences of the appropriate set of attributes associated with the behavior or action to be used to verify the successful execution of the action or behavior within a specified temporal duration.

6.5.2 Module interfaces

The EEM receives the following requests as input:

1. A cue or outcome event monitoring request – An event occurrence request in the form of a parameterized instantiation of an event type is normally required as input to the EEM. Event types are represented as simple temporal networks (STNs). Event occurrence or non-occurrence replies are output by the monitors and sent to the original callers.

2. An action or behavior execution monitoring request – An execution monitoring request in the form of one or more metric interval temporal logic formulas (MITL) is normally required as input to the EEM. Execution success or failure replies are output by the monitors and sent to the original callers.

6.5.3 Data structures

The data structures used by the EEM are event monitoring request structures, execution monitoring request structures, policies (for the ESGM), event monitoring reply structures, execution monitoring reply structures, MITL formula representation structures, STN representation structures.

6.6 User Interface module

6.6.1 General description

This module displays status information about the robot (log data, sensor readings, etc.) and can be used to transfer commands, programs and hand-coded affordance representation from a user to the robot's on-board notebook computer.

6.6.2 Module interfaces

The Deliberation module receives goal descriptions from the UI. The Execution control module sends status information and log data to the UI. The UI sends hand-coded affordance representation triples to the Affordance Representation Repository. In "Remote Control" mode, the robot can be steered via a joy stick. The control commands are sent from the UI to the Execution module.

6.6.3 Data structures

The remote control commands exist. Their documentation needs to be included into the MACS reference documents. The affordance representation triples are described elsewhere in this document.

Data structures for the following information types need to be designed:

- Goal descriptions
- Status information
- Log data
- Sensor readings

6.7 Learning module

6.7.1 General description

This module is responsible to populate the *Affordance Representation Repository* using data obtained from the interaction of the robot with its environment. Specifically, the module reads the information provided by the *Perception Module*, starts its internal learning processes (see Deliverable D5.3.1 [5]) and stores the derived affordance representations in the *Affordance Representation Repository*. During the learning processes the *Learning Module* will also read the current content of the Affordance Representation Repository in order to be able to relearn, to recombine or to decide if the gathered knowledge differs from previous made experience. The interaction of the robot with its environment is either determined by its current goals or purposefully triggered by the *Learning Module* (by influencing the *Deliberation Module*). The *Learning Module* can influence the execution of behaviors by changing parameters of hand-coded routines via the *behavior* part of the affordance representation. A detailed description of the learning part will be provided in deliverable D5.3.2.

6.7.2 Module interfaces

Sends requests to the *Perception Module* to influence the data structures going from *Perception Module* to *Learning*. On the one hand the interface of the ESGM is used to request certain entity structures and on the other hand, the interfaces of the filters etc. are used to change the perception parameters.

Updates the *Affordances Repository* by

- sending newly derived affordance representations to the repository or by
- removing old triples from the repository, e.g. if they showed up to be incomplete/unsufficient or wrong.

Requests *affordance representations* ((c,b,e)-Triples) from the *Affordance Representation Repository* in order to be able to relearn, to recombine or to decide if the gathered knowledge differs from previous made experience.

Sends the above described data structures to the *Deliberation Module*, to be able to purposefully influence the behavior of the robot.

6.7.3 Data structures

The *Learning Module* receives the data structures as provided by the *Perception Module*, i.e. *entity structures* from the *ESGM* as well as raw and preprocessed sensor data.

The *Learning Module* transfers the internally derived cue-outcome characteristics into *affordance representations* (→Def.) to be stored within the *Affordance Representation Repository*. In addition the *Learning Module* requests *affordance representations* from the repository for relearning etc.

The data structures send to the Deliberation Module contains data to request the agent to

- act on a certain cue
- execute a certain behavior

- generate a certain outcome
- perform any goal directed behavior as provided by the *Goal Generation component* (see also definition of *User Interface* and ontology used in *Goal Generation component*).

The data structures to request certain entity structures, which are send to the *Perception Module*, are defined by the ESGM specification (see sec. 6.4).

6.7.4 Components

6.7.4.1 Application Spaces: The agent applies its behaviors to the environment. Thereby the agent permanently monitors its environment and the internal states before, during and after the application of the behavior. These time series are stored within behavior specific Application Spaces to be available for the learning processes. The begin (State_{t1}) and the end (State_{t2}) of the application of the behaviors must be marked within each stored time series. To be able to learn cues for the existence of an affordance and the consequences of using an affordance, the recorded time series have to include a certain amount of time before ($t[State_{t1} - \delta_{t1}, State_{t1}]$) and after the application of the action ($t[State_{t2}, State_{t2} + \delta_{t2}]$).. The *Application Spaces* module can be seen as an intermediate short time memory.

6.7.4.2 Partitioning: In the application space, belonging to a behavior b_i, sets of similar results of applications of behaviors should exist after a sufficient number of trials. To find these partitions is task of the Partitioning module. For this, teacher-based learning methods as well as self organised methods can be used.

6.7.4.3 Relevant Sensor Channel Extraction: The first step to extract essential information to learn characteristics of cue events and outcome events, is to find representative sensor channels in the time series of each partition within the application spaces. For learning coutcome event characteristics an example is: in case of performing a behavior that causes lifting an object, the partitions resulting from liftable and non liftable objects will differ in the height (y-position trajectory) and (optional) force sensor channel. For learning cue events, the channel of color blobs could be relevant to be used as a basis for extracting cue events for behavior lifting.

6.7.4.4 Cue Event / Outcome Event Characterisation: After the extraction of the relevant sensor channel(s), a description of what is characteristic for the extracted channel(s) is derived, i.e. cue event characteristics and outcome event characteristics are derived. These characteristics can then be used to enable the agent to recognise affordances (in case of characterising cue event related channels) or to monitor the outcome of the application of a behavior (in case of characterising outcome event related channels). There can be one or more charateristics for each partition of the concerning behaviors application space, since an application of a behavior can cause multiple outcome events and multiple cue events could be used to detect the existance of an affordance. In addition the similar characteristics could be extracted for more that one behavior, since one cue event could be characteristic for the detection of multiple affordances and since one outcome could

be caused by multiple behaviors. Thus the raletion between cue events, behaviors, and outcome events is a l:n:m relation.

6.7.4.5 Learning Control: This module is responsible for controlling the learning process that is described in deliverable D5.3.1. Therefor it coordinates the data flow within the *Learning Module* and invokes the previously described modules *Application Spaces*, *Partitioning*, *Relevant Sensor Channel Extraction*, and *Cue Event / Outcome Event Characterisation*, in the correct order, i.e. it is resposible for

- storing the data, received from perception, into the application spaces,
- deciding when there is enough data for starting the partitioning process,
- invoking the partitioning,
- invoking the extraction of relevant sensor channels for the outcome events and the cue events, once there is a valid partitioning of an application space,
- invoking the characterisation process, once the relevant sensor channels are extracted,
- starting a re-learning process,
- updating the *Affordance Representation Module*.

6.8 Affordance Representation Repository

6.8.1 General description

The Affordance Representation Repository stores affordance representations of the form (e, c, b) generated through manual design or through learning. The Repository has basic data base capabilities for managing the representations.

6.8.2 Module interfaces

The Affordance Representation Repository is populated with acquired affordance triples by the Learning module, or with hand-coded triples by the User Interface module. Affordance triples may be requested by the Deliberation module, or, more specifically, by the Planner. The Affordance Representation Repository examines the request information and returns the requested triples, if any.

6.8.3 Data structures

Affordance representation triples.

6.9 Deliberation module

6.9.1 General description

Gets the goal from the Goal Generation component, converts it into a plan using the Affordance Representation Repository and executes subtasks by passing the associated

affordance representations to the Execution Controller, and monitors the execution. In this sense, this module, which will not be fully specified now, should include at least a *Planner* and a *Scheduler*.

6.9.2 Module interfaces

- Goal description: A *goal description* is sent to Deliberation Module by the Goal Generation Component. Using the Planner and Scheduler subcomponents, Deliberation Module turns this goal description into affordance representations that can be employed by the Execution module.

- Execution request: An *execution request* is sent from Deliberation to Execution Control. It specifies a behavior to be executed at that point in the plan, together with other information for use by the Execution Control. These information include the cue descriptors, outcome descriptors and other possible constraints on the behavior.

  For more detail, see the interface descriptions of the Scheduler subcomponent.

- Status feedback: A *status feedback* is sent to Deliberation from Execution Control. The feedback indicates any problem during or before the execution of an affordance representation. Upon receiving information about such a problem, the Planner and Scheduler subcomponents review the old plan and generate alternatives.

- Request/receive affordance triple: To generate a plan, Deliberation Module accesses the affordance triples in Affordance Representation Repository through this interface.

  For more detail, see the interface descriptions of Planner and Scheduler subcomponents.

6.9.3 Data structures

Detailed specifications of goal descriptors, affordance representations, status feedback and the various request formats will be supplied later.

6.9.4 Goal Generation component

6.9.4.1 General description Converts the goal, given externally by the user or generated internally (e.g., by the Learning module), into a goal description for use by the Deliberation Module, or, to be more specific, the Planner component. The new description of the goal defines it in terms of the outcomes that must be created in the environment to achieve that goal.

6.9.4.2 Module interfaces

- User goal: A *user goal* is sent from User Interface to the Goal Generation component. It indicates a goal specified by the human user. The structure of the human goal representation is not clear at this point.

- Motivation for exploration and learning: A *motivation for exploration and learning* is set by the Learning Module. The Goal Generation Component, then, converts these into goal representations to be used by Deliberation Module. The structure to represent the motivation for learning and exploration is not clear at this point.

- Goal description: A *goal description* is sent from Goal Generation Component to Deliberation Module. It indicates the goal to be achieved in terms of the outcomes that must be created in the environment. These outcomes, in turn, must be of type *entity prototype*, to provide a basis for the Deliberation to make plans using affordance representation triples.

6.9.4.3 Data structures Detailed specifications of goal descriptors, affordance representations, status feedback and the various request formats will be supplied later.

6.9.5 Planner

6.9.5.1 General description The Planner gets the current goal and generates a sequence of subgoals, by using the affordance representation triples in the Affordance Representation Repository. These subgoals are, then, passed to the Scheduler.

The planning algorithm and other details about the Planner will be supplied later. We will probably employ a backward planner and adapt it to the task at hand. The Planner will make use of affordance representations, but it will not itself be affordance-based. We will leave this demanding topic to other projects.

6.9.5.2 Interfaces

- Goal description: The *goal description* is sent to the Planner from the Goal Generation Module. The Planner turns this goal description into a sequence of subgoals.

- Request/receive affordance triple: To generate a plan, the Planner accesses the affordance triples in Affordance Representation Repository through this interface. The Planner can query the Affordance Representation Repository by using either the outcome (goal oriented search), or the cue (entity oriented search), or the behavior (action oriented search).

- Replanning request: If a subgoal cannot be achieved, replanning is required. In such a situation, the Planner receives a replanning request that contains the available information about the circumstances of the failure to achieve the subgoal.

6.9.5.3 Data structures Affordance representation triples and more. Detailed specifications of goal descriptors and the various request formats will be supplied later.

6.9.6 Scheduler

6.9.6.1 General description The Scheduler receives the subgoal descriptions from the planner. It converts a subgoal into an executable subtask for accomplishing a subgoal. The information necessary to execute a subtask, including the behaviors that need to be invoked, are passed as an execution request to the Execution Control. It has not yet been decided whether this component should be part of the Deliberation module or part of the Execution module (cf. “Open questions” section at the end of this document). This also depends on the choice of Planner, which has to be decided.

6.9.6.2 Interfaces

- Execution request: An *execution request* is sent from the Scheduler to Execution Control. It includes a behavior descriptor for the Execution Control to execute at that point in the plan. Together with the behavior descriptor, cue descriptors, outcome descriptors and other possible constraints for the execution of the behavior are also included in the *execution request* to Execution Control. Then, Execution Control utilizes EEM to monitor the cues in the perceptual input, executes the behavior, and tries to validate the outcomes of the behavior.

6.9.6.3 Data structures Detailed specifications of the various request formats will be supplied later.

6.10 Execution control

6.10.1 General description

This module aims to execute a desired behavior which is specified in the affordance representation passed to it by the Deliberation Module as an affordance triple. It first sends the `<cue>` to the Event and Execution Monitor and waits for the Monitor to detect the cue in the environment. Once the cue is detected, then the corresponding behavior in the Behavior System is triggered and the `<outcome>` is sent to the Event and Execution Monitor to check whether the actualization of the affordance representation is going as expected. This controller also gets possible exception signals from the Behavior System and passes such exceptions to the Deliberation module so that it refines the plan.

Another functionality of the Execution Control module is to trigger active perception behaviors, when a specific cue is needed in the environment.

6.10.2 Interfaces

- Execution request: An *execution request* is sent from Deliberation module to Execution module. It specifies a behavior to be executed at that point in the plan, together with the cue and outcome descriptors for that behavior. Hence, an *execution request* must include an affordance representation triple. Optionally, the request might also include additional constraints on the actualization of the affordance representation. For example the request can include the affordance representation for 'liftability' along with the additional constraint of 'redness', if the plan requires the robot to lift not any liftable object, but specifically a red one.

- Status feedback: A *status feedback* is sent from Execution Control to Deliberation. The feedback indicates any problem during or before the execution of an affordance representation. These include the exceptions delivered from the Behavior Module, being unable to find the cues in the environment for a certain amount of time, and a nonmatch between the outcome description in the affordance representation and the actual outcomes of the behavior in the environment.

- Trigger/Stop behavior: A *trigger/stop* signal is sent from Execution Control to Behavior System. It indicates which behavior to start or stop executing. Along with the

trigger/stop signal, an identifier for the behavior is also passed to the Behavior Module.

- Perception configuration request: A *perception configuration request* is sent by the Execution Control to the Perception Module. This request configures the Perception Module to activate the ESGM and the corresponding filters for the specified entity prototypes. The configuration may be for monitoring affordance cue entity trajectories, affordance outcome trajectories as well as other arbitrary entities (e.g. to feed in into Learning Module).

- Active perception request: An *active perception request* is sent from Execution Control to Behavior System. When a `<cue>` is needed, the Execution Control initiates an Active Perception behavior in the Behavior System. Along with the request an identifier for the behavior is also passed to the Behavior Module.

- Replanning request: If a subgoal cannot be achieved, the Exedcution control sends a replanning request to the Planner component of the Deliberation module. It contains a reference to the failed subtasks, information of the robot's current state of plan execution progress, and information about the circumstances of the failure.

- `<cue>` request: A *cue request* is sent from Execution Control to the Event and Execution Monitor. The Execution Control requests the `<cue>` to be detected in the environment, so that the scheduled behavior can be executed after its detection. The existence of the `<cue>` in the environment is indicated by the *cue presence* signal from Event and Execution Monitor to Execution Control.

- `<outcome>` request: An *outcome request* is sent from Execution Control to the Event and Execution Monitor. The Execution Control requests the `<outcome>` to be detected during the execution of a behavior. The existence of the `<outcome>` in the environment is indicated by the *outcome presence* signal from Event and Execution Monitor to Execution Control.

- Behavior execution exceptions: A *behavior execution exception* is sent from the Behavior System to Execution Control. It indicates a problem or emergency situation during the execution of a behavior.

- Status information and Log data: This information is supplied by the Execution Control to the User Interface for human inspection.

6.10.3 Data structures

Detailed specifications of goal descriptors, affordance representations, status feedback and the various request formats will be supplied later.

6.11 Behavior System

6.11.1 General description

Module responsible for executing actions in the given environment. Triggered and monitored by the execution control high-level behavior (HLB) routines run sequentially or in

parallel trying to fulfil their particular task. The high-level behaviors will use (trigger or inhibit) a group of low-level reactive behaviors (LLBs) that work directly on the perceptual input in order to achieve robust and safe operation within a dynamic environment. They might as well trigger other high-level behaviors. The Behavior System can deliver feedback to other modules coded either as a degree of completion of their task or an activity rating depending on their current state and influence to the actuator control. This feedback can be understood as a virtual sensor.

The Behavior system includes hand-coded behavior routines. These hand-coded routines are used by the Execution control to execute a plan created by the Deliberation module (Planner component). This plan uses information from the Affordance Representation Repository which is either generated by the Learning module or by a user. In the latter case, the information has been put into the Affordance Representation Repository via the User Interface module. The architecture, in its current form, will enable the learning of `<cue>`s needed to trigger the behaviors as well as their `<outcome>`s. On termination of a behavior routine this routine returns whether it terminated normally or whether an exception occurred, e.g., a hardware defect (cf. [8]).

6.11.2 Low-level behaviors

6.11.2.1 General description Low-level behaviors fulfil simple tasks that are mostly reactive to the current sensor readings. This naturally implies a very tight coupling to the perception module for ensuring a safe and robust exploration of the environment. Parts of the processing of sensory information is done within these behaviors to achieve the necessary reaction speed. This processing is, hoewever, held very simple and is normally unimodal, i.e. restricted to only one sensory modality.

The feedback of the low-level behaviors to the complete system is normally organized as an activity in percent that is dependent on how strong the reaction of the behavior is triggered by the current environment.

6.11.2.2 Interfaces

- Trigger/stop low-level behavior: A *trigger/stop* signal is sent from a HLB to LLBs that are used/not used by the according HLB. The trigger can be enriched by parameters like e.g. a target pose.
- Feedback as virtual sensor: LLBs can report their internal state of activation that changes dynamically with respect to how active they are in the current environmental setup as a virtual sensor reading.
- Input from Perception Module: LLBs have a tightly coupled action-perception loop. They mainly need fast access to the readings of the laser scanner.

6.11.2.3 Data structures LLBs are parameterized by the HLBs or by predefined configuration settings suitable for the actual robot.

6.11.3 High-level and affordance related behaviors

6.11.3.1 General description High-level behaviors are a group of goal oriented behaviors. They fulfil more complex tasks than merely reacting to one aspect of sensory

input. Furhtermore they can make use of other high-level or low-level behaviors. These behaviors are meant in a way that they can achieve goals like picking up items, carrying them around or pushing something. Therefore they are not meant to execute complete plans but instead to fulfil more basic subgoals. Thus the high-level behaviors are triggered by the execution control module.

The subgroup of affordance related high-level behaviors are those whose successfull execution implies that an item being the target to an action affords that action. An example would be the affordance of liftability after successfully lifting an item.

The feedback of the high-level behaviors to the complete system is normally organized as a degree of completion as a virtual sensor reading.

6.11.3.2 Interfaces

- Trigger/stop high-level behavior: A *trigger/stop* signal is sent from the Execution Control or another HLB stopping/starting a behavior. If a behavior is triggerd the Execution Control has to specify certain parameters (depending on the actual behavior) for meaningful execution. These parameters may contain cues to look for, paths to follow or entity structures (and their poses) that should be subject to an action. Time constraints can be defined as well. See [8] for a more detailed description.
- Feedback: HLBs can give a feedback to the execution control that might consist of information of the termination conditions, e.g. whether a normal termination or an exception occurred, along with information about the current status. They furthermore can return a *degree of completion* of their task if their execution consists of multiple steps. For example, a *Lift* behavior has to sequentially follow the steps of orienting the crane above a test object, lowering the magnet, switching it on, lifting the test object up and probably turning the crane out of the scan field. After the behavior switched to a new step, the degree of completion would be a certain percentage higher.

6.11.3.3 Data structures HLBs need to be parameterized with information cues, entity structures or even locations relative to the robot. Furthermore, time constraints can be specified. Feedback return values are normally simple percentages within a range of $[0, 1]$ depicting a degree of completion of the task.

6.12 Actuators

6.12.1 General description

There are five different types of actuators involved in the operation of the robot:

- DC motors that drive the robot's wheels
- DC motors that move the crane
- the crane's magnet (on/off)
- servo motors that pan/tilt the cameras
- servo motor that tilts the laser scanner

Drive motors The robot has three wheels on each side, which are connected by a toothbelt. A single DC motor drives each side. The current for the motors is delivered by a motor controller board, which is plugged in a socket on the robot's mainboard. The motor voltage is controlled by pulse width modulation. The necessary pwm signals are generated by the microcontroller (Infineon C167) and sent to an IC of type LMD18200, which is the main part of the motor controller board.

Crane actuators The crane's motors and the magnet are controlled by a separate TMC200 board, which is able to control up to 3 motors with a power of 200 W. One digital port of the TMC200 is used to switch the magnet on or off. The 3 motor channels are used for:

- rotational movement of the boom
- horizontal shift of the magnet
- to lift the magnet up or down

The crane's operation can be monitored by some sensors that are also connected to the TMC200 (cf. section 6.3).

Servo motors for sensors Servos are used for performing pan and/or tilt motions of KURT3D's cameras and the Laser scanner. It enables the robot to actively perceive its environment without moving the robot or in cooperation while moving the robot. The servos are controlled with a small servo device (S10) which can be used to control 10 different servos.

6.12.2 Interfaces

Drive motors and crane actuators The actuators are controlled by the Behavior System using a specific API that abstracts from the CAN messages described above. All functions return a character string describing the reason for an eventual error that occurred during the execution. A NULL pointer is returned if the function was completed successfully. The API is described in the appendix (sec. B.2.2).

Servos The servos are connected to the servo device S10. The servo device itself is connected via serial RS232 or via a USB to RS232 device to the on-board notebook computer.

6.12.3 Data structures

Drive motors and crane actuators The drive motors and crane actuators described above are triggered by the Behavior System that runs on the robot's on-board notebook computer. It sends related CAN messages to the C167 MC. These CAN messages have CAN Id. 0x01 resp. 0x21 and are discriminated by the first two bytes of the data frame, which defines the control mode. A detailed description of the data structures can be found in the appendix (sec. B.2.1).

Servos Each servo can be set to a certain position via a simple ASCII command e.g. "Sn,g#" (move servo n to position $g \in [0, 550]$). Each servo position n is correlated with an angle of $[0°, 170°]$ degree. Further details of the implemented servo class can be found in the implementation reference guide.

7 Outlook

The next steps in the development of the affordance-based control architecture will be its continued implementation. The resulting system will be tested and evaluated in our demonstrator scenario. The results of the evaluation will be used to further refine the architecture.

7.1 Next steps

More specifically, we will implement the following components:

- Deliberation module including Planner, Scheduler, Goal Generation component
- Event and Execution monitor
- Learning Module, including learning control, application spaces, cue and outcome event characterizations, relevant sensor channel extraction and partitioning components
- Affordance Representation Repository
- Virtual sensors

We will continue the implementation of

- ESGM (SW designation "DyKnow")
- Perception module, including Feature extraction
- Behavior system
- User Interface module
- CORBA communications layer
- Execution module

We will add more detailed specifications of

- Entity structures for affordance triples, specifically for `<cue>` and `<outcome>`
- Status information transmitted from Behavior system to Perception module
- Status information, log data, and sensor readings for display on User Interface

7.2 Open questions

The following major questions have to be settled in the near future:

- *Deliberation Module:*
  - Planner: Which existing planner can we use in the first attempt?
    It would be very interesting as well as challenging to develop an affordance based planner in order to make use of all the possible advantages of the affordance approach. Unfortunately this would clearly exceed the scope of the MACS project. To show the general benefits of our concept a standard planner would suffice. The remaining question is how it can be supported on a symbolic level (how can we deal with time and continuous event descriptions?).

  - Does the planner need direct access to the PM (actual sensor data stream) or will it get all necessary information from EC only when a new plan is needed?
  - Where is the Scheduler located: in Deliberation / Planner or in Execution Control?
  - how can the goal be represented for this planner? and how can the LM influence that?
  - what format does the plan have

- *Execution Control:*
  - can it work on complete plans (and subplans)?
  - what kind of decision competencies does EC have before the planner has to be informed? How abstract and dependant on external input can the plans be described?
  - how does it monitor the behaviors? Does it have to have special knowledge about every single behavior or is there a standardized interface method? What signals are coming from the behaviors?
  - Cue- & Effect-Monitor: right now pure reactive behavior based on affordances is not possible for only the DM has an interface to AR. Should a new affordance list function be added to the CEM instead? The CEM could deliver a list of perceived affordances at every point in time - based on the AR (then a link has to be established between AR and CEM). This would support pure reactive behavior as well as plan based behavior on the level of the EC. This way the level of detail of a plan is determining the kind of behavior of the system. In this case the search for a certain cue (original function) is only a special case using a predefined filter mechanism to narrow down the list of perceived affordances.

- *Behavior System:*
  - right now we are dealing with different behavior paradigms at the same time - we use actions (with certain effects) and behaviors (with their outcomes) in our BS:
  - how can they be represented in PM, AR, Planner and EC (with duration, constraints and parameters if necessary)?
  - what kind of behaviors are used here? is there a standard description/interface or are different kinds of behaviors in use? do the have activation values, completion values, etc.?
  - We need to analyze the timing constraints for perception support
  - How are behaviors triggered? - how are they stopped?
  - exceptions: are they part of internal sensors or is there a direct link to EC?
  - It may be reasonable to bring EC and BS closer together in the implementation. It could be much easier to "corbarize" only the EC as of an execution request to start and parameterize the behaviors directly from there.

- *Perception Module:*

  - Specification of internal/virtual sensors to sense internal states (timings, synchonization constraints)
  - Should internal sensors be handled separately e.g. for checking preconditions for robot behavior execution?

- *Learning Module:*
  - Hand-coding of affordance triples
  - If we use MITL/ temporal logic based formalization: can those formulas be learned - and how?

7.3 Future research

Given the currently investigated affordance-inspired architectural approach, a number of questions for future research can already be identified:

- Since the presented approach focuses on a pure function-based perception component, it would be interesting to investigate the interplay between individual object recognition and affordance-inspired function-based perception. One possibility would be to have a closer look on Neisser's work [13], who already proposed a combination of an affordance system and an object recognition system.
- The successful implementation of a 'learning by experimentation' method for generating affordance representations is a prerequisite for more complex learning approaches, like affordance representation learning by imitation. Learning by imitation is a demanding research topic on its own, aspects of it are being investigated in the Cognitive Systems projects RobotCub, JAST, MindRACES and Hermes. Affordance representation learning by imitation could be a joint follow-up activity of MACS and some of these projects.
- The current architectural approach employs a standard type planner that makes use of affordance representations. That is, known planning algorithms are applied to suitably transformed goal and task descriptions that include affordance representations. A future research topic could be the question as to how the planning algorithms themselves could make use of affordances.

8 Acknowledgements

The authors express their gratitude to Prof. Dr. Joachim Hertzberg for reviewing a draft version of this document. His valuable remarks helped us to improve this document.

References

[1] Rainer Worst, Claus Hoffmann, Björn Wingman, Maya Çakmak, and Martin Hülse. Specification of module interfaces. Deliverable MACS/1/1.2 v2, Fraunhofer Institut für Autonome Intelligente Systeme, Sankt Augustin, Germany, 2005.

[2] Erol Şahin, Ralph Breithaupt, Erich Rome, and Patrick Doherty. Identification of architectural requirements of an affordance-based control. Deliverable MACS/2/1.1 v3, Middle East Technical University Dept. of Computer Engineering, Ankara, Turkey, 2005.

[3] Lucas Paletta, Gerald Fritz, Erol Şahin, and Manish Kumar. Affordance recognition from visual cues. Deliverable MACS/3/1.2 v1, Joanneum Research Institute of Digital Image Processing Computational Perception (CAPE), Graz, Austria, 2004.

[4] Patrick Doherty, Torsten Merz, Piotr Rudol, and Mariusz Wzorek. Tentative proposal for a formal theory of affordances
tentative proposal for an affordance support architecture
prototype: Affordance-based motion planner. Technical Report MACS/4/2.1 v1, Linköpings Universitet, IDA Group, Linköping, Sweden, 2005.

[5] Georg Dorffner, Jörg Irran, Florian Kintzler, and Patrick Pölz. Robotic learning architecture that can be taught by manually putting the robot through action sequences. Deliverable MACS/5/3.1 v1, österreichische Studiengesellschaft für Kybernetik (öSGK), Vienna, Austria, 2005.

[6] Ralph Breithaupt, Simone Frintrop, Joachim Hertzberg, Erich Rome, and Bernd S. Müller. Specification of final demonstrator. Deliverable MACS/6/1.1 v2, Fraunhofer Institut für Autonome Intelligente Systeme, Sankt Augustin, Germany, 2004.

[7] Ralph Breithaupt, Simone Frintrop, Erol Şahin, Joachim Hertzberg, Patrick Pölz, Piotr Rudol, Emre Uğur, Patrick Doherty, Erich Rome, and Bernd S. Müller. Report on experiment design. Deliverable MACS/6/4.1 v1, Fraunhofer Institut für Autonome Intelligente Systeme, Sankt Augustin, Germany, 2004.

[8] Christopher Lörken and Hartmut Surmann. Specification of a prototype behavior system. Technical Report Draft Version 0.5, Fraunhofer AIS, March 2006.

[9] Erich Rome, Ralph Breithaupt, Lucas Paletta, Patrick Doherty, Erol Şahin, and Georg Dorffner. Periodic activity report year 1. Deliverable MACS/0/1.3 v1, Fraunhofer Institut für Autonome Intelligente Systeme, Sankt Augustin, Germany, 2005.

[10] Erich Rome. Macs project web site, 2006. http://www.macs-eu.org/.

[11] James J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, Boston, MA, 1979.

[12] Rainer Worst. Kurt2 c167 firmware documentation, 2006. http://www.ais.fraunhofer.de/KURT2/v300/k2167htm/index.html.

[13] U. Neisser. *Cognitive psychology*. Prentice-Hall, Englewood Cliffs, NJ, 1967.

[14] Merriam-Webster online. Definition: Actuator, 2006. http://www.m-w.com/-dictionary/actuator.

[15] Editors of Merriam-Webster, editor. *Webster's New Encyclopedic Dictionary*. Federal Street Press, Jan 2002.

[16] Wikipedia. Definition: Feature extraction, 2006. http://en.wikipedia.org/wiki/Feature_extraction.

[17] Patrick Doyle. Definitions: Planner, plan, scheduler, 2006. http://www.cs.dartmouth.edu/brd/Teaching/AI/Lectures/Summaries/-planning.html.

[18] Inc. Stottler Henke Associates. Definition: Machine learning (glossary of ai terms, 2006. http://www.stottlerhenke.com/ai_general/glossary.htm.

[19] Merriam-Webster online. Definition: Perception, July 2006. http://www.m-w.com/dictionary/perception.

[20] Encyclopedia Britannica online. Definition: Perception, July 2006. http://www.britannica.com/eb/article-219082?query=perception&ct=.

[21] EU NoE Planet. Definitions: Planner, plan, scheduler, 2006. http://scom.hud.ac.uk/planet/repository/.

A Glossary of Terms

Affordance Representation Repository: Stores affordance representations (→def.) generated through manual design or through learning.

Action: An *action* is a distinguished subset of events deemed under an agent's control. In artificial intelligence, actions generally have a structure consisting of a *precondition* representing the constraints required to successfully invoke the action, *durative conditions* (if the action has duration) representing the causal dependencies and *durative effects* the action has during its execution and a *postcondition* representing the *direct effects* the action has. In the context of a causal theory, one often distinguishes direct effects from *indirect effects* or *ramifications* of the action. The term →*outcome* will be used as another term for the direct effect of an action, or in the case of causal theories, for the combination of direct, durative and indirect effects.

Actuators: "a mechanical device for moving or controlling something" (definition from Merriam-Webster, [14]).

Affordance-based behaviors: An affordance-based behavior is an enhancement of a robot behavior by enriching the sensor space and knowledge base of an artificial agent by affordance information. Robot behaviors that are especially prepared for such enhancements are called →*affordance-related behaviors*.

Affordance-related behaviors: A group of →*high-level behaviors* that are directly related to agent affordances (→def.) in terms of manipulating objects or trying to manipulate objects. Examples would be behaviors that lift, push or carry items.

Affordance hypothesis verification: Suppose the robot perceives a cue c that supports the presence of an affordance A represented by at least triple (o,c,b). A offers the potential of executing a behavior b. The robot hypothesizes that the execution of the behavior b leads to the outcome described by o. During the execution of b (and eventually a certain time thereafter) it observes the outcome o'. During the whole time period, the outcome o' must match to the hypothesized outcome o. In this case the affordace hypothesis is verified. The step of comparing o' and o is then called *Affordance Hyphothesis Verification*.

Behavior system: The collection of →*robot behaviors* for a given robot.

Behavior: See →*robot behavior*.

Components: Refers to modules and other parts of the robot control architecture. Typically, a part of a module would be named component.

Cue: "Something serving as a signal or suggestion ... `<hint>` ... a suggestion for action given briefly or in an indirect manner ..." [15]. Here: the perceptual data part of an →*affordance representation* that supports the existence of the associated agent affordance.

Deliberation module: →*Components* of the control architecture that enable deliberate or planned acts.

Entity schema: An entity structure (→def.) that consists of pairs of attributes and value ranges, e.g.—arbitrary notation used here—((form, circular), (sizes, (10:20, 30:50, 80:120))).

External sensor: →*Robot sensor* for perceiving the world. An external or exteroperceptive sensor is a device that senses the surrounding environment, i.e. a sensor that takes measurements of the surrounding environment and translates these measurements into useful data for the robot. Including, but not limited to: cameras, ranging sensors, odometry sensors, gyroscopes.

Feature extraction: "Feature extraction is a special form of dimensionality reduction and is in the area of image processing also connected with shape recognition. ... It can be used in the area of image processing which involves using algorithms to detect and isolate various desired portions or shapes (features) of a digitized image or video stream." (definition from Wikipedia, [16]). Feature extraction may employ specialized →*filters* for implementing the extraction of particular features from an image.

Filter: A computer program to transform or selectively reduce the amount of information in a set of data or a data stream. Also: analog or digital image processing (IP) operations to enhance or modify an image. Here, the data stream is a stream of sensory data. Filters for a stream of image data are called *image filters*. Image filters take an image as input, perform certain operations on it, and return the result image. An example is a Gaussian blur filter that smoothes a small image portion around a center pixel in a specified radius.

Goal: "Usually specified as one or a set of world states. There are three kinds of goals: maintaining some condition, preventing some condition from occurring, or sequencing activities." [17]

High-Level / Complex behaviors: Goal-oriented robot behaviors that fulfil more complex tasks than merely reacting to perceptual stimuli. They often may use other high-level or →*low-level behaviors* to reach their →*goal*. For example, a behavior that explores the environment by wandering around may use reactive behaviors for driving and for avoiding obstacles. All →*affordance-related behaviors* are high-level behaviors.

Internal sensor: An internal or proprioceptive sensor is a →*robot sensor* device that monitors and senses conditions of the robot and its hardware. Including, but not limited to: temperature sensor for monitoring the internal temperature, inclinometers for sensing the robot pose, voltage sensors for monitoring the battery status.

Learning module: Uses →*machine learning* techniques for acquiring and structuring information and for realizing adaptivity.

Low-Level / Simple behaviors: See →*reactive behaviors*.

Machine learning: "Machine learning refers to the ability of computers to automatically acquire new knowledge, learning from, for example, past cases or experience, from the computer's own experiences, or from exploration. ... Machine learning enables computer software to adapt to changing circumstances, enabling it to make better decisions than non-AI software." (definition from Stottler Henke Associates, Inc. Glossary of AI terms, [18]).

Outcome: Here: the perceptual data part of an →*affordance representation* that is employed to verify the robot's hypothesis of the expected results of its behaviors (→*affordance hypothesis verification*). Outcome refers to a situation and thus may have a temporal component.

Perception module: "... 3a: awareness of the elements of environment through physical sensation ..." (definition from Merriam-Webster online [19]). Also: "In perception the environment is scanned by means of various sensory organs, real or artificial, and the scene is decomposed into separate objects in various spatial relationships." (definition from Encyclopedia Britannica online, article on Artificial Intelligence [20]). In the context of robotic systems, this module should generate an interpretation, using the raw spatiotemporal multimodal data incoming from extero- and proprioceptive →*robot sensors* (see sec. 6.2), for the benefit of the →*execution module*, the →*behavior system*, and the →*learning module*.

Plan: "A plan is a representation of a course of action." [17]

Planning: "Deciding upon a course of action before acting. A →*plan* is a representation of a course of action. A finished plan is a linear or partially ordered sequence of operators. Planning is reasoning about future events in order to verify the existence of a reasonable series of actions to take in order to accomplish a →*goal*. " [17]

Planner: "Artificial Intelligence (AI) Planning is areas of study concerned with the automatic generation of a →*plan* to solve a problem within a particular domain. Given an initial state, the planner tries to find the actions required to achieve some →*goal* conditions." [21]

Raw sensor data: →*Robot sensor* data that are provided by the →*Perception module* without prior →*feature extraction*. For the MACS architecture, raw sensor data might have passed synchronisation services and/or transformation into a suitable data structure, like an entity frame (→def.). Such services are provided, for instance, by the →*Entity Structure Generation Module*.

Reactive behaviors: →*Robot behaviors* with a very tight perception-action coupling that have the need to react fast to sensory input. For example, a reactive brake behavior will stop the robot directly if an obstacle is detected in close proximity in heading direction.

(Robot) behavior: A robot control routine using a perception-action cycle for performing certain limited actions. For example, a "wall following behavior" may use range sensors to keep the robot at a specified distance from a wall while driving along that wall at a specified speed.

Robot sensors: A sensor for a mobile robot is a physical device that detects or senses a signal or physical condition of the robot or the robot's environment. A robot sensor yields sensory data that can be processed by a robot's perception software, e.g. the →*perception module* in general and its →*feature extraction* part, for instance. The robot's hardware sensors are coarsely divided into →*external* and →*internal sensors*. →*Virtual* (software) sensors can be added.

Scheduler: "Scheduling is deciding how to allocate one or more resources to accomplish particular activities over time so that input demands are met in a timely and cost-effective manner. Most typically, this involves determining a set of activity start and end times, together with resource assignments, which satisfy all temporal constraints on activity execution, satisfy resource capacity constraints and optimize some set of performance objectives to the extent possible." [21]

Virtual sensor: Virtual sensors are software modules that filter or combine sensory data in order to generate new information. →*Feature extraction* may be viewed as a virtual sensor. Monitors for control program state information may be implemented as virtual sensors, too.

B Additional descriptions of architectural components

B.1 Sensors

B.1.1 Additional sensor descriptions

External sensors

3D Laser Scanner Some technical data: The SICK LMS 200 has a wide range (up to 80m), a weight of 4.5 kg and a power consumption of 20 Watt. The dimensions of the LMS scanner are (155 mm x 21 0mm x 156 mm). The horizontal opening angle of the SICK scanner is 180^{deg}, the vertical opening angle depends on the servo control and hardware constraints and has an upper limit of 120^{deg}.

3D scannning generates accurate consistent 3D data (resolution 1cm) if the robot stands still, which is suitable for mapping tasks. 3D scans can be delivered with different resolutions from 0.4 to 0.1 Hz according the resolution. Continuous operation while the robot is moving leads to more complex and real-time acquisition of the 2D laser data together with some internal sensor data e.g. a motor/servo and the use of odometery and/or gyro data to correct the 3D data according to the robot pose. Therefore, planes of the 3D data are continuously updated with rate of 36 Hz (1 deg. resolution).

The point clouds produced by 3D scanners are not only used directly e.g. for the navigation. To build a complete digital model of the environment different 3D point clouds

from different positions have to be registered in common coordinate system. These registration involves finding and connecting adjacent data points in order to create a continuous digital model of 3D points. These 3D points are the basis for the feature extraction and reconstruction of the higher perceptual functions. Furthermore, the transformation found by the registration algorithm is used to correct the robot pose estimate by other sensors. A typical example for a registration method is the iterative closest point (ICP) algorithm.

Internal sensors

KURT2 IMU The KURT2 IMU is a small device with one Gyroscope to measure the yaw rotation and two acceleration sensors to measure the pitch and role inclination. The inclinations are robust and precise if the robot is standing still or moving constant. Furthermore, the IMU has an analog compass (H6100). The device is attached directly to KURT2's C167 module and the data can be read via a can messages (as specified in sec. B.2.1). The IMU is an optional sensor that is currently only present in the KURT3D robot of FhG/AIS.

B.1.2 Interfaces

Kurt2-IMU The device is attached to the C167 microcontroller module of the robot.

B.1.3 Data structures

Kurt2-IMU The IMU is connected to the C167 microcontroller board and the data can be read via a CAN messages CAN Id. 0x0D resp. 0x0E resp. 0x1E (as specified in the appendix sec. B.2.1).

B.2 Actuators

B.2.1 Actuator data structures

- *CAN Id. 0x01* provides the following control modes:
  - 0x0000 RAW control mode:
    The two motors are controlled by three parameters: direction of rotation, brake (switch motor on or off), and pulse width. Direction is controlled by one bit, where 0 means "forward" and 1 means "back". Brake is controlled by another bit, where 0 means "brake off" (i.e. actually drive) and 1 means "brake on". Pulse width for left and right motor may be set separately to a value between 0 and 1024, where the meaning of 0 is "full power", 256 is corresponding to a duty cycle of 75 %, 512 is corresponding to a duty cycle of 50 %, and the meaning of 1023 is "no power at all". The intermediate values are treated in an analogous manner. This definition is in direct accordance with the operation of the C167's pulse width modulation module as described in chapter 15 of the C167 user's manual.
  - 0x0001 SPEED control mode:
    The speed for left and right motor may be set separately to a value between -100 and +100, where the meaning of -100 is maximum speed back, 0 means

no speed at all, and the meaning of +100 is maximum speed forward. The intermediate values are treated in an analogous manner. The desired speed will be maintained by the microcontroller.

- *CAN Id. 0x21* provides the following control modes for the crane:
  - 0x0001 SPEED control mode:
    The speed for the crane's three motors (bytes 3 and 4 for the first, bytes 5 and 6 for the second, and bytes 7 and 8 for the third motor) are set separately to a value between -100 and +100, where the meaning of -100 is maximum speed back, 0 means no speed at all, and the meaning of +100 is maximum speed forward. The intermediate values are treated in an analogous manner. The desired speed will be maintained by the microcontroller.
  - 0xFF00 MAGNET_ON control mode:
    The magnet's state is switched to "on".
  - 0xFF01 MAGNET_OFF control mode:
    The magnet's state is switched to "off".
    The current values of the crane-related sensors are transmitted from the TMC200 to the on-board notebook computer using CAN Id. 0x29 and 0x2A.
- *CAN Id 0x29* is used for the values of the motor encoders, which are transmitted in 3 groups of 2 bytes, meaning number of encoder signals per 10 msec.
- *CAN Id 0x2A* is used for the remaining sensors, i.e., bytes 1 and 2 give the rotational position of the boom, byte 3 gives the front limit switch of motor 2, byte 4 gives the back limit switch of motor 2, byte 5 gives the top limit switch of motor 3, byte 6 is reserved, and bytes 7 and 8 give the current load.

B.2.2 Interfaces

The actuator API includes the following functions:

- *char *can_motor(int left_pwm, char left_dir, char left_brake, int right_pwm, char right_dir, char right_brake)*

  The function *can_motor* controls the left and right motor. The two motors are controlled by three parameters: pulse width, direction of rotation and brake (switch motor on or off).

  Pulse width for left and right motor may be set separately to a value between 0 and 1024, where the meaning of 0 is "full power", 256 is corresponding to a duty cycle of 75 %, 512 is corresponding to a duty cycle of 50 %, and the meaning of 1024 is "no power at all". The intermediate values are treated in an analogous manner.

  Direction is controlled by one character, where 0 means "forward" and 1 means "back".

  Brake is controlled by one character, where 0 means "brake off" (i.e. actually drive) and 1 means "brake on".

- *char *can_speed(int left_speed, int right_speed)*

  The function *can_speed* sets the desired speed of the left and right motor. The speed for left and right motor may be set separately to a value between -100 and +100, where the meaning of -100 is maximum speed back, 0 means no speed at all, and the meaning of +100 is maximum speed forward. The intermediate values are treated in an analogous manner. The desired speed will be maintained by the firmware with means of a proportional controller.

- *char *can_cr_drive(int motor1, int motor2, int motor3)*

  The function *can_cr_drive* sets the voltage (PWM signal) of the 3 crane motors. The voltage for the motors may be set separately to a value between -100 and +100, where the meaning of -100 is maximum voltage back, 0 means no voltage at all, and the meaning of +100 is maximum voltage forward. The intermediate values are treated in an analogous manner.

  - motor1: rotation, positive values for counter-clockwise movement.
  - motor2: horizontal shift, positive values for outside movement.
  - motor3: lift, positive values for lifting down.

- *char *can_cr_on(void)* The function *can_cr_on* switches on the crane's magnet.

- *char *can_cr_off(void)* The function *can_cr_off* switches off the crane's magnet.

- *char *can_cr_encoder(int *motor1, int *motor2, int *motor3)*

  The function *can_cr_encoder* delivers the current values of the motor's encoders in raw format, represented as number of encoder signals per 10 msec.

  - motor1: rotation, positive values for counter-clockwise movement
  - motor2: horizontal shift, positive values for outside movement
  - motor3: lift, positive values for lifting down

- *char *can_cr_sensors(int *m1pos, char *m2front, char *m2back, char *m3top, int *load)*

  The function *can_cr_sensors* delivers the current values of the other crane sensors in degrees:

  - m1pos: position of motor 1 (0 - 359), 0 if the arm is directed straight to the back
  - m2front: limit switch of motor 2, front side, 0 if pressed and 1 otherwise
  - m2back: limit switch of motor 2, back side, 0 if pressed and 1 otherwise
  - m3top: limit switch of motor 3 on top, 0 if pressed and 1 otherwise
  - load: weight of the load currently lifted by the crane