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

D4.4.3 An evaluation of the MACS planning module in the context of the MACS architecture

Due date of deliverable: August 31, 2007
Actual submission date v1: October 15, 2007
Actual submission date v2: January 14, 2008

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

University of Osnabrück (UOS)

Revision: Version 2

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 D4.4.3

An evaluation of the MACS planning module in the context of the MACS architecture

Christopher Lörken, Frank Meyer, Andreas Bartel, Jens Poppenberg, Joachim Hertzberg

Number: **MACS/4/4.3**
WP: *4.4*
Status: *final*
Created at: *September 20, 2007*
Revised at: *v2 – January 14, 2008*
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:

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

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 describes the means for assessing and evaluating the value of the planning system introduced in [1] in the context of the MACS robot control architecture. It will furthermore describe the experiments that have been made and present their results.

The following section will shortly describe on the one hand the benefits that arise from utilizing a planner for the overall capability of the system with respect to the affordance concept and, on the other hand, the beneficial influence of affordances on a planner. Section 3 will then describe the different scenarios and use cases that have been selected to demonstrate and evaluate the planning system as part of the affordance architecture. Section 4 will describe the work that has been conducted at UOS in order to create the complete planning module and to generate this evaluation deliverable whereas the reminder of this document will present and discuss the results of the experiments.

2 Planning with Affordances

2.1 Advantages for the Overall System

Basically, affordances are opportunities the environment offers to an agent and the agent can act on those opportunities. For merely demonstrating the usability of the affordance concept in robotic applications one could thus argue that it is sufficient to employ a purely reactive system that demonstrates its learned understanding of affordances by applying appropriate actions on environmental stimuli or test objects that afford that specific action.

However, following a line of argument that we have partially already presented in [2], we are of the opinion that it makes indeed sense to reason about affordances instead of acting directly upon an affordance percept. This point has as well been picked up by [3] who explicitly argue that an agent does not merely respond to a directly perceived stimulus by applying the action that is afforded in that situation. It is not controlled by its environment. It can rather use the information provided by the affordances of a situation and reason about them in a goal-directed manner selecting those afforded actions that will lead to its goal. Speaking in terms of the MACS demonstrator scenario, the robot can neglect the perceived pushability affordance of one object if it wants to open the door by putting something on the switch in order to get to the other room.

Such a level of complexity of action control is not possible in a purely reactive system as the agent would basically only open the door by chance when it selects exactly the trigger-switch affordance out of its pool of perceived affordances to act upon. In other words, as soon as the agent is meant to act in a goal-directed way the usage of a planning system that develops the necessary action sequences becomes inevitable.

We are furthermore of the opinion that it is necessary in the MACS project to integrate planner-based and thus goal-directed purposeful task execution in order to demonstrate not only that affordances *can* be used in a robotic control architecture, but in addition that it is supportive, meaningful, and advisable to integrate the affordance concept as a first-class citizen.

For example, an approach in which a robot avoids an obstacle because the robot perceives the corresponding affordance cannot be differentiated objectively from standard potential field approaches that have been manifoldly demonstrated in the past. We thus argue that this demands a focus on what capabilities environmental objects afford for *interaction* rather than merely for *reaction*.

Especially tasks like purposeful object manipulation require a different level of understanding by the robot. They thus qualify better for demonstrating deliberation, understanding, and usage of affordances within an architecture. If a robot is able to perceive the functionalities afforded by objects, or in other words the concrete actions it can perform on them, and if it is able to evaluate, or plan, on that perception and its knowledge, it will in the end be able to demonstrate robust and successful affordance-based task execution. For example, the robot will select objects to interact with according to their afforded functionality rather because it recognizes them as belonging to a certain labelled category, i.e. it will put any liftable object on a switch to open a door instead of explicitly searching for the one object that is labelled door-opening-weight.

The evaluation of the planning system will thus not deal with reactive tasks like affordance-based reactive navigation as that has already been shown in the MACS project in [4]. It will instead focus on higher-level tasks like goal-directed object manipulation.

2.2 Advantages for the Planning Module

As aforementioned, employing a planning module in the MACS robot control architecture brings along a very interesting advantage for the planner system itself since it shows a methodology for the difficult question of *operator grounding* (see as well [1]). The basic idea behind this is that a plan normally consists of a sequence of operators; e.g. a sequence for triggering the switch with an object labelled box_A:

$$< lift(box_A),\ carry(box_A, switchRegion),\ trigger\text{-}switch(box_A) >$$

However, it is not always necessary to actually anchor, or ground, all objects of the plan in the world. In the example just provided, it actually does not matter if we execute the planning branch containing the operator $trigger\text{-}switch(box_A, switchRegion)$ or another branch with $trigger\text{-}switch(box_B, switchRegion)$.

So sometimes, it is not important with which particular object or tool an action is being accomplished or, speaking in planning terms, on what environmental object a plan operator is actually being grounded. So why should the planner care about it? The necessary plan step is just to open the door by putting *any suitable item* on the switch and no one cares which item that actually is. Such a suitable item is, of course, exactly that environmental object that affords the corresponding action to the agent. So by formalizing the operators in an affordance-based notion[1] we have actually gained the possibility to formulate an operator to lift anything $liftable$ instead of explicitly lifting box_A. This makes it possible to ground the same $lift$-operator either on box_A or on box_B or even on $cylinder_{42}$. It is left to the execution phase of the system to select the next appropriate object to ground the current plan operator on. So when the robot is meant to lift something it will select the next object in its environment that affords to be $liftable$.

[1] See [1] for the detailed description.

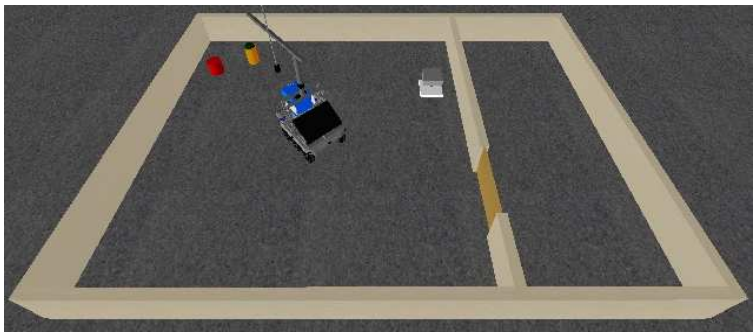

Figure 1: The MACS demonstrator scenario. Screenshot from MACSim.

3 Evaluation Setup

As an evaluation of the planning module and its mutual benefit with the affordance architecture we will present an experiment in which the robot performs what has been introduced in the last section. It will thus perform a goal-oriented object manipulation task and use an on-the-fly grounding mechanism for its operators. Since we want to demonstrate capabilities that are objectively distinguishable from what a merely reactive system can perform, we chose to use the task of operating the switch to open the door.

In all different experiments, the robot will be located in the left room of the MACS demonstrator scenario (cf. Fig. 1 and 2). The planner will work either on a predefined world model or after an exploration phase and will present the necessary operator sequence to acquire the goal of the task. The execution module will instantiate the different operators and will select appropriate environmental objects to ground them based on the content of the affordance representation repository (see again [1] for the full description of the work flow).

We define the following tasks for assessing the capability of the system according to the standards defined in [5]:

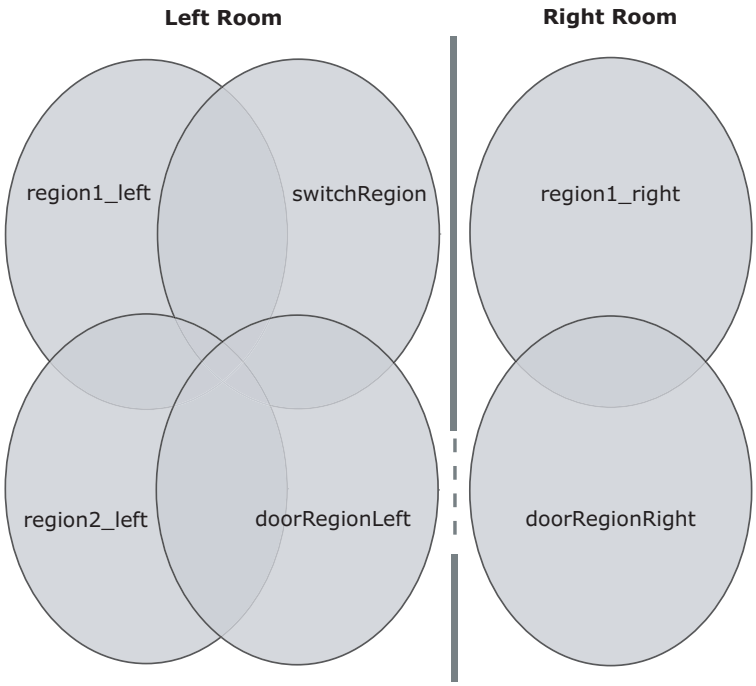

Figure 2: World Representation Map of the Planning Module.

3.1 Task 1 - Lift object

Name:	PL_Lift	Category:	combined	Level:	1
Scene Description:	• The robot is located in $region2\_left$. • There is exactly one test object in another region that can be lifted with the crane.				
Essential Affordances:	Liftability				
Task Description:	The robot has to drive to the correct region and lift the test object.				
Expected Performance:	See description.				

3.2 Task 2 - Lift the liftable object

Name:	PL_SelectAndLift	Category:	combined	Level:	2
Scene Description:	• The robot is located in $region2\_left$. • There is at least one test object in another region that can be lifted with the crane. • There is at least one test object in another region that cannot be lifted with the crane.				
Essential Affordances:	Liftability				
Task Description:	The robot has to select the (or a) test object that affords lifting and lift it.				
Expected Performance:	• The robot selects the appropriate item and thus grounds the operator on a correct test object. • When there are multiple liftable test objects, the robot may select any of them.				

3.3 Task 3 - Open the door

Name:	PL_OpenDoor	Category:	combined	Level:	3
Scene Description:	• The robot is located in $region2\_left$. • There is exactly one test object in another region that can be used to trigger the door opening switch.				
Essential Affordances:	Liftability, switch-triggerable				
Task Description:	The robot has to place the test object on the switch in order to open the door.				
Expected Performance:	See description.				

3.4 Task 4 - Open the door with the right test object

Name:	PL_SelectAndOpen	Category:	combined	Level:	4
Scene Description:	• The robot is located in $region2\_left$. • There is at least one test object in another region that can be used to trigger the switch. • There is at least one test object in another region that cannot be used to trigger the switch.				
Essential Affordances:	Liftability, switch-triggerable				
Task Description:	The robot has to select the appropriate test object to trigger the switch.				
Expected Performance:	• The robot selects the appropriate item and thus grounds the operator on a correct test object. • When there are multiple suitable test objects, the robot may select any of them.				

3.5 Task 5 - Change rooms

Name:	PL_ChangeRooms	Category:	combined	Level:	5
Scene Description:	• The robot is located in $region2\_left$. • There are multiple test objects that either can or cannot be used to trigger the switch.				
Essential Affordances:	Liftability, switch-triggerable, passable				
Task Description:	The robot has to get into the other room. Therefore, it has to select the appropriate test object to trigger the switch and open the door. Afterwards it has to use the passable-affordance to drive through the open door into the second room.				
Expected Performance:	• The robot selects the appropriate item to trigger the switch and thus grounds the operator on a correct test object. • When there are multiple suitable test objects, the robot may select any of them. • The robot drives to the $doorRegionLeft$ to use the passable-affordance and to drive through the door.				

4 System Description and Implementation

The system that has been implemented by UOS during the last months consists of the following modules:

- Planning Module,
- Graphical User Interface for the Planning Module,
- Execution Control Module,
- Affordance detection units,
- Behavior System.

These components will now be described.

4.1 Planning Module

The planning module has been implemented in version 1 as it has been defined in [1]. The design and implementation of version 2 was, as expected, not possible due to the limited time constraints. The system is based on a PDDL domain description [6] that is completely listed in appendix A and explained in detail in [1]. The system uses a slightly customized version of FF-planner [7]. A plan consits of a sequence of operators that are represented as structs containing the name of the operator, the label of a connected affordace (if any), and a parameter list for executing the operator.

Representation of both the world model and the goal state is fully integrated in the GUI and can be transmitted to the planning system via Corba. For listings of the interfaces, refer to appendix B. The interfaces and the workflow between all components are based on the internal technical report of the bilateral meeting between METU-KOVAN and UOS in Ankara in July 2007.

4.2 Graphical User Interface

The graphical user interface of the planning module has been implemented using the Java programming language. The system contains the possibility to connect to the different Corba servers (of execution control and planning module) and to accept connections from both modules for feedback and communication reasons. The main functionality of the system lies in the complete integration of modelling tools to generate world models (figure 3) and goal requests (figure 4) for the planning module.

As schematically shown in figure 2, the GUI represents the different regions of the world. On the right of figure 3 and 4, the symbol list of represented facts and affordances is given.[2] The depicted symbols can be placed by drag & drop in the different regions and the world model can be transferred via the aforementioned Corba interface to the planning module.

The representation of the goal state, as depicted in figure 4, works just like the representation of the world model. When having completed a goal description, the GUI can

[2]Please note here that, in contrast to the examples of [1], the demonstration setup of the planner does not distinguish between items that are liftable and can trigger the switch and those that are as well liftable but cannot trigger the switch. We deem this as a bearable limitation for a proof of concept.

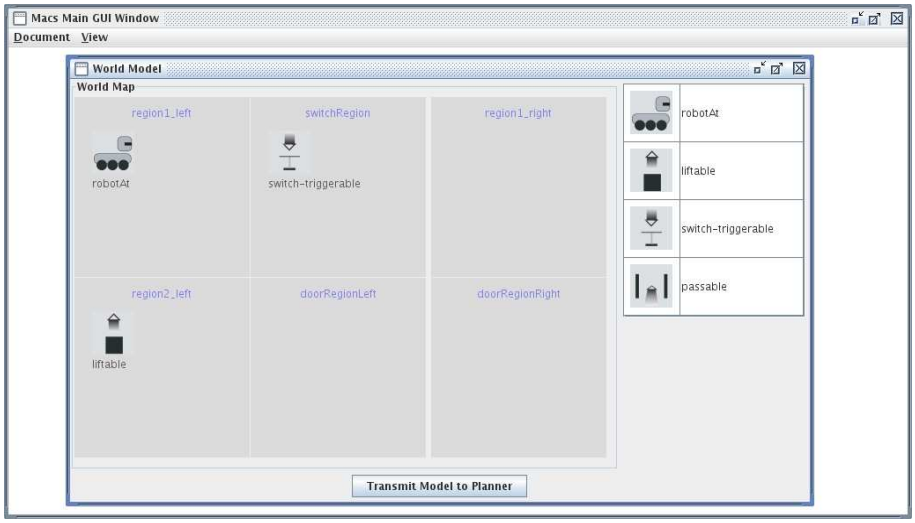

Figure 3: Screenshot of the UOS planer GUI. Depicted is the generation of the world model for the planner.

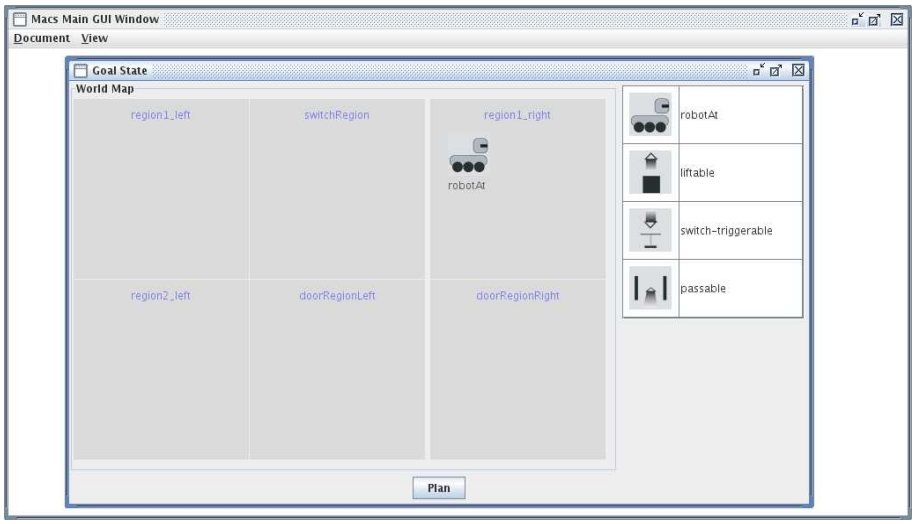

Figure 4: Screenshot of the UOS planer GUI. Depicted is the generation of the goal state for the planner.

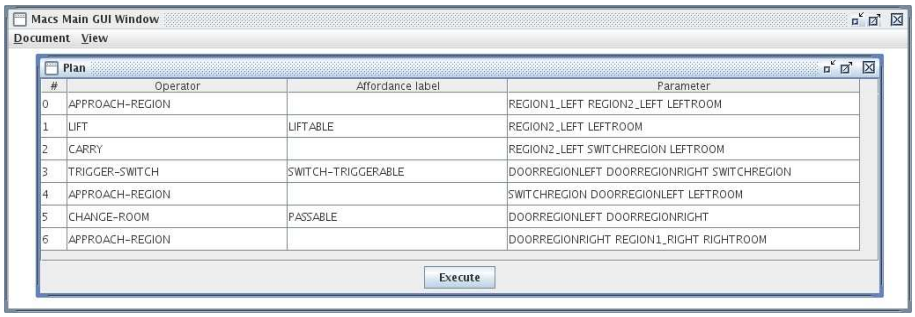

Figure 5: Screenshot of the UOS planer GUI. Depicted is a plan that has been generated by the planning module and is ready to be send to the execution module.

send the different goal conditions to the planner and trigger the planning process. The planning module will generate a fitting plan and return the result to the GUI.

Figure 5 exemplarily depicts such a plan. The user can then accept the plan and transmit it to the execution module or discard it and alter goal state or world model accordingly. The general workflow thus passes through the user interface.

Note here that, for the purpose of this document, the world model is completely hand modelled. It is, of course no problem to add world facts to the world model from the general architectire execution control when it perceives an affordance in a region.

4.3 Execution Control Module

The execution control module that has been implemented by UOS is specifically designed to meet the needs of this deliverable. The module receives plans that consist of operator sequences. The different operators trigger different execution methods that (if specified) extract the affordance label from the operator and retrieve the corresponding (as identified by their label) affordance representation triples (ARTs) from the affordance representation repository (initially implemented by OFAI). These triples contain in their cue-discriptor references to the affordance detection units that will detect the affordance (see 4.4). The execution control module then triggers the necessary modules and executes the behaviors that are encoded in the behavior descriptor of the detected triple. The actual ARTs that were used for this document have again been manually generated and again, it would not change anything from the execution and planning module's point of view if the needed representation were learned.

4.4 Affordance Detection Units

The affordance detection units that have been implemented by UOS to support the evaluation of the planning module are able to detect the affordance of a test object to be liftable to the robot, of the switch to be triggerable, and of the door to allow to drive through.

Both liftability and triggerability affordances are detected by means of surface detection in 3D laser scans. The implemented system extracts horizontal surfaces (see e.g. [8] or [9]) by clustering 3D scan points using a variant of the QT clustering algorithm [10] and selects those of appropriate size and height that correspond to either test objects with a flat top that can be lifted or the flat top of the switch that can be triggered. Note here

that we assume, for the purpose of this deliverable, that all test objects with a flat top are liftable and can trigger the switch while those test objects with a curved top are not liftable.

The affordance to drive through the open door is based on the robot's 2D laser data. The walls are detected using a simple line detection mechanism and the gap in between that is caused by an open door affords to the robot to drive through.

For the sake of simplicity, these affordance detection units were not integrated into the perception module of the overall architecture and while these affordance detection units surely are less complex then the ones developed for the complete architecture they proofed appropriate for the evaluation of the planning system.

4.5 Behavior System

For the purpose of demonstrating the whole robot in action, we have implemented a complete set of behaviors both for driving the robot and for controlling the crane. The set of crane control behaviors has been designed theoretically in a trilateral meeting between IAIS, OFAI and UOS in St. Augustin in May 2007. UOS has implemented the following set of behaviors:

DirectGoToPoseBehavior - The behavior gets a target pose, turns the robot on the spot to face into the right direction, drives to the target position and turns the robot to its target orientation.

3DScanBehavior - The behavior stops the robot and acquires a 3D scan over the area of interest as suitable in the MACS demonstrator arena.

ReachBehavior - The behavior gets a target position that should ideally correspond to the location of a detected liftability affordance, postions the crane (when the position is in reach) above the corresponding test object, and lowers the crane until a contact has been established.

PullBehavior - After the ReachBehavior has made contact, the pull behavior turns on the magnet of the crane and tries to lift the object a bit. It monitors the crane's weight sensor to decide if there is a weight attached to the crane or not.

RaiseBehavior - If the PullBehavior has monitored a weight on the crane, the test object is magnetic, liftable, and attached. The RaiseBehavior then lifts the already attached object

ReleaseBehavior - If the robot had a weight on its magnet, i.e. if it has lifted something, and makes contact to another item, e.g. the switch, using the ReachBehavior, the ReleaseBehavior is triggered that turns off the magnet and brings the crane back to its default position.

These behaviors are designed in a way that they can trigger themselves in a sequence. For a complete lift operation on a known location, the robot can thus simply activate the ReachBehavior and trigger the Pull- and RaiseBehavior afterwards. For putting an object on the switch, it can likewise use the sequence of reaching and releasing when already carrying an object.

All behaviors are designed to detect failure or success and report it back to the execution control that then can report it back to the GUI to allow for replaning.

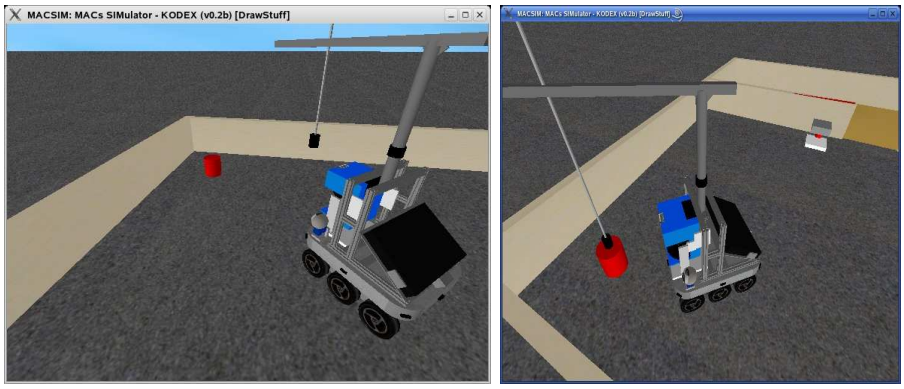

Figure 6: **Left:** The scenario setup for Task 1. **Right:** The robot has successfully lifted the test object.

5 Results

5.1 Task 1 - Lift Object

Listing 1 Plan for Task 1

1:	APPROACH-REGION	region1_left	region2_left	leftroom
2:	LIFT	region1_left		

Figure 1 displays the plan that was generated for Task 1. The robot has to drive to the region of the environment that is known to contain something liftable (See the scenario picture in figure 6 (left)). After the robot did reach the target area, it retrieved the affordance representation triple for liftability from the representation repository and instantiated the 3D-scan-behavior that was specified in the cue descriptor. The robot scanned the region with the red can in it and recognized the liftability affordance. It then grounded the lift operator on the red can that affords the liftability. The system hence performed as expected showing the successful grounding of the lift-operator.

5.2 Task 2 - Lift the liftable object

The scenario with a liftable and a not liftable object is shown in figure 9 (left). The plan generated for this task does not differ from the one generated for task 1. During the execution, the robot scans the region and detects only the red object as liftable since only flat can-tops are being detected by the liftability affordance detection unit (see section 4.4). The robot hence always chooses the red can to lift showing again its performance to ground the operator on the appropriate object.

5.3 Task 3 - Open the door

Listing 2 shows the plan generated by the planning module for task 3. The robot maps the lift operator during execution to the affordance label liftable and the trigger-switch operator to the affordance label switch-triggerable. In both cases it retrieved the corresponding

Figure 7: **Left:** The scenario setup for tasks 2 and 4. The red can has a flat magnetic top and can thus be lifted by the robot. The orange can has a round not magnetizable top and does not afford liftability. **Right:** The robot has successfully lifted the correct test object.

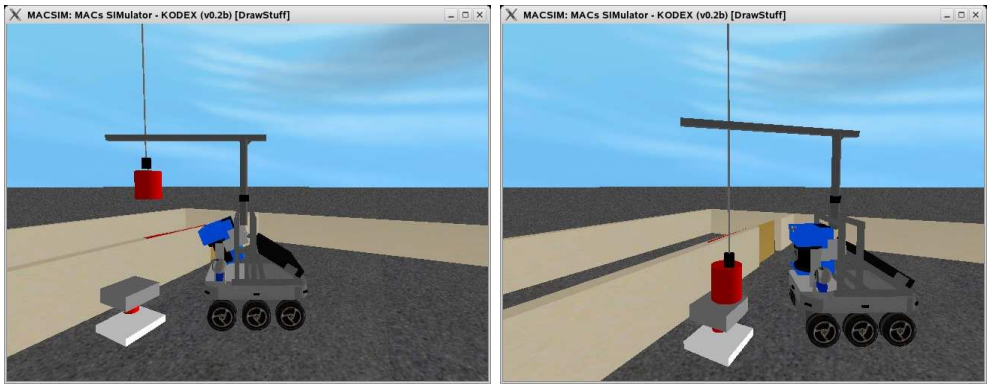

Figure 8: **Left:** The robot has already lifted the can and has approached the region with the switch. It scans the region to detect the switch-triggerable affordance. **Right:** The robot has successfully just placed the test object on the switch and the door in the background is opening.

Listing 2 Plan for Task 3

1:	APPROACH-REGION	region1_left	region2_left	leftroom
2:	LIFT	region1_left		
3:	CARRY	switchRegion	region1_left	leftroom
4:	TRIGGER-SWITCH	switchRegion		

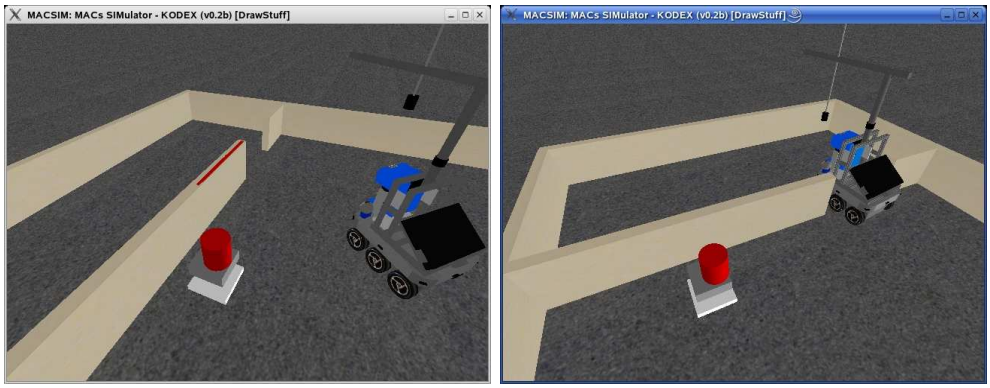

Figure 9: **Left:** The robot is facing the opened door and detecting the passable affordance. **Right:** The robot drives through the door.

ART from the ARR and instantiated the necessary behavior to detect the cue. In case of the switch-triggerable ART, the behavior descriptor contains the behavior sequence reach and release while for the lift operator the sequence of reach, pull, and raise is being used to ground the operator. Figure 9 shows the robot scanning for the switch-triggerable affordance (left) and opening the door by acting upon it (right).

5.4 Task 4 - Open the door with the right test object

This task is the combination of tasks 2 and 3. The generated plan is the same as for task 2 and, of course, the robot selected the appropriate object to trigger the switch as in the lifting case of task 2.

5.5 Task 5 - Change rooms

The most complex task that can be shown in the MACS demonstrator arena is to goal-directedly open the door in order to get to the other room. The plan that corresponds to this task makes use of the three affordances liftable, switch-triggerable and passable to ground the operators lift, trigger-switch and change-room. The complete plan is listed in listing 3 and in its GUI version in figure 5. The initial world model and the goal state are those depicted in figures 3 and 4.

The robot successfully executed the plan by selecting the appropriate affordance triples from the repository and grounding the operators on those test objects that afforded the desired actions.

To conclude this section and thus the evaluation of the planning module we can summarize that all experiments have shown the expected results. The demonstrated planning

Listing 3 Plan for Task 5

1:	APPROACH-REGION	region1_left	region2_left	leftroom
2:	LIFT	region1_left		
3:	CARRY	switchRegion	region1_left	leftroom
4:	TRIGGER-SWITCH	switchRegion		
5:	APPROACH-REGION	doorRegionLeft	switchRegion	leftRoom
5:	CHANGE-ROOM	doorRegionLeft	doorRegionRight	

system provides a valid approach for an affordance-based architecture and it shows most promising results on the area of operator grounding. Together with the defined representation and workflow of planning and plan execution, the planning module provides the functionality to elegantly combine the directly perceivable nature of affordances with goal-directed action selection and execution.

The experiments are also available as videos on:
http://www.informatik.uni-osnabrueck.de/∼cloerken/macs-videos.html

References

[1] C. Lörken and J. Hertzberg. A specification for a propositional planner and its interface to the MACS execution control module. Technical Report MACS/2/3.2 v1, University of Osnabrück, KBS Group, Osnabrück, Germany, March 2007.

[2] C. Lörken. Introducing affordances into robot task execution. In K.-U. Kühnberger, P. König, and P. Ludewig, editors, *Publications of the Institute of Cognitive Science (PICS)*, volume 2-2007. University of Osnabrück, Osnabrück, Germany, May 2007. ISSN 1610-5389.

[3] A.P. Duchon, W.H. Warren, and L.P. Kaelbling. Ecological robotics. *Adaptive Behavior*, 6(3):473–507, 1998.

[4] P. Doherty, T. Merz, P. Rudol, and M. Wzorek. Tentative Proposal for a Formal Theory of Affordances, Tentative Proposal for an Affordance Support Architecture, Prototype: Affordance-Based Motion Planner. Deliverable D4.2.1 + D4.3.1, MACS Internal Technical Report, LiU-IDA Linköpings Universitet, Linköping, Sweden, October 2005.

[5] R. Breithaupt. Report on experiment design. Technical Report MACS/6/4.1 v3, FhG/AIS, St. Augustin, Germany, September 2005.

[6] D. McDermott. PDDL - The planning domain definition language. Technical report, Yale University, 1998.

[7] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[8] S. Stiene, K. Lingemann, A. Nüchter, and J. Hertzberg. Contour-based Object Detection in Range Images. In *Proceedings of the 3rd IEEE International Symposium on 3D Data Processing, Visualiza tion and Transmission (3DPVT '06)*, June 2006.

[9] A. Nüchter, K. Lingemann, and J. Hertzberg. Extracting Drivable Surfaces in Outdoor 6D SLAM. In *Proceedings of the 37rd International Symposium on Robotics (ISR '06) and Robotik 2006*, Munich, Germany, May 2006.

[10] L.J. Heyer, S. Kruglyak, and S. Yooseph. Exploring Expression Data: Identification and Analysis of Coexpressed Genes. In *Genome Res.*, volume 9-1999, pages 1106–1115, Munich, Germany, 1999.

A Domain Description

```
(define (domain macs-demonstrator)

   (:requirements :strips :typing :equality )

   (:types doorRegion switchRegion - region
           switchRoom - room )

   (:predicates

        (passable
                  ?startRegion - region
                  ?targetRegion - region )
        (switch-triggerable ?region - switchRegion )
        (affords-removing-from-switch ?region - switchRegion )

        (liftable ?region - region)

        (robotAt ?robotRegion - region )
        (inRoom ?region - region ?room - room )
        (hasLiftedSomething )

   )

(:action approach-region
    :parameters (?startRegion - region
                 ?targetRegion - region
                 ?room - room)
    :precondition
        (and
            (robotAt ?startRegion )
            (not (hasLiftedSomething ))
            (inRoom ?startRegion ?room )
            (inRoom ?targetRegion ?room )
            (not (= ?startRegion ?targetRegion )))
    :effect
        (and
            (robotAt ?targetRegion )
            (not (robotAt ?startRegion ))))

(:action carry
    :parameters (?startRegion - region
                 ?targetRegion - region
                 ?room - room )
    :precondition
        (and
            (robotAt ?startRegion )
            (hasLiftedSomething )
```

```
            (inRoom ?startRegion ?room )
            (inRoom ?targetRegion ?room )
            (not (= ?startRegion ?targetRegion )))
    :effect
        (and
            (robotAt ?targetRegion )
            (not (robotAt ?startRegion ))))


(:action lift
    :parameters (?region - region ?room - room )
    :precondition
        (and
            (robotAt ?region )
            (liftable ?region )
            (inRoom ?region ?room )
            (not (hasLiftedSomething )))
    :effect
        (and
            (hasLiftedSomething )
            (not (liftable ?region ))))

(:action drop
    :parameters (?region - region ?room -room)
    :precondition
        (and
            (robotAt ?region )
            (inRoom ?region ?room )
            (hasLiftedSomething ))
    :effect
        (and
            (not (hasLiftedSomething ))
            (liftable ?region )))

(:action trigger-switch
    :parameters   (?doorRegion ?otherDoorRegion - doorRegion
                   ?switchRegion - switchRegion )
    :precondition
        (and
            (robotAt  ?switchRegion )
            (hasLiftedSomething )
            (switch-triggerable ?switchRegion )
            (not (= ?doorRegion ?otherDoorRegion )))
   :effect
        (and
            (passable ?doorRegion ?otherDoorRegion )
            (passable ?otherDoorRegion ?doorRegion )
            (not (switch-triggerable ?switchRegion ))
            (affords-removing-from-switch ?switchRegion )
            (not (hasLiftedSomething ))))


(:action remove-from-switch
```

```
    :parameters    (?doorRegion ?otherDoorRegion - doorRegion
                    ?switchRegion - switchRegion )
    :precondition
        (and
            (robotAt  ?switchRegion )
            (not (= ?doorRegion ?otherDoorRegion ))
            (not (hasLiftedSomething ))
            (affords-removing-from-switch ?switchRegion )
            (passable ?doorRegion ?otherDoorRegion )
            (passable ?otherDoorRegion ?doorRegion ))
   :effect
        (and
            (not (passable ?doorRegion ?otherDoorRegion ))
            (not (passable ?otherDoorRegion ?doorRegion ))
            (switch-triggerable ?switchRegion )
            (not (affords-removing-from-switch ?switchRegion ))
            (hasLiftedSomething )))

(:action change-room
    :parameters (?doorRegion - doorRegion
                 ?targetDoorRegion - doorRegion )
    :precondition
        (and
            (robotAt ?doorRegion)
            (not (hasLiftedSomething ))
            (not (= ?doorRegion ?targetDoorRegion ))
            (passable ?doorRegion ?targetDoorRegion ))
    :effect
        (and
            (not (robotAt ?doorRegion ))
            (robotAt ?targetDoorRegion )))

(:action carry-through
    :parameters (?doorRegion - doorRegion
                 ?targetDoorRegion - doorRegion )
    :precondition
        (and
            (robotAt ?doorRegion )
            (hasLiftedSomething )
            (not (= ?doorRegion ?targetDoorRegion ))
            (passable ?doorRegion ?targetDoorRegion ))
    :effect
        (and
            (not (robotAt ?doorRegion ))
            (robotAt ?targetDoorRegion )))
)
```

B Module Interfaces

B.1 IPlanningModule.idl

```
module Macs
{
  module idl {
    interface IPlanningModule {
      struct StrOperator {
        string name;
        string affordanceLabel;
        string parameter;
      };
      typedef sequence<StrOperator> PlanSeq;
      typedef sequence<string> GoalSeq;

      /** Specify a domain description (PDDL syntax) that the
        * planner will work on. */
      void setDomain(in string szDomain);

      /** Specify a complete world model (PDDL syntax) that the
        * planner will work on. */
      void setWorldModel(in string szWorldModel);

      /** Triggered by GUI. Will call EM executePlan when plan is
        * ready and tell this to the GUI. */
      void plan(in GoalSeq goal);

      /** Triggers the planner with the current world model and
        * goal. */
      void planCurrent();

      /** Triggered by EC on failure. The failIndex indicates the
        * operator that has failed. If it was an affordance related
        * operator, the robot will remove the affordance from that
        * region within its world model.
        *
        * Will call EM executePlan when plan is ready and tell this
        * to the GUI. */
      void replan(in long failIndex);

      /** Resets the world model of the planner to the default
        * setting. */
      void resetWorldModel();

      /** Resets the goal state of the planner to an empty goal
        * state. */
      void resetGoalState();

      /** Triggered by EC on percept of new affordance triple type.
        * Can be called by GUI as well.
        * If the boolean is false, the percept will be reomved from
        * that region. */
```

```
      long updateAffordancePercept(in string affordanceLabel,
                                   in string regionLabel,
                                   in boolean add);

      /** Called for visualization by GUI. Returns all affordance
        * percepts of the world model. */
      string getAffordancePercepts();

      /** Called by EC when the robot changes its region.
        * Can be called by GUI as well. */
      boolean updateRobotPosition(in string regionLabel);

      /** Called by the GUI server when it connects and when it
        * disconnects. To allow the planner to connect to the GUI
        * itself. */
      void connectToServer(in string client, in boolean connect);
    };
  };
};
```

B.2 IGUI.idl

```
module Macs {
  module idl {
    interface IGUI {
      /** Called by the planner if the plan has changed. */
      void setCurrentPlan(in IPlanningModule::PlanSeq plan);

      /** Set by EM. Contains the plan-index of the currently
        * executed operator. */
      void setExecutingOperator(in long index);

      /** Called by the planner if the world model has changed.
        * Note that the robot pose is being read from the ESGM. */
      updateWorldModel( in string affordanceLabel,
                        in string region,
                        in Boolean add);
    };
  };
};
```

B.3 IExecutionControlModule.idl

```
module Macs
{
  module idl {
    interface IExecutionControlModule {
      struct ART {
        string cueDescriptor;
        short behaviorDescriptor;
        string outcomeDescriptor;
      };

    /**
      * This method is used by the Planner Module to pass a
      * Plan structure to the ECM, to be executed.
      * Currently this function only supports totally ordered
      * plans.
      */
    void executePlan (in IPlanningModule::PlanSeq plan);

    /**
      * This method is used by the Planner Module to stop
      * the execution of the current plan, it has previously
      * started.
      */
    void stopExecution ();

    /**
      * This method is called by the Behavior Module
      * to indicate that a problem has occured during the
      * execution of a behavior.
      * Currently ECM responds to this situation by stopping the
      * execution of the current plan, and informing Planner Module
      * of the failure to execute the plan
      */
    void behaviorException ();

    /** Called by the GUI server when it connects and when it
      * disconnects. To allow the planner to connect to the
      * GUI itself.
      */
    void connectToServer(in string client, in boolean connect);
    };
  };
};
```