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

D2.3.1 Implementation of the affordance-based control architecture

Due date of deliverable: November 30, 2006
Actual submission date v1: February 20, 2007

Start date of project: September 1, 2004 Duration: 39 months

Middle-East Technical University (METU-KOVAN)

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 D2.3.1

Implementation of the Affordance-based Control Architecture

Erol Şahin, Erich Rome, Lucas Paletta, Ralph Breithaupt, Georg Dorffner, Mehmet R. Doğar, Christopher Lörken, Emre Uğur, Maya Çakmak, Ali Emre Turgut, Florian Kintzler, Gerald Fritz, Hartmut Surmann, Patrick Doherty, Fredrik Heintz, Mariusz Wzorek, Björn Wingman, Piotr Rudol

Number: **MACS/2/3.1**
WP: *2.3*
Status: *draft, version 1*
Created at: *Jan 2, 2007*
Revised at: *v1 – February 20, 2007*
Internal rev:

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
UOS	Universität Osnabrück, Osnabrück, D

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:

Asst. Prof. Dr. Erol Şahin
Middle East Technical University
Dept. of Computer Engineering
Inonu Bulvari
TR-06531 Ankara, Turkey

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
Institute of Digital Image Processing
Computational Perception (CAPE)
Wastiangasse 6
A-8010 Graz
Austria

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

Contact:
Dr. Lucas Paletta

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:
Asst. Prof. Dr. Erol Şahin

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

Tel.: +43 1 5336112 0

Contact:
Prof. Dr. Georg Dorffner

Universität Osnabrück
Institut für Informatik
Albrechtstr. 28
D-49076 Osnabrück
Germany

Tel.: +49 541 969 2622

Contact:
Prof. Dr. Joachim Hertzberg

Contents

1 Introduction

This document reports the current state of the implementation of the affordance-based robot control architecture that was specified in Deliverable D2.2.2 [1].

This document aims to integrate the implementation efforts of MACS partners into a single architectural framework and provide each partner with a general picture of the current status of the implementation. This is a "living document", in the sense that it will be updated as the implementation of the MACS architecture progresses. The current version reports the implementation status of the MACS architecture at the time of the second MACS review meeting.

In order to ease the readability of the document and make it as much self-contained as possible, some parts from Deliverable D2.2.2 "Development of an Affordance-based Control Architecture" are used, in the introductory parts of this document. This includes the next section, which briefly overviews the architectural design and its components. After that, the architectural components and control and data flow between them are illustrated by an appropriate diagram, which is also taken from D2.2.2.

The status of implementation for each component of the architecture are presented in Section 4. For each module, the implementation of the internal components, the implemented module interfaces and the current status of integration with other modules are reported. The results of module testing experiments are presented wherever available.

2 Overview of Architecture Design

In this section, an overview of the MACS architecture is presented to provide the reader with a quick introduction to main components and their basic functionality. The description is taken from D2.2.2 “Development of an Affordance-based Control Architecture”.

2.1 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 parameterizable 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.

3 Architecture diagram

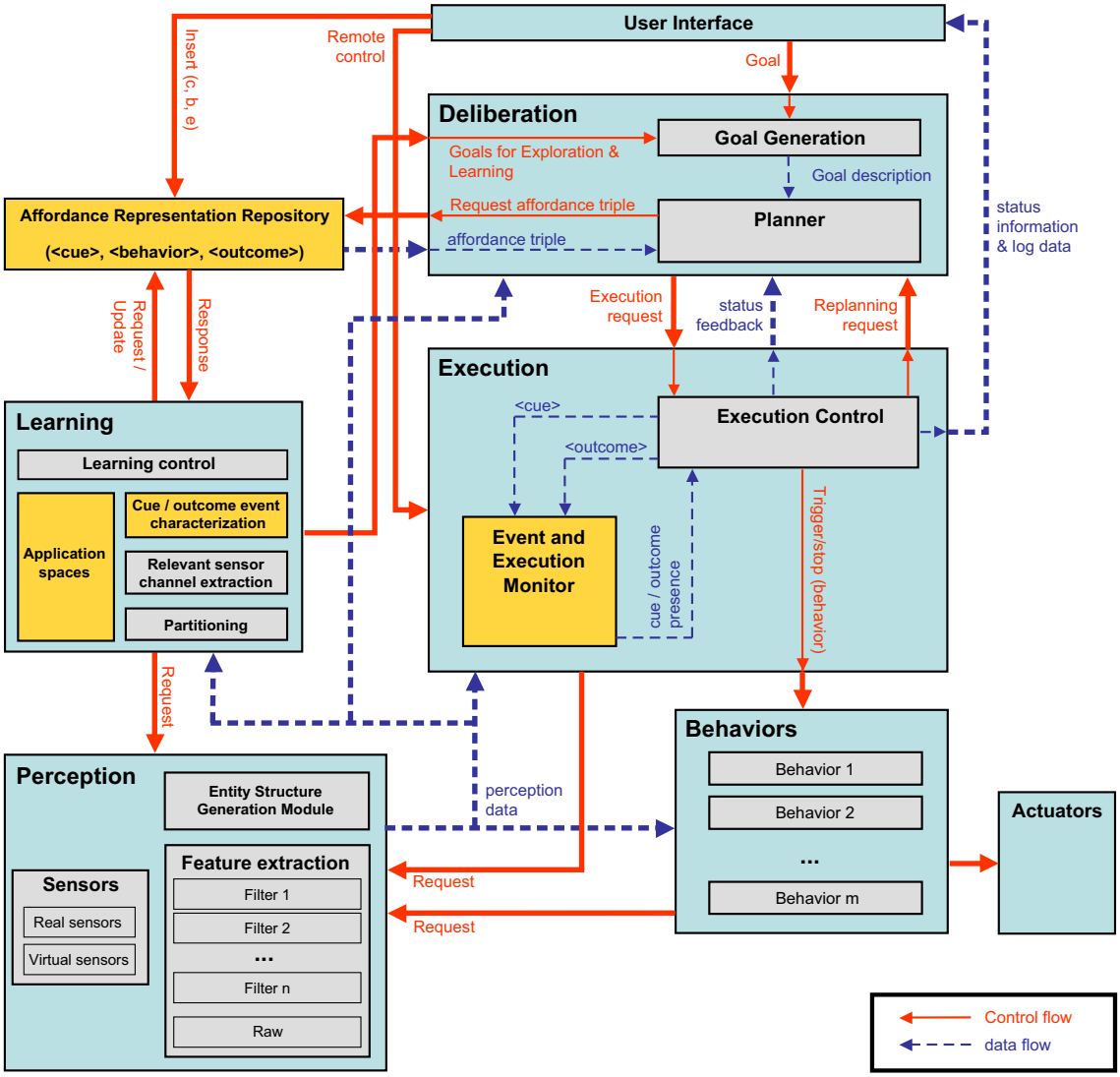

Figure 1: Modules, data and control flow of the MACS control architecture.

Red arrows between components A and B in the diagram (Fig. 1) are 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.

Blue, dashed arrows between components A and B in the diagram are of type data flow. The data flow arrows do 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.

4 Components

This section gives an account of the current state of implementation for each architectural module. This includes the implementation of each module along with their CORBA interface specifications, and their state of integration with other modules through these interfaces.

4.1 Robot computing hardware

4.1.1 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.

4.1.2 Current state of implementation

The robot computing hardware is fully implemented and functional. The details about the implementation and interfaces can be found in deliverables D6.3.1 and D1.3.1 [2; 3].

4.2 Perception module

4.2.1 General description

This module generates a perceptual view of the environment, using the raw data incoming from extero- and proprioceptive sensors. Figure 2 gives a more detailed view about the perception module from an architecture perspective. The perception module (PM) consist of the following parts. The Entity Structure Generation Module (ESGM) packs the results of sensor readings and their processing outputs into entity structures (be it entity or entity trajectory) to be used by the Learning module and the Event and Execution Monitor Module. See 4.4 for a general discussion as well as deliverable D4.4.1 and D4.3.2 for a more detailed view into ESGM and EEM [4; 5]. The sensor toolbox (ST) is the interface to extero- and proprioceptive sensors. Section 4.3.2 gives an overview and lists existing interfaces for sensors like camera or laser scanner. The entity trajectory cache is filled by the ESGM every time a new request was sent to the perception module. Each trajectory stands for the result of one perceptional computation, which is encapsulated into computational units (CU). The CUs are interfaces to algorithms like blob detection, basic image filtering, abstract affordance cue detection, etc. Each computational unit generates a predefined data structures in terms of entity frames used by other modules

(e.g. learning module, execution control). These single frames are integrated over time into entity trajectories and feed into the entity trajectory cache (ETC). Other modules interface the perception module using the following principles. First request for some perceptional features, afterwards read the data from the trajectory, and possibly change the parameter set used to configure the computational unit. The Entity Frame Monitor depicted in Figure 2 signals similar functionality compared to the event and execution monitor and may be integrated in the future in order to monitor single perceptional units over time.

4.2.2 Current state of implementation

The perception module provides information about the environment as well as status information about the robot. Due to limited resources in computational power and memory the robot has to focus on useful information for the given task. Modules, such as the learning module or the execution control module, can send a *request* to the perception module to initiate the information extraction. This request specifies a computational unit (CU), diverse parameters as well as sampling rate and cache size. The concept of computational units used in the perception module is simple. The CU takes an entity trajectory as input and transforms this data into another entity trajectory which is added to the ETC as a result of the computation. For example a CU performing color-blob detection takes an image as input and a list of blob descriptions is the result. One exception of the general concept are raw sensor data needed by the learning module. The input of those CU are no trajectories. The CU has direct access to the sensor via the sensor toolbox and bypass the raw sensor information into the output entity trajectory. More complex CUs are built from simpler ones, i.e. a chain of basic CUs gives a complex CU. The whole bundle of CUs available in the perception module is summarized as computational perception toolbox (CPT). Modules initiating perceptual computation via a request call are responsible to read the information calling the *read* method. Due to limited memory having many entity trajectories means smaller cache sizes for them. The perception module was not intended as a long-term memory. Such kind of data management has to be done within those modules, that needs historical data over a long time period (e.g. learning module). Figure 2 depicts a detailed view of the perception module.

Components The following components are implemented as computational units and integrated as part of the CPT.

- Color Blob Extraction: This CU can extract multiple color blobs out of the camera images.

- SIFT Features: This CU can extract a list of interest points and a description of the local neighborhood around that point. This was used in [6] to recognize affordances out of images.

Other computer vision algorithms, including methods to detect affordance cues, have already been developed and used within MACS, however, their implementation into the ESGM framework is on-going work.

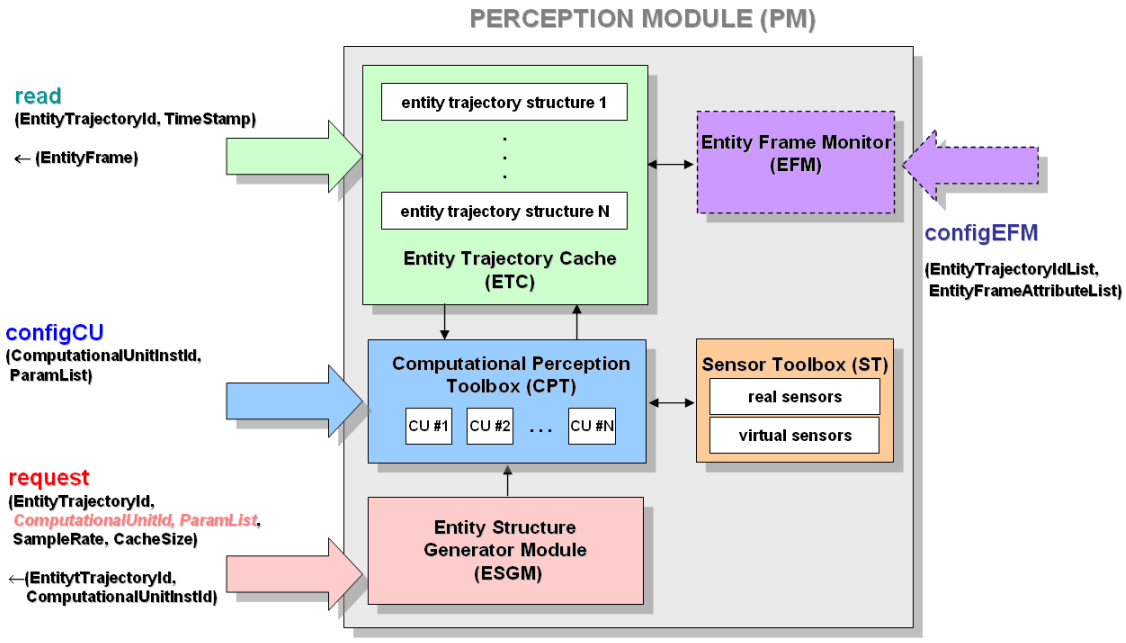

Figure 2: A detailed view of the perception module from an architecture perspective.

4.2.3 Interface specification

As an interface to the perception module the following mechanisms are proposed.

- ***request()***: The first parameter is an existing entity trajectory identifier to specify the input of the computational unit. In the special case of raw sensor reading this value is ignored. Each CU has a unique key which is given next in order to configure the type of computation (e.g., color blob detection or bypassing raw sensor data). Each CU is configured using the parameter list and the sample rate as well as the size of the data cache are given to complete the request. The request returns the identifier of the entity trajectory which contains the result of the computation and in addition a reference to the computational unit for a later parameter change.

- ***configCU()***: This mechanism allows the system to change the settings of a specific computational unit during runtime. The reference to the CU, returned by the request call, and the new parameter settings are needed to call this method.

- ***read()***: The perception module provides information about the environment and the state of the robot via real or virtual sensors. Other modules can access those data using this particular method. The identifier of the entity trajectory is needed to specify the data source. This identifier is returned by the previous request call. The second parameter is a time stamp in order to signal the position on the trajectory. As a result the entity frame is returned.

- ***configEFM()***: This method will actually be handled by the EEM (Sec. 4.5) and is only mentioned here for the purpose of demonstrability of PM based results.is not implemented.

4.2.4 Current state of integration with other modules

Processing of perceived data runs on different locations. Basic skills of the robot like obstacle avoidance need some fast and highly frequent processing of sensor readings. Those processes run directly on the robot in contrast to more complex calculations that can be performed by some server. See 8 for a more detailed discussion about that testing different configurations concerning sensor locations and data processing.

The agreed configuration between perception module and behavior module was an ESGM running at the server and various sensor interfaces running at the client side, i.e. directly on the robot. All other modules (e.g. learning module, execution control) can access raw and preprocessed data which are encapsulated into entity trajectories from the perception module. Computational units are the building blocks of the CPT as described above. They may have variable timing constrains or run times because they do not effect the robot's control loop. Initial timing experiments between perception and behavior module have shown that the delays in accessing the sensor readings are acceptable compared to different configuration. The camera interface is fully integrated in the system.

4.2.5 Testing of the software

The software has been tested off-line as well as in cooperation with the behavior module. Results are promising, however tests within the MACS scenario are currently prepared and will be preformed in the near future.

4.3 Sensors

4.3.1 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.

4.3.2 Current state of implementation

Components

- External sensors: stereo camera system, 3D laser scanner
- Internal (proprioceptive) sensors: wheel encoder
- Virtual sensors: odometry (robot pose and speed), orientation into free space, distance to next obstacle, virtual 2D obstacle map, activation values of behaviors

All real sensors are implemented on both the hardware and simulation level. Device handlers have been implemented that can either access the sensors (and actuators) directly (if the real robot is used) or by means of the CORBA sensor interfaces [7]. The sensor readings are integrated with the *Perception module* since they implement the *Perception module*'s SensorToolbox. They are being accessed in pulling mode. That is, the *Perception*

module samples all sensors at fixed intervals by accessing them through CORBA. The benefit of accessing the sensors in pulling rather than in pushing mode is that first, the robot does not have to take care of the additional timing that is being handled implicitly by the *Perception module* and second, the different perception algorithms have complete control over the desired sensor sampling rates. A more complex sensor fusion algorithm that takes longer execution time might use a slower sampling rate than a for instance a Kalman filter integrating several sources for pose estimations.

This sensor integration is currently implemented for the stereo vision system as well as for the virtual senors of the *Behavior module*. All other sensors will follow shortly.

Sampling the real sensors delivers raw data that can be accessed e.g. by the perception filters from the *Perception module*. The access can be realized in a subscription mode that notifies the subscribers each time a new sensor reading arrives. The output of the virtual sensors is based on pre-processed sensor data (e.g. odometry) or reflects the current system state (e.g. behavior activation values).

4.3.3 Testing of the software

Accessing the real sensors has been successfully tested directly or through CORBA both to the simulator and the real robot. The interfacing with the *Perception module* has as well been tested successfully. It showed, that the transmission of the camera data as raw data streams demands the *Perception module* to be executed on a separate computer that does not run the main robot control loop. This is due to timing constraints that arise in a high system load depriving the needed CPU time for processing from the *Perception module*. The decision to run the *Perception module* on a separate computer allows furthermore to apply more complex perception algorithms on the sensor readings without interfering with the actual robot control.

4.4 Entity Structure Generation Module (ESGM)

4.4.1 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.

4.4.2 Current state of implementation

The Entity Structure Generation Module (ESGM) module is fully integrated into the MACS architecture and used by other modules (e.g. perception module 4.2). It has been described in great detail in deliverable D4.4.1 and D4.3.2 [4; 5].

4.4.3 Interface specification

The interface specification is fully described in [4; 5].

4.4.4 Current state of integration with other modules

The perception module has already integrated the ESGM, see Section 4.2 for further details.

4.5 Event and Execution Monitor (EEM)

4.5.1 General description

The event and execution monitor serves two purposes. First, it can be used to recognize the occurrence of *events*. 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.

EEM can be also used to monitor the execution of actions and/or behaviors. 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.

4.5.2 Current state of implementation

As described in Deliverable D4.4.1 [5] (A software prototype for an affordance monitoring module with empirical testing using various MACS robotics platforms - The Event and Execution Module), EEM is implemented using a knowledge processing middle-ware called Dyknow.

4.5.3 Interface specification

EEM supports mainly two types of requests, namely *i*) cue or outcome event monitoring request, and *ii*) behavior execution monitoring requests. It in turn sends *i*) event occurance, and *ii*) execution success/failure signals back to the original caller.

In EEM, cue and outcome monitoring is provided through monitoring the events that are represented as Simple Temporal Networks (STNs). In [5], how event monitoring is implemented is described in Section 4.2. Details of the interface functions `monitor_event(), stop_monitor_event()` are provided in this section. In the tutorial part of the same document, some example entity structures are created and monitored. Additionally, an example STN is formed, where a zig-zag behavior is detected on a robot which drives towards a target.

Behavior execution monitoring is requested from EEM through metric interval temporal logic formulas (MITL). In Chapter 3, several example MITL formulas are provided and the means of evaluation of these formulas through a technique called progression is given. In Section 4.1, the implementation of execution monitoring is provided, with the details of the interface functions like `monitor_execution() , stop_monitor_execution()` .

4.5.4 Current state of integration with other modules

The interface of the EEM module is specified but the integration with other modules is currently missing. EEM plays a key role between the *Execution Control* module and the *Perception* module, which makes the integration of EEM with these modules a task that should be accomplished in near future.

4.5.5 Testing of the software

The EEM module is tested in a scenario, where an object following mobile robot is monitored through event and execution monitors. The details can be found in [5].

4.6 User Interface module

4.6.1 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.

4.6.2 Current state of implementation

In the current state of the *User Interface* module, only a joystick interface is implemented. Using the joystick interface, one can drive the robot, take 3D scan images from the laser scanner, and capture images from the cameras. The joystick interface was demonstrated in the second MACS review meeting, controlling both the real KURT3D robot, and the simulated robot inside MACSim.

4.6.3 Interface specification

The joystick interface is specified and implemented. The specification of other functionalities of the *User Interface* module is a task waiting to be done.

4.6.4 Testing of the software

The testing of the implemented joystick interface is done successfully, and demonstrated in the second MACS review meeting.

4.7 Learning module

4.7.1 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 [8] and D5.3.2 [9]) and stores the derived affordance representations in the *Affordance Representation Repository*.

The learning architecture is described in deliverable D5.3.2 [9] and the state of implementation is described in detail in deliverable D5.3.3. This section gives a brief introduction to the implementation. Please see deliverable D5.3.3 for a more detailed description.

4.7.2 Current state of implementation

As described in deliverable D5.3.2 [9], the learning module contains several sub modules which use a set of different data structures. The modules are implemented in Java and connected to the components of the affordance based control architecture using the middleware CORBA. The *Learning Module* processes data read from the *Perception Module*. To be able to easily add and change data types in the affordance based control architecture, the learning procedure is implemented type independent. The only restriction is, that for all data types there must be at least one `Comparator` in the *Learning Module* to be able to compare two trajectories, consisting of data of the concerning data type and thus to be able to cluster these trajectories. To be able to store the trajectories, constructors and writers are additionally to be implemented. Currently `Comparator`s and concerning constructors and writers are implemented for the basic data types `Double`, `Integer`, `Boolean`, `Index` and for the complex type `Action` (which contains the status of an action: running, triggered, and paused).

The data structures implemented to store the data needed for learning are:

- `GenericTrajectory`$< T >$, is a generic container storing time stamped data of type `T`
- `TrajectorySet`, stores `GenericTrajectory`s which were recorded simultaneously during the application of a behavior.
- `ApplicationSpace`, is associated to a behavior of the agent and stores the `TrajectorySet`s that were recorded during the application of the associated behavior. In addition an `ApplicationSpace` stores the `Partitioner`, the `ChannelExtractor` and the `EventCharacteriser`s derived from the stored data.

The main modules currently implemented as a distributed system are:

- `Application Spaces Module`, a data base that stores the behavior related `ApplicationSpaces`, which contain the data that is to be used for learning.
- `Partitioning Module`, creates `Partitioner`s for `ApplicationSpace`s.
- `Relevant Sensor Channel Extraction Module`, creates `ChannelExtractor`s for the pre- and post-application phase of the data in the partitions of the associated `ApplicationSpace` (for cue-event and outcome-event detection).
- `Event Characterisation Module`, creates `EventCharacteriser`s for the relevant sensor channels, extracted by the `ChannelExtractor`s (cue- and outcome-related characteristics).

- `Learning Control`, controls the data- and control-flow within the *Learning Module* and controls the communication with the *Perception Module*, the *Affordance Representations Repository* and the *Deliberation Module*.

4.7.3 Interface specification

As shown in figure 3 the `LearningControlModule` is the central point in the learning architecture that manages the incoming and outgoing data of the *Learning Module*. The `LearningControlModule` is connected to the *Perception Module*, the *Affordance Representations Repository* and the *Deliberation Module*.

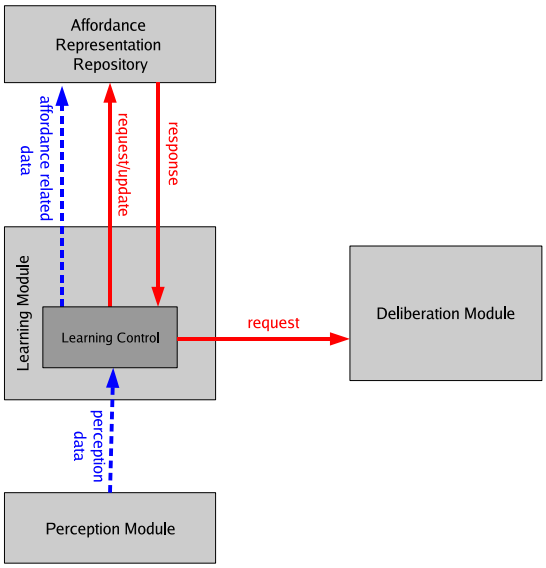

Figure 3: Embedding of the affordance learning architecture into the affordance based control architecture as described in deliverable D5.3.3.

For the connection to the *Perception Module* the `LearningControlModule` contains the sub-module `BehaviourMonitoringModule` that is responsible for the incoming data from the perceptual system. The `BehaviourMonitoringModule` monitors the state of the agents perceptual system and is responsible for storing the recorded `TrajectorySets` with the appropriate length (including pre- and post-application phase, see Deliverable D5.3.2 and D5.3.1) into the `ApplicationSpaces` stored in the *Application Spaces Module*.

The `BehaviourMonitoringModule` is to be adapted, when the underlying perceptual or behavioral system changes (e.g. new types of sensor data) or is replaced. The data is pulled from the *Perception Module*. Thus the interface of the *Perception Module* and the data types provided by that module, must be known to adapt the `BehaviourMonitoringModule` to the used interfaces and data structures. When the perception system changes and new data types are introduced, new `Comparators`, `TrajectoryWriters` and `TrajectoryConstructors` are to be implemented, but the learning procedure itself does not need to be changed.

The affordance knowledge acquired by the learning process must be stored to be used by the *Execution Module* and the *Deliberation Module*. Therefore the affordance based control architecture provides the *Affordance Representation Repository* (*ARR*). As described in

section 2.3 in deliverable D5.3.2. the ARR stores $n : m : l$ relations between cue events, behaviors and outcome events. From this data, affordance triples can be derived either within the *ARR* or within the execution or deliberation part of the control architecture.

The *Learning Module* is designed to push the derived data into the *Affordance Representation Repository*. As shown in figure 3 the `LearningControlModule` is responsible for pushing the data into the repository. The data structures used in the *ARR* depend on what the deliberation and execution part of the overall affordance based control architecture require. The current state of implementation of the *Learning Module* provides trajectory prototypes and the method described in section 4.2.3 of deliverable D5.3.2 (characterisation of the trajectories using neural networks).

The connection to the *Deliberation Module* is designed to request the execution of certain behaviors in order to acquire data for the concerning `ApplicationSpace`. This request is currently under implementation and will, depending on the development of the *Deliberation Module*, be finished for the second version of D5.3.2 which is due in July 2007.

4.7.4 Testing of the software

The learning architecture is tested with test-*ApplicationSpaces* which are provided together with the implementation. The *Learning Module* is connected to the Kurt2 robot, as demonstrated at the second MACS review meeting.

4.8 Affordance Representation Repository

4.8.1 General description

The Affordance Representation Repository stores affordance representations of the form (`<cue>`,`<behavior>`,`<outcome>`) generated through manual design or through learning. The Repository has basic data base capabilities for managing the representations.

4.8.2 Current state of implementation

The data type of the information stored in the *Affordance Representation Repository* depends on the information needed by the *Deliberation Module* and the *Execution Module* and the information provided by the *Learning Module*. The current implementation of the *Learning Module* provides prototypical trajectories of cue-events and of outcome-events and optional a characterisation of the trajectories using neural networks (see deliverable D5.3.2). The data type describing the behavior depends on the information provided by the *Behavior Module*. The implementation of interrelations of the cue-events, the outcome-events and the behaviors is currently under discussion between the MACS project partners.

4.9 Deliberation module

The Deliberation Module of the MACS architecture is being developed by the University of Osnabruck, who joined the MACS consortium recently. The progress in this module will be reported in later revisions of this document.

4.10 Execution control

4.10.1 General description

This section describes the Execution Control(EC) module as it is implemented in the MACS architecture. The Execution Control module's basic functionality can be described as taking the (<cue>,<behavior>,<outcome>) triples as subgoals from the Deliberation Module and trying to achieve the specified <outcome> by executing the specified <behavior>. But before executing the <behavior> it makes sure that the <cue>s, which imply that the <behavior> results in the intended outcome, are available. Once the <cue> is detected by the EEM, the corresponding <behavior> in the Behavior Module is triggered. EC module also gets possible exception signals from the Behavior module and passes such exceptions to the Deliberation module so that it re-evaluates the plan. Another functionality of the Execution Control is to trigger active perception behaviors, when a specific cue is needed in the environment. For this purpose, the Execution Control module mainly interacts with the Event and Execution Monitor (EEM), the Behavior Module (BM), and the Deliberation Module (DM). There is also the special case of User Interface (UI), which bypasses the normal execution cycle (for instance, the control of the robot via a joystick). The UI interface is also used to give status feedback to the human controller. The module and its interfaces with other modules are demonstrated in Figure 4.

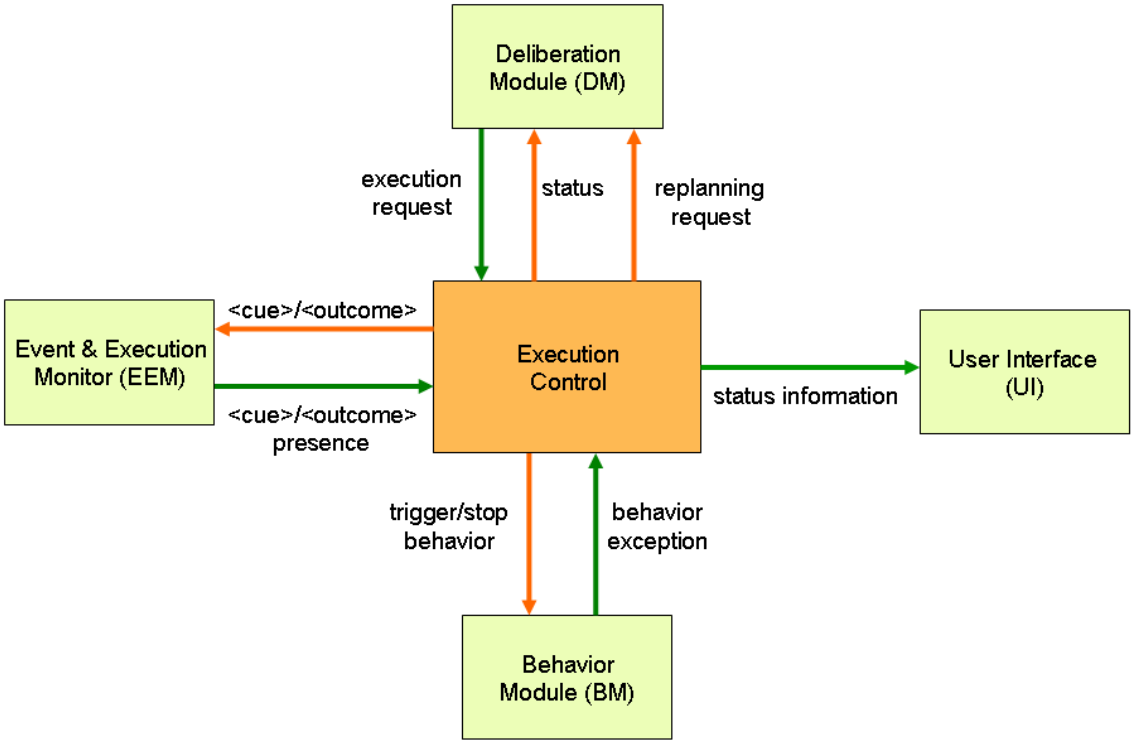

Figure 4: The interface of Execution Control with other modules.

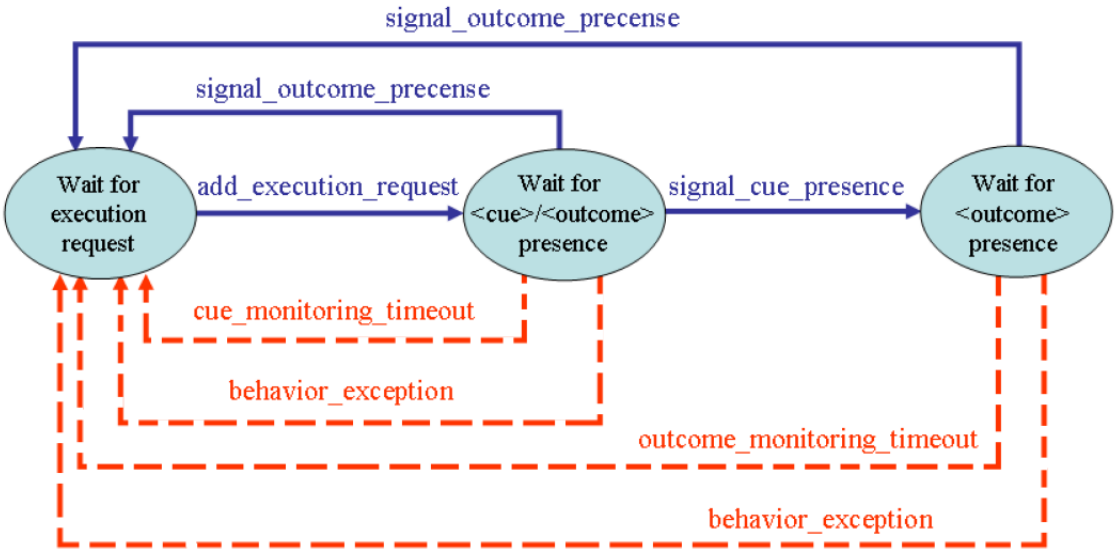

Figure 5: The solid blue lines correspond to the normal execution flow. The dashed red lines represent emergency/failure situations.

The Execution Control will be an interrupt-driven module, ready to respond requests from the planner, exceptions from the behaviors, and cue/outcome presence signals from the EEM, asynchronously. The description found below rests on this interrupt-driven approach. We identify the possible interrupts/requests that Execution Control may receive, and specify what the EC needs to do in each of these situations.

First, we list each of these possible interrupts to the EC, along with a description of what they mean, and what they result in, in terms of the external and internal actions EC takes. Note that these interrupts also constitute the interface of the EC module to the other modules. A summary of this is also supplied in a table, where the function name, the actions taken, and its effect on internal state of the EC are shown explicitly.

We also present a state transition diagram (Figure 5), on which we show the effect of each interrupt on the state of the EC.

4.10.2 Current state of implementation

The functions that are described below are currently implemented in C++. In the next step, CORBA interfaces will be decided with other partners, and integration with other modules will be done.

4.10.3 Interface specification

In this section, we will give the detailed description of the functions that are implemented in this EC module.

***add_execution_request* ():** This function is called by the Deliberation module (Scheduler) in order to pass the subtask that needs to be executed. There is no precondition for

calling this function, thus Deliberation module has complete control over the Execution Control module. It can add, delete and overwrite the subtasks anytime. As described Section 6.8.6 in [1] , the subtasks are identified by the behaviors that are executed together with the corresponding cue and outcome descriptors. Thus, this function is called with a (`<cue>`, `<behavior>`, `<outcome>`) triple argument. Additionally, it is desired to add multiple (`<cue>`, `<behavior>`, `<outcome>`) triples by the Deliberation module, and multiple `<cue>`s could be detected simultaneously by EEM. Thus, the deliberation module should assign priorities to the corresponding triples while calling this function. Following, we will give a brief description of this function:

1. The behaviors are only afforded if the corresponding cues are found in the environment. Thus, EEM module is configured to detect `<cue>`.

2. In order to search for the corresponding cues, the *active perception behaviors* are triggered.

3. The robot keeps an internal `<cue>` timer ($t_{\texttt{cue}}$), that is started in this stage. The immediate environment may not offer the affordances to execute the subtask, or the robot may be unsuccessful in searching for the cues. In either situation, after a certain amount of time, the robot should stop searching for the cues.

4. Together with each behavior, an (expected) outcome descriptor `<outcome>` is provided from Deliberation Thus, in order to find out if the execution of the subtask is successful or not, the outcome of the behavior should be monitored and should be compared with the expected outcome. In some cases, without executing any behavior, the expected outcome may exist in the environment. In such cases, there is no need to look for the cue descriptors and trigger corresponding behaviors. Thus, we will configure EEM module to detect `<outcome>`, independent of the execution of the behaviors.

***remove_execution_request* ():** This function is called by the Deliberation module (Scheduler), in order to delete any subtask that was assigned to the Execution Control. If the `<cue>` has not been detected yet, Execution Control stop the *active perception behaviors*, reset $t_{\texttt{cue}}$, send stop signal to EEM module which monitors the corresponding `<cue>` and `<outcome>`. If the `<cue>` was found, and the corresponding `<behavior>` is running, Execution Control stop the triggered behavior, send stop signal to EEM module which monitors `<outcome>`, and reset the internal `<outcome>` timer.

***signal_cue_presence* ():** This function is called by EEM, when the `<cue>` is detected. EEM was configured in function **Add subtask** to monitor `<outcome>`.

1. The *active perception behaviors* that are executed to search for the cue is stopped.

2. Since the cue is detected, monitoring of that cue should be stopped. Thus, a "stop" signal is sent to EEM for that particular `<cue>`.

3. Since the cue is detected, the `<behavior>` is executed by sending the appropriate signal to the Behavior Module.

4. The robot keeps an internal `<outcome>` timer ($t_{\texttt{outcome}}$), that is started in this stage. When the expected outcome is not detected within a certain amount of time, execution of the behavior and this subtask should be stopped.

***signal_outcome_presence* ():** This function is called by EEM, when the `<outcome>` is detected. Detection of the `<outcome>` means that the subtask is executed successfully, thus Deliberation module should be informed. Additionally, all processes relating to this subtask should be stopped, and all variables should be reset.

1. The Deliberation module is informed by sending "success" signal.
2. Since `<outcome>` detected, a signal is sent to EEM to stop monitoring the outcome.
3. If EEM monitors `<cue>`, a signal is sent to EEM to stop monitoring the cue. Such a situation may occur, if the `<outcome>` is obtained without executing any behavior, which means that the expected outcome already exists in the environment.
4. A stop signal is sent to Behavior module for the executing `<behavior>`. Additionally, all *active perception behaviors* are stopped if they are active.

***behavior_exception* ():** This function is called from Behavior module, whenever it encounters with a problem during the execution of the behaviors. Additionally, Execution control is informed about the emergency situations via this function. Depending on the nature of the problem, appropriate signals are sent to Deliberation and/or User Interface modules.

If the expected outcome is not detected within a certain time after triggering the corresponding behavior, the subtask is labelled as failure. Thus, the Deliberation module should be informed and all processes relating to this subtask should be stopped.

1. The Deliberation module is informed by sending "fail" signal.
2. A signal is sent to EEM to stop monitoring the outcome.
3. A signal is sent to Behavior module to stop the `<behavior>`.

***cue_monitor_timeout* ():** If the cue is not detected within a certain time after triggering the *active perception behaviors*, the subtask is labelled as failure. Thus, the Deliberation module should be informed and all processes relating to this subtask should be stopped.

1. The Deliberation module is informed by sending "fail" signal.
2. A signal is sent to EEM to stop monitoring the cue.
3. A signal is sent to EEM to stop monitoring the outcome.
4. A signal is sent to Behavior module to stop the *active perception behaviors*.

4.11 Low Level Behavior System

4.11.1 General description

The *Behavior Module* of the architecture can be divided into two major submodules: the *low-level behavior system* and the *high-level and affordance-related behavior system*. We present the state of implementation of these two submodules separately; the low-level behaviors system in this section, and the high-level behavior system, in the next.

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, however, 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.

4.11.2 Current state of implementation

Behaviors

- **Steer Behavior** The steering behavior performs on the input of the virtual sensor reading that reflects an orientation into free space. The robot's current driving direction is being adjusted to reflect that orientation.

- **Brake Behavior** The braking behavior performs mainly on the input of the virtual sensor reading of the distance to a nearby obstacle. That information is being gained from the virtual 2D obstacle map. The behavior applies a virtual roadway approach that will slow the robot down if the obstacle is closer than a certain threshold that is adaptive to the robot's current speed.

- **Turn Behavior** The turning behavior works on the same information as the *Brake* behavior as well as on the activation value of the *Brake* behavior. If the robot came to a full stop, i.e. the *Brake* behavior has an activity of 100%, the robot will turn on the spot into the preferred direction of free space. It will stop after it recognizes that it's virtual roadway is again free.

4.11.3 Interface specification

The interfaces of the low-level behaviors are not fully specified yet. But some of the basic interfaces are already available although they still need to be connected to other components of the architecture through CORBA interfaces.

- Trigger/stop low-level behavior: The behaviors are active all the time, as long as they are not set to be inactive by the Execution control or other behaviors. The interface is not fully implemented yet but will contain a setActive(double) method. Valid values are either 0 or 1.

- Feedback as virtual sensor: The behaviors provide a feedback that tells both their current activation value and whether they are currently meant to be active or not.

- Input from Perception Module: The behaviors have direct access to the sensor readings as they apply this tight action-perception coupling. There are no specified interfaces for this.

4.11.4 Testing of the software

All low-level behaviors have been tested successfully on their own and in combination. A video of these tests can be found on
`http://www.informatik.uni-osnabrueck.de/~cloerken/videos.html`

4.12 High-level and affordance related behaviors

4.12.1 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. Furthermore 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 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 successful 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 grade of completion as a virtual sensor reading.

4.12.2 Current state of implementation

Behaviors

- **Approach** The approach action approaches a specified robot pose. It can be used with leaving the low-level behaviors activated resulting in a target-directed obstacle avoidance behavior.

- **Push** The push action is the most basic object manipulation action. The robot drives forward and tries to push anything in front of it. If the robot recognizes that it needs to much force for driving, the action aborts and the robot reverses.

- **Explore** Exploration is a subtask that has to be triggered by the Execution Module. When in this mode, the robot drives around in the arena and tries out its set of action on interesting items.

4.12.3 Testing of the software

Approach, *Push*, and *Explore* have all been tested successfully. Videos of these tests can be found on
`http://www.informatik.uni-osnabrueck.de/~cloerken/videos.html`

4.13 Actuators

4.13.1 General description

The robot's actuators are solely accessed by the behavior system. Higher level components have no direct access.

4.13.2 Current state of implementation

The implementation is very similar to that of the robot sensors. Device handlers have been implemented to access the actuators either directly or through CORBA on the real robot and the simulator.

- Drive motors: Firmware is implemented and fully functional.
- Crane actuators: A first firmware version exists as well as the CORBA IDL specification.
- Servo motors for sensors: Direct access is implemented, the CORBA access is not ready yet.

4.13.3 Interface specification

The interface specification is described in deliverable D1.1.2[7].

4.13.4 Current state of integration with other modules

The actuator implementation is integrated with the behavior system that is the superordinate architectural component as well as with the physical and simulated actuators.

4.13.5 Testing of the software

The available actuators have successfully been tested. An exception is the still not available crane. The servo drive integration to the simulator is currently being developed.

References

[1] 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, and Christopher Lörken. Development of an affordance-based control architecture. Deliverable MACS/2/2.2 v1, Fraunhofer Institut für Autonome Intelligente Systeme, Sankt Augustin, Germany, 2006.

[2] Ralph Breithaupt, Rainer Worst, and Hartmut Surmann. Physical robot demonstrator and scenario. Deliverable MACS/6/3.1 v4, Fraunhofer Institut für Autonome Intelligente Systeme, Sankt Augustin, Germany, 2006.

[3] Rainer Worst, Hartmut Surmann, and Martin Hülse. Integrated implementation of reference control system. Deliverable MACS/1/3.1 v1, Fraunhofer Institut für Autonome Intelligente Systeme, Sankt Augustin, Germany, 2006.

[4] Fredrik Heintz, Patrick Doherty, Björn Wingman, Piotr Rudol, and Mariusz Wzorek. A software prototype for affordance support - the entity structure generation module (esgm). Deliverable MACS/4/3.2 v2, Linköpings Universitet, IDA Group, Linköping, Sweden, 2006.

[5] Fredrik Heintz, Patrick Doherty, Björn Wingman, Piotr Rudol, and Mariusz Wzorek. A software prototype for an affordance monitoring module with empirical testing using various macs robotics platforms - the event and execution monitor module (eem). Deliverable MACS/4/4.1 v2, Linköpings Universitet, IDA Group, Linköping, Sweden, 2006.

[6] 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.

[7] 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.

[8] 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.

[9] Georg Dorffner, Jörg Irran, Florian Kintzler, Lucas Paletta, and Gerald Fritz. Robotic learning architecture capable of autonomously segment action sequences into affordances. Deliverable MACS/5/3.2 v1, österreichische Studiengesellschaft für Kybernetik (OFAI), Vienna, Austria, 2006.