Project Acronym:	GRASP
Project Type:	IP
Project Title:	Emergence of Cognitive Grasping through Introspection, Emulation and Surprise
Contract Number:	215821
Starting Date:	01-03-2008
Ending Date:	28-02-2012

Deliverable Number:	D8
Deliverable Title :	Systems Integration in GRASP
Type (Internal, Restricted, Public):	PU
Authors	T. Asfour, M. Do, S. Ulbrich, F. Hecht, M. Przybylski, B. Leon, A. Morales, D. Kragic, and R. Dillmann
Contributing Partners	UniKarl, KTH, UJI

Contractual Date of Delivery to the EC:	28-02-2009
Actual Date of Delivery to the EC:	28-02-2009

Contents

1 Executive Summary **5**

2 Grasping on the Humanoid Robot ARMAR-III **7**

2.1 The Humanoid Robots ARMAR-IIIa and ARMAR-IIIb 7

2.2 The Software Framework MCA (Modular Controller Architecture) 8

2.3 Robot Interface: Robot Programming Interface (API) 8

2.4 Grasping and Manipulation System 10

3 Software Interfaces **13**

3.1 Software Interfaces to Existing Simulators and Frameworks 13

3.1.1 Software Interfaces between ARMAR-III and GraspIt! 13

3.1.2 Software Interfaces between ARMAR-III and OpenRAVE 14

3.2 Software Interfaces for the GRASP Simulator 16

3.3 Interfaces for Vision on ARMAR 17

3.4 Interfaces for Haptics on ARMAR 17

3.5 Interfaces for Actions 18

4 Scene Representation and Object Models **21**

4.1 Scene Representation 21

4.2 GRASP Object Database 22

4.2.1 Initial Object Set 22

4.2.2 Acquisition of Object Models 22

4.2.3 XML Representation 25

4.3 Object Representations 25

4.3.1 Object Attributes 26

5 GRASP Robot Editor **29**

5.1 Concept and Features 29

5.2 The Blender Project 29

5.2.1 Framework for the Robot Editor 29

5.2.2 Robot Models in Blender 30

5.2.3 Client for Visualization 31

5.3 COLLADA 31

5.3.1 Scene and Robot Models in COLLADA 31
5.3.2 Compatibilty and GRASP-specific extensions 32

6 Further Integration Activities **35**
6.1 GRASP Software Subversion Repository 35
6.2 Administration of GRASP Wiki Online Content 35
6.3 Visual Data Recording of Human Grasping 35
6.4 Grasping based on Box Decomposition on ARMAR 36
6.5 Integration of Controller Architecture of LUT at UJI 36
6.6 Object Database 36
6.7 Grasp Taxonomy and Human Grasping Database 37

7 Conclusions and Future Work **39**

Chapter 1

Executive Summary

This Deliverable deals with system and integration aspects within the GRASP project. This includes the coordination and alignment of representations necessary for the integration of functional and software components as well as the definition of an infrastructure that allows for efficient exchange of software modules of different partners in order to minimize the overhead in the overall system development. One of the goals of the project is the implementation, demonstration and evaluation of the application scenario on different platforms available in the project. Therefore, efforts in the first year were devoted to the definition of clear interfaces between perception, action, learning and reasoning components at all system levels for the humanoid platform at UniKarl. In this context, the integration of the aimed-at GRASP simulator and the transferability of representations of objects, actions, and human grasping developed within the project received special attention.

Integration in an Integrated Project like GRASP is a very challenging task, which requires a careful consideration of the interactions between different system components. In the first year, the work was concentrated on the one hand on providing interfaces for perception, action, reasoning and planning by extending the currently implemented software control architecture and the robot application programming interface (API) of the humanoid robot ARMAR-III. On the other hand, additional components have been developed to meet the requirements of the project and in particular the integration of the GRASP simulator in a holistic cognitive control architecture.

According to the Technical Annex, Deliverable D8 presents the activities performed in the first year of the project in the context of the tasks 7.2, 7.3, 7.4 and 7.6. The objectives of these tasks are:

Task 7.1 Development and implementation of a cognitive control architecture for grasping and dexterous manipulation necessary to bootstrap the system integration and the evaluation of methods and algorithms in different scenarios on different robot platforms as it is the long-term goals of this work package, in particular of task 7.5.

Task 7.2 Software infrastructure that allows for smooth and efficient exchange of modules in the project in order to minimize the overhead in the overall system integration.

Task 7.3 Specification of interfaces for knowledge and control flows between perception, action, learning and reasoning modules.

Task 7.4 Definition of the first year scenario, which is necessary for evaluating the developed algorithms in the fist year of the project.

Task 7.6 Benchmarking environment for grasping and dexterous manipulation including software robot control frameworks, data sets of objects, robot and human hand models as well as grasp related algorithms.

In addition, initial work has been also performed in close cooperation with all partners, in particular within WP2 and WP7 on task 7.1, which deals with the identification of key components of the architecture and the conceptual design as well as the implementation of this architecture.

The document is organized as follows. In Chapter 2, we briefly present the manipulation and grasping system on the humanoid robot ARMAR-III at UniKarl, underlying software framework and the robot interface, which offers an API for convenient access to the robot's sensors and actors and provides three abstraction levels: skills, tasks, scenarios for integration of higher level components. We further describe the currently implemented grasping system, which incorporates a vision system for the localization and recognition of objects, a path planner for the generation of collision-free trajectories, and an offline grasp analyzer that provides the most feasible grasp configurations for each object using the grasping simulation environment GraspIt!.

In Chapter 3, the software interfaces established in the first year are described. These are interfaces to existing, widely used and publicly available simulation environments and software frameworks in the robotics community and in particular in the context of grasping as well as interfaces for perception (vision and haptics) and action. In addition, an initial specification of the interfaces to the GRASP simulator is given.

Chapter 4 deals with the scene and object models. For the scene representation, a scene graph is used to store the information about objects and environment. In particular, we describe the initial object set, the different object representations, the used object database as well the acquisition of 3D object models.

Chapter 5 describes the current state of the development of the *Robot Editor*, a standardized tool which allows the creation of geometric, kinematic and dynamic robot models as well as their convenient conversion from a variety of CAD formats. This is necessary to allow the integration of different robot platforms used in the project in the GRASP simulator.

Chapter 6 provides a summary on further integration activities in the project, which are related to software infrastructure and databases established and already used in the project. Chapter 6 concludes the Deliverable with a view on future work.

Chapter 2

Grasping on the Humanoid Robot ARMAR-III

2.1 The Humanoid Robots ARMAR-IIIa and ARMAR-IIIb

The humanoid robots ARMAR-IIIa and ARMAR-IIIb (see Fig. 2.1) which are used as platforms in GRASP have been developed within the Collaborative Research Center (SFB 588)[1] under a comprehensive view so that a wide range of tasks can be performed. From the kinematics control point of view, each of the robots consists of seven subsystems: head, left arm, right arm, left hand, right hand, torso, and a mobile platform. In the following, the subsystems of the robot are briefly described. For detailed information the reader is referred to [AAV$^+$08] and [ARA$^+$06]. For a detailed description of the mechanics, the reader is referred to [ABB04].

The head has seven DoF and is equipped with two eyes. The eyes have a common tilt and can pan independently. Each eye is equipped with two color cameras, one with a wide-angle lens for peripheral vision and one with a narrow-angle lens for foveal vision. This combination allows simple visuo-motor behaviors such as tracking and saccadic motions toward salient regions, as well as more complex visual tasks such as hand-eye coordination. The visual system is mounted on a four DoF neck (lower pitch, roll, yaw, upper pitch). For the acoustic localization, the head is equipped with a microphone array consisting of six microphones (two in the ears, two in the front and two in back of the head). Furthermore, an inertial sensor is installed in the head for stabilization control of the camera images.

The upper body of the robot provides 33 DoF: 14 DoF for the arms, 16 DoF for the hands and three DoF for the torso. The arms are designed in an anthropomorphic way: three DoF in the shoulder, two DoF in the elbow and two DoF in the wrist. Each arm is equipped with a five-fingered hand with eight DoF (see [SPK$^+$04, KAP$^+$05, GSK$^+$08]). The goal of performing manipulation tasks in human-centered environments generates a number of requirements for the sensor system, especially for that of the manipulation system. Each joint of the arms is equipped with motor encoder, axis sensor and joint torque sensor to allow position, velocity and torque control. In the wrists 6D force/torque sensors are used for hybrid position and force control. Four planar skin pads (see [GWBW06]) are mounted on the front and back side of each shoulder, thus also serving as a protective cover for the shoulder joints. Similarly, cylindrical skin pads are mounted to the upper and lower arms respectively.

The locomotion of the robot is realized using a wheel-based holonomic platform, where the wheels are equipped with passive rolls at the circumference (Mecanum wheels or Omniwheels). The sensor system of the platform consists of a combination of three Laser-range-finders (Laser-scanner) and optical encoders to localize the platform. The platform hosts the power supply of the robot and the main part of the robot computer system.

[1] `http://www.sfb588.uni-karslruhe.de`

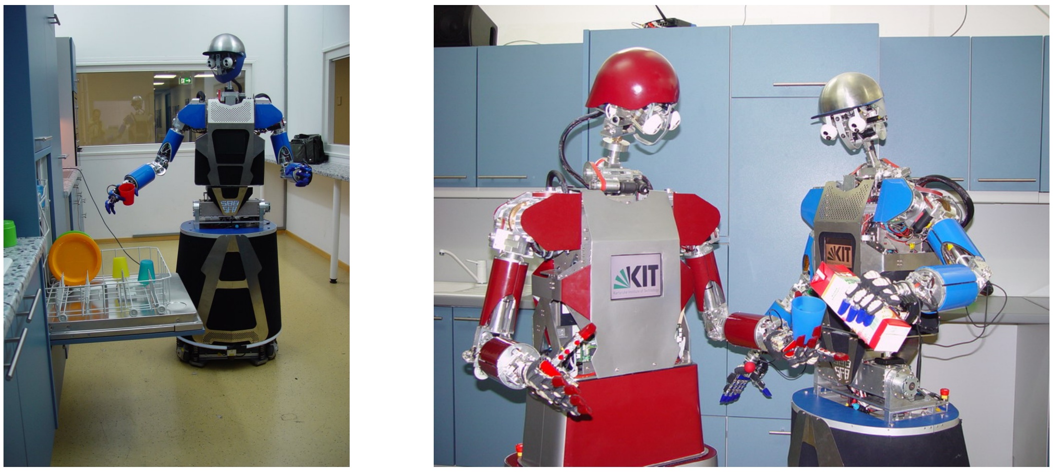


Figure 2.1: The humanoid robots ARMAR-IIIa (blue) and ARMAR-IIIb (red) in the kitchen. Each robot has an active head with foveated vision, two arms, two five-fingered hands and a holonomic mobile platform.

2.2 The Software Framework MCA (Modular Controller Architecture)

The basic control software is implemented in the Modular Controller Architecture framework MCA[2]. MCA is an open source modular, network transparent and real-time capable C/C++ framework for controlling robots and other kinds of hardware. In MCA, all methods are implemented by C++ modules with standardized interfaces and thus allow an easy integration of software components. The control parts can be extended under Linux, RTAI/LXRT-Linux, Windows, or Mac OS, and communicate beyond operating system borders. A graphical debugging tool (`mcabrowser`), which can be connected via TCP/IP to the MCA processes, can visualize the connection structure of the control modules and their current values at run-time. The tool `mcagui` can create graphical user interfaces at runtime with various input and output entities. Both interfaces provide full access to the control parameters at runtime.

2.3 Robot Interface: Robot Programming Interface (API)

Although MCA offers a very clean and transparent mechanism for developing and integrating relatively low-level software components related to control, kinematics, and dynamics, it is not suitable for the convenient implementation of higher-level modules. Thus, we have built a so-called ”Robot Interface” on top of MCA, building an abstraction level for the access to the robot’s sensors and actuators. By using the robot interface, the user is not aware in which way the information is communicated to and from the relevant component of the robot. Scenarios can be implemented by different partners by simply inheriting and implementing the scenario interface. In this way, the incoming messages, which result from sensory data processing are passed by calling the corresponding callback method of the interface. On the action side, from within a scenario, commands can be passed to the robot by simple method calls, such as:

- `MoveHead`: This method has different modes including forward kinematics, inverse kinematics with absolute positions, and two different tracking modes.
- `MoveArm`: This method has different modes including forward kinematics and different types of inverse kinematics.
- `MovePlatform`: This method has different modes including velocity-based control, point to point navigation, and navigation on graphs.

On the perception side, a scenario has access to the following:

[2] `http://www.mca2.org`

- All current joint angle values for the head and the arms.
- Current position of the robot platform.
- Other sensor values, such as sensors integrated in the hands.
- Higher-level perceptive modules such as vision components accomplishing a specific task for the visual analysis of the environment. In general, two classes can be distinguished: Modules recognizing and localizing known entities and modules exploring unknown scenes.

From the integrative point of view, it is important that all modules share the same interface in order to guarantee that vision components from the partners can be built in and replaced with minimum effort. Especially, the step from analyzing scenes containing known entities, which have been learned, to the autonomous exploration of unknown scenes can be prepared very elegantly. Our perception modules produce output as 3D rotations and translation in terms of 3D object models or 3D primitives (given in the Open Inventor format), such as planes, spheres, or cylinders. These are at the same time used as input of the robot's grasp planning and execution system. By sharing the same data structures and output formats, we will be able to replace our currently used object recognition and localization systems by a vision system being developed in the project.

To allow effective and efficient programming of the robot, in addition to direct access to the robots sensors and actors, three abstraction levels are defined: scenarios, tasks and skills (see Fig.2.2). While skills are implemented capabilities of the robot that can be regarded as atomic, tasks have been been implemented so far manually by combination of several skills for a specific purpose. Scenarios are combinations of several tasks and skills which are necessary to perform a specified goal. Currently, on ARMAR-III the following skills are available:

- `SearchForObjectSkill`: Searching for known objects using the active head. This skill scans the space in front of the robot by moving the head and performing a full scene analysis at each target position. The skill can be parameterized to either search for a specific object or include all object representations currently available in the database.
- `VisualGraspSkill`: Grasping of objects. This skill makes use of a visual servoing approach including one arm and the hip to grasp an object that has been previously recognized and localized. It controls the position closed-loop while continuously updating the position of the robots hand and the target object. The hand is tracked by using a marker.
- `PlaceSkill`: Placing objects on even surfaces. With this skill, it is possible to place a previously grasped object on an even surface, such as a table. Force information acquired from a 6D force

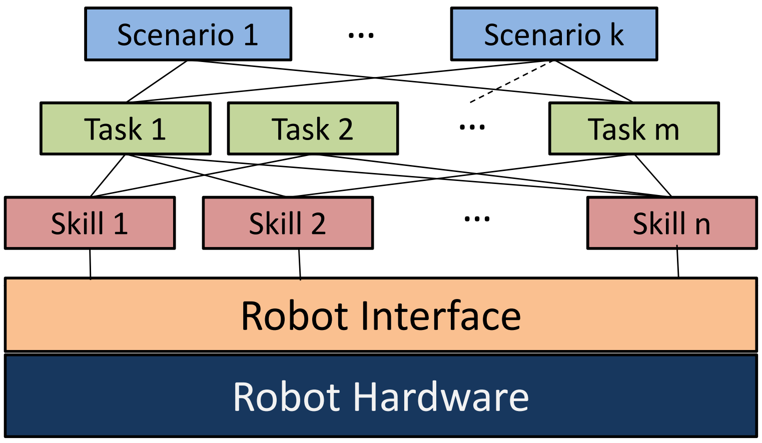


Figure 2.2: Software control architecture. In addition to the robot interface (API), three abstraction levels are defined: skills, tasks and scenarios. Skills are implemented as atomic capabilities of the robots. Tasks are combinations of skills and scenarios are combinations of skills and tasks to achieve a specified goal.

sensor in the wrist is evaluated to determine the contact forces with the surface while placing the object.

- `HandOverSkill`: Handing over objects to or from the robot. With this skill, objects can be either handed over to the robot or received from the robot. In both cases, information acquired from a 6D force sensor is evaluated to determine when to close (open) the hand.
- `OpenDoorSkill`: Opening various doors. This skill can be regarded as a more complex respectively higher-level skill compared to the previously introduced ones, since it makes use of other skills. It first uses a module that can recognize and localize handles of doors, which are then grasped making use of the VisualGraspSkill. Once the handle has been grasped, this skill opens the door using a force-control approach.
- `CloseDoorSkill`: Closing various doors. Using this skill, open doors can be closed, given an initial target position for touching the door of interest. Again, a force-control approach is used to perform the action.

All skills have in common a continuously called run method, in which they perform their action and sensing. The return value of this method signalizes its state, i.e. whether the skill is still *operating*, finished with *success*, or finished with *failure*. After success or failure, a skill switches to the state *waiting*. Each skill can be parameterized and has its own specific configuration data structure for this purpose. In the same way, the result of a skill is communicated.

2.4 Grasping and Manipulation System

The central idea of our approach for the programming and execution of manipulation tasks is the existence of a database with 3D models of all the objects encountered in the robot workspace and a 3D model of the robot hand. This allows for an extensive offline analysis of the different possibilities to grasp an object, instead of focusing on tuned online approaches. From this central fact, we have developed an integrated grasp planning system, which incorporates a vision system for the localization and recognition of objects (see [AAD06, AAD07a]), a path planner for the generation of collision-free trajectories (see [VAD07, VSA+08]), and an offline grasp analyzer that provides the most feasible grasp configurations for each object using the grasping simulation environment GraspIt! [MA04]. The results provided by these modules are stored and used by the control system of the robot for the execution of a grasp of a particular object. The functional description of the grasp planning system is depicted in Fig. 2.3. We emphasize that our approach describes a first step toward a complete humanoid grasping and dexterous manipulation system. The integrated grasp planning system, which has been presented in [MAA+06], will be explained briefly in the following.

The system consists of the following parts:

- The *global model database*. It contains not only the CAD models of all the objects, but also stores a set of feasible grasps for each object. Moreover, this database builds the interface between the different modules of the system.
- The *offline grasp analyzer* that uses a model of the object to be grasped together with a model of the hand to compute a set of stable grasps in a simulation environment. The results of this analysis are stored in the grasp database and can be used by the other modules.
- A *online visual procedure to identify objects in stereo images* by matching the features of a pair of images with the 3D pre-built models of such objects. After recognizing the target object it determines its location and pose.
- Once an object has been localized in the workspace of the robot, a grasp type for this object is then selected from the set of pre-computed stable grasps. This is instantiated to a particular arm/hand configuration that takes into account the particular pose and reachability conditions of the object. This results in an approaching position and orientation. The path planner generates collision-free trajectories to reach the specified grasp position and orientation.

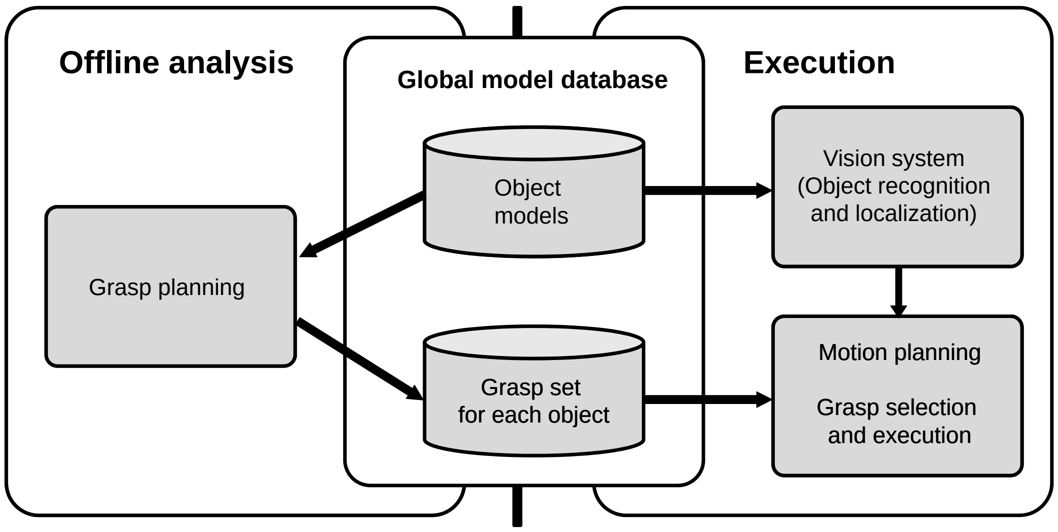

Offline analysis
Global model database
Execution
Object models
Grasp planning
Grasp set for each object
Vision system (Object recognition and localization)
Motion planning
Grasp selection and execution

Figure 2.3: Functional description of the integrated grasp planning system.

Offline grasp analysis: In most of the works devoted to grasp synthesis, grasps are described as sets of contact points on the object surface where forces/torques are exerted. However, this representation of grasps suffers from several disadvantages when considering the grasp execution in human-centered environments. These problems arise from the inaccuracy and uncertainty about the information of the object. Since we are using models of the objects, this uncertainty comes mainly from the localization of the object. Usually, the contact-based grasp description requires the system to be able to reach precisely the contact points and exert precise forces. In our approach, grasps are described in a qualitative and knowledge-based fashion. Given an object, a grasp of that object will be described by the following features (see Fig. 2.4):

- *Grasp type:* A qualitative description of the grasp to be performed. The grasp type has practical consequences since it determines the grasp execution control, i.e. the hand preshape posture, the control strategy of the hand, the fingers that are used in the grasp, the way the hand approaches the objects and how the contact information of the tactile sensors is interpreted.
- *Grasp starting point (GSP):* For approaching the object, the hand is positioned at a distant point near it.
- *Approaching direction:* Once the hand is positioned in the GSP it approaches the object following this direction. The *approaching line* is defined by the GSP and the approaching direction.
- *Hand orientation:* the hand can rotate around the approaching direction. The rotation angle is a relevant parameter to define the grasp configuration.

It is important to note that all directions are given with respect to an object centered coordinate system. The approach directions result from matching of this relative description with the localized object pose in the workspace of the robot. An important aspect when considering an anthropomorphic hand is how to relate the hand with respect to the grasp starting point (GSP) and the approaching direction. For this purpose we define the grasp center point (GCP) of the hand. It is a virtual point that has to be defined for every hand and that is used as reference for the execution of a given grasp (see Fig. 2.4). The GCP is aligned with the GSP of the grasp. Then, the hand is oriented and pre-shaped according to the grasp descriptors and finally moves along the approaching line.

The main advantage of this grasp representation lies in its practical application since the grasp can be executed based on the information given by the grasp definition. In addition, this representation is more robust to inaccuracies since it only describes starting conditions and not final conditions like a description based on contact points.

We perform an extensive offline grasp analysis for each object by testing a wide variety of hand pre-shapes and approach directions. The analysis is carried out in a simulation environment, where every

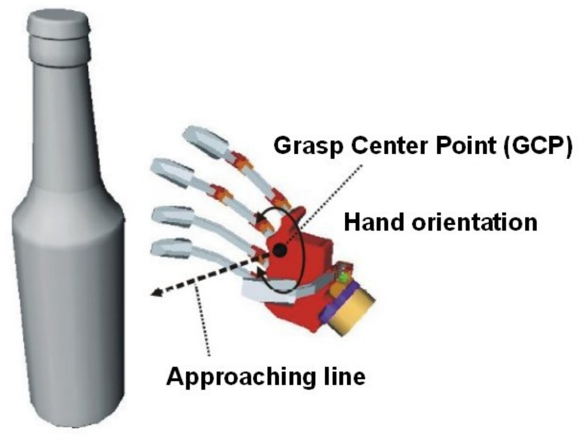


Figure 2.4: Schematics with the grasp descriptors

tested grasp is evaluated according to a quality criterion. The resulting best grasps for each object are stored in order to be used during online execution on the robot. As grasping simulation environment we use GraspIt! [MA04], which has convenient properties for our purposes such as the inclusion of contact models and collision detection algorithms, as well as the ability to import, use and define object and robot models. Due to the mechanical limitations of the robot hand, we have made a selection of the most representative grasps that can be executed by the robot hand. Fig. 2.5 shows the grasp patterns we have considered in our analysis. These are three power grasps (hook, cylindrical and spherical) and two precision grasps (pinch and tripod).

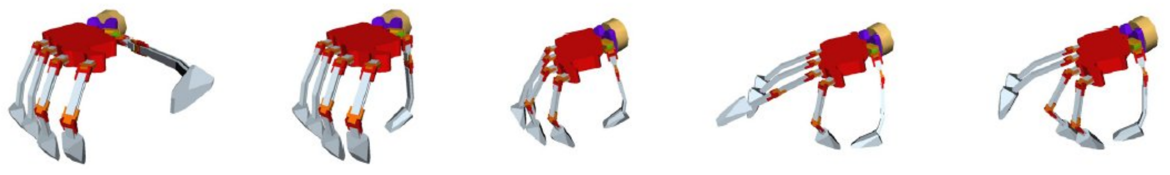


Figure 2.5: Hand preshapes for the five representative grasp types.

On the humanoid robot ARMAR III, implementations of the approaches mentioned above are successfully applied for manipulation and grasping tasks in a kitchen environment using visual servoing, as shown in [AAV$^+$08, VWA$^+$08].

Chapter 3

Software Interfaces

In this chapter, we describe extensions of the software interfaces on ARMAR-III to meet the requirements for the integration software components developed in other work packages. This includes interfaces to existing grasp simulators and robot control frameworks, interfaces to the aimed-at GRASP simulator as well as interfaces for perception and action components.

3.1 Software Interfaces to Existing Simulators and Frameworks

3.1.1 Software Interfaces between ARMAR-III and GraspIt!

Similar to the grasp planning system introduced in Section 2.4, other project partners have developed systems that also incorporate the simulator GraspIt!. Therefore, providing an interface between GraspIt! and the ARMAR-III API, which we call *robot interface* (see Section 2.3), is essential for the integration of these systems. For a case study, the box decomposition grasping system presented in [HRK08, HK08] has been integrated on the humanoid robot ARMAR-III. For this purpose, a communication framework has been created, which provides GraspIt! with essential information for grasp planning and sends the resulting grasp hypotheses to ARMAR-III.

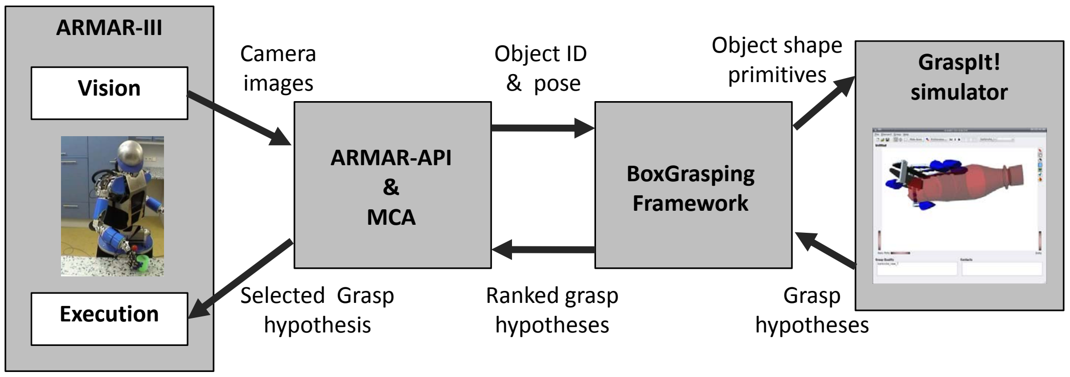


Figure 3.1: Connecting ARMAR-III and GraspIt!. Communication was established using the TCP protocol to exchange the object ID and poses as well as the resulting grasp hypotheses.

The focus of this section is on communication interface; details of the box decomposition grasping method are described in Section 6.4. The framework consists of the following components:

- **GraspIt!**: A robotic grasp planning simulation environment (see Section 2.4).
- **BoxGrasping** system: implementation of the box decomposition grasping method presented in [HRK08, HK08]. The system computes a course geometric approximation of an object model using

minimum volume bounding boxes. The resulting simplified box-based representation is then sent to GraspIt! for grasp planning.

- **ARMAR-III**: ARMAR-III is the humanoid robot that is used as platform for experimental evaluation of the developed concepts and systems.
- **Robot Interface**: The robot programming API for ARMAR-III (see Section 2.3). It provides a high-level application interface which allows for easy access to the robot's sensors and actuators.
- **Box Grasping scenario**: A scenario is the highest abstraction level of the robot interface. The box grasping scenario implements the communication on the robot side with the box grasping framework and executes the grasps on the robot.

As depicted in Fig. 3.1, the interface connects the humanoid robot ARMAR-III to GraspIt! via TCP/IP communication. To enable grasp planning on objects in the robot's environment, information acquired by the robot's perception system about object size, shape, and pose has to be sent to GraspIt!. For this purpose, the box grasping scenario has been implemented, which has access to the sensory information of the robot and sends the relevant information for grasp planning to GraspIt!. Based on the object information, GraspIt! performs a grasp planning procedure, which results in a number of grasp hypotheses including approach directions. To perform a grasp, these hypotheses are sent back to the box grasping scenario, which evaluates and ranks the hypotheses according to their quality and feasibility. The best candidate is then executed. An experimental evaluation involving grasp planning and grasp execution with different kinds of objects has proved the operability of this integrating framework.

3.1.2 Software Interfaces between ARMAR-III and OpenRAVE

Since OpenRave was accepted by the project partners as the basis of the new robot simulator being developed in WP6, communication between the humanoid robot ARMAR-III and the robot simulation software OpenRAVE [DK08][1] was established. The communication setup was quite similar to the interface that is described in Section 3.1.1. OpenRave already offers an input/output interface that is mainly intended to communicate with MATLAB. This interface allows a user to define and execute actions in a simulator. However, the original interface is not well suited for communication between the actual robot and the simulator. As OpenRAVE is extendible via plugins, an actuator plugin representing the robot was created. All joint movement commands passed to the plugin during simulation are packaged and transmitted to a handler listening in the robot's control loop. The handler on his part reports the actual position of its joints and end effectors back to the simulation. The protocol used in this communication again consists of the plain transmission of command objects over a TCP/IP connection (see Fig. 3.3). This exemplary setup can be seen as a proof of concept using OpenRAVE as a simulator in conjunction with a real robot platform.

Recently, the developers of OpenRAVE have established compatibility to the Robot Operating System (ROS)[2] and consequently suggest to use its functionality as a standard communication protocol for distributed robot applications. As a matter of fact, this system seems to meet all desired capabilities on top of being freely available under an open source license. The features are:

- **Multiple language support:** Currently, C++, Python, Lisp and MATLAB are supported. Most robot controllers are implemented in C++ while MATLAB and Python can be used to directly address commands to the robot and simulator. Moreover, support of the Python language can be useful for building up a visualization client with the same tools as the Robot Editor. The Robot Editor will be discussed in the Sections 5.1 and 5.2.
- **Distributed Communication:** Clients communicate directly with each other over a network. A master node is only needed to build up communication.

[1]OpenRAVE was founded by Rosen Diankov at the Planning And Autonomy Lab of the Carnegie Mellon University Robotics Institute. It was inspired by the RAVE simulator James Kuffner had started developing in 1995 and used for his experiments ever since. The OpenRAVE project was started in 2006 and is a complete rewrite of RAVE. The main goal with OpenRAVE is to create a planning architecture that would give robotics researchers an easy open source interface to control their robots both in simulation and in the real-world without having to worry about the small details.

[2]`http://pr.willowgarage.com/wiki/ros`

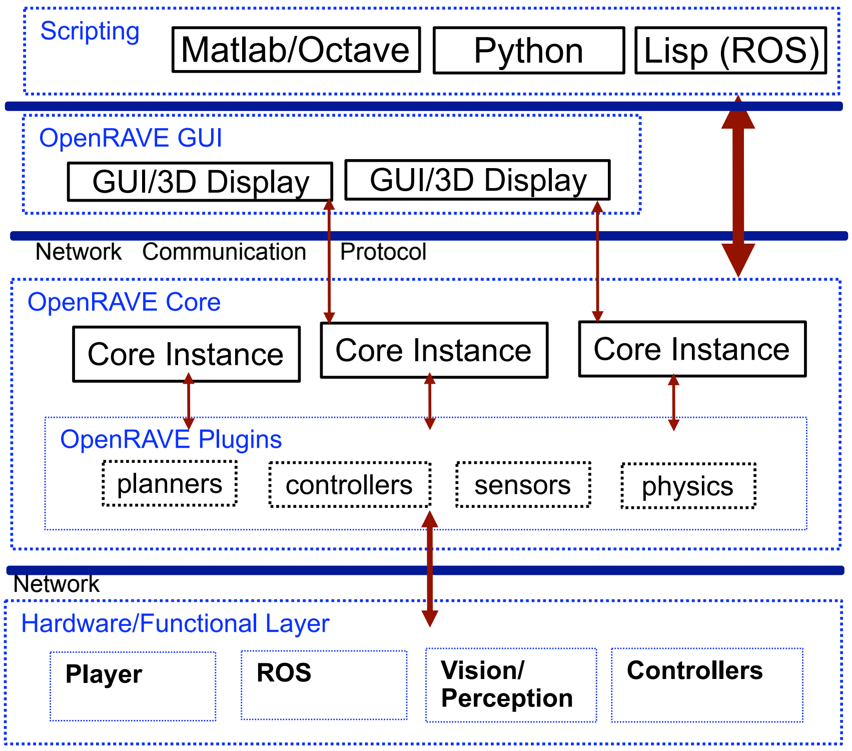


Figure 3.2: The OpenRAVE architecture is divided into several layers: the scripting layer, the GUI layer, the core OpenRAVE layer, and the plugin layer that can be used to interact with other robot architectures.

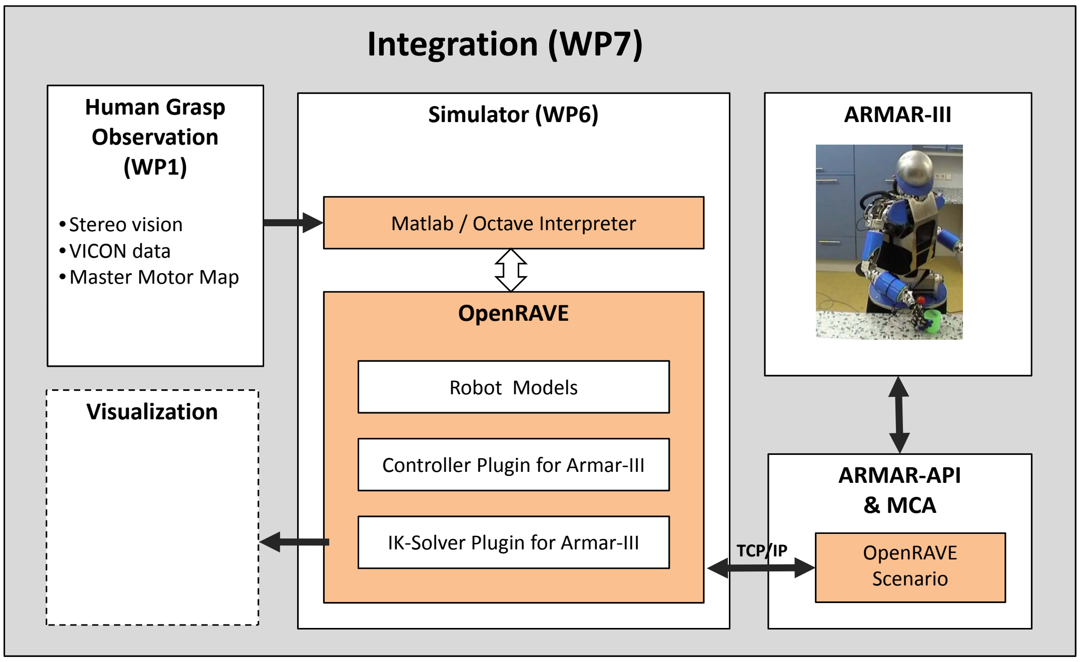


Figure 3.3: Connecting ARMAR-III to OpenRAVE. Communication was established using a TCP based bidirectional protocol for exchanging control commands and perceived joint and end-effector positions.

At the time of writing, this architecture is discussed of been chosen as the standard protocol for simulator interfaces.

3.2 Software Interfaces for the GRASP Simulator

The idea of using the simulator, which is being developed in WP6, as a tool to predict the consequences of an action and to evaluate the performance after their execution requires a thorough and comprehensive specification of the simulator's interfaces. Fig.3.4 shows the simulator's interfaces at different levels of abstraction. On the Inter-Process Communication (IPC) level there are three interfaces, which must be defined:

- **Update the current scene state *(input)*:** Changes in the real world recognized by the platforms perception have to be transmitted to the simulator. Then the internal world state representation in the simulator will be synchronized with the perceived real world state which increases the accuracy of the predicted simulation outcome.
- **Actions execution *(input)*** Whenever an action is applied to the real world, the simulator can be used to predict its outcome. Hence, the simulator needs to supply its clients with an interface that allows the definition and execution of actions.
- **Prediction of action outcomes *(output)*.** After each simulation cycle the clients will expect a change of state in the simulated world model for further analysis and processing. This interface is very similar to the input of the scene state.

A common standard for these interfaces has yet to be defined during the process of development of the simulator. A candidate for the common communication is currently a subject of discussion (see Section 3.1.2). With the definition of a common IPC it is possible to remove the visualization from

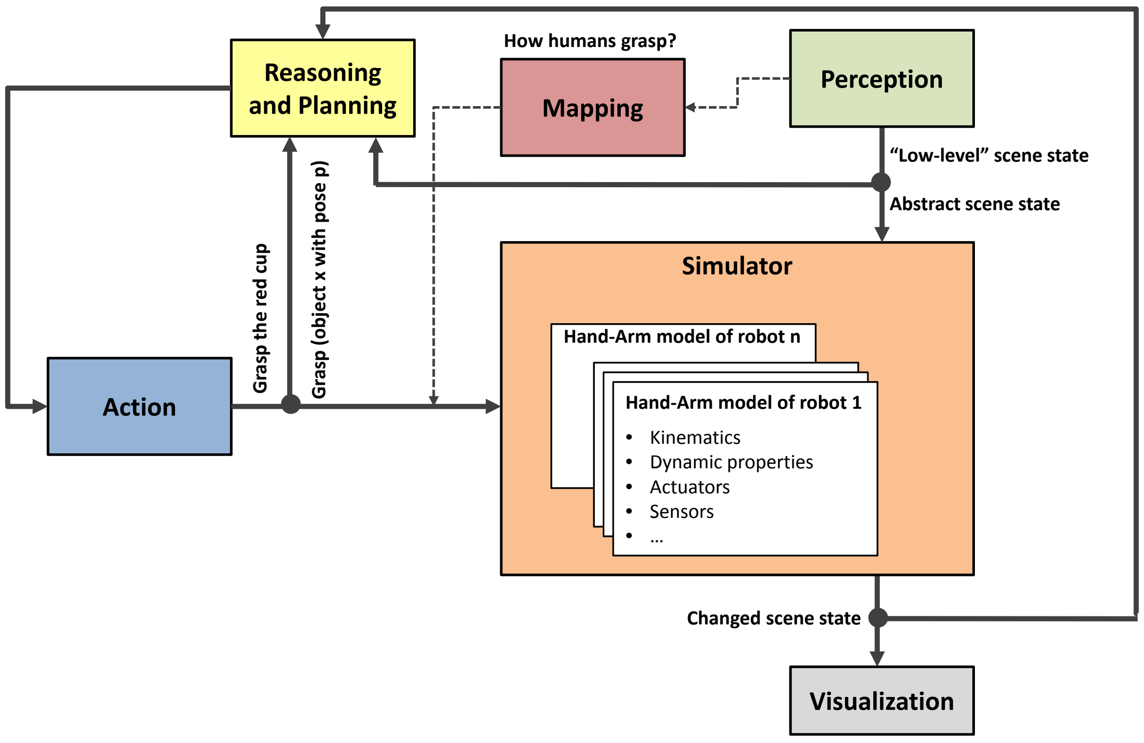


Figure 3.4: The simulator in the center of the architecture: Interfaces at different levels of abstraction are shown in a schematic way. The scene state provieded by the perception component and the actions required to perform a task required as input to the simulator. The expected outcome on an action is output of the simulator.

the simulator and incorporate it in a client on its own. This aspect is discussed in greater detail in Section 5.2.3.

In addition to the three presented runtime interfaces, a fourth interface used for both input and output is necessary as the state of the world and the robot models should be separated from their geometric and functional (semantic) descriptions. A consensus regarding the definition of this interface based on the standardized file format COLLADA (see Section 5.3) has been achieved through intensive discussion and in collaboration between UniKarl and UJI.

3.3 Interfaces for Vision on ARMAR

To achieve successful integration of vision modules on the humanoid robot ARMAR-III, an interface is needed that does not only grant access to the vision system of the humanoid, but also includes image processing capabilities in order to simplify and standardize the development of new vision software components. Based on the Integrating Vision Toolkit (IVT)[3], UniKarl defined such an interface, which is provided to all project partners.

The IVT is a computer vision library that was developed at UniKarl, allowing to start the development of vision components within minimum time. We have experienced that even though in theory many problems in computer vision have been solved for many years, in practice there is no library implementing the corresponding algorithms in a complete toolkit. Thus, usually each partner starts to develop their own library, offering the functionality they need. In addition to the enormous additional implementation effort, this approach would lead to crucial problems when wanting to put the modules from the partners together into an integrated system. Among the problems solved and functions implemented by the IVT are:

- Integration of various cameras and other image sources via a clean interface.
- Providing a generic and convenient application for calibrating single cameras and stereo camera systems.
- Undistortion and rectification.
- Various filters and segmentation methods.
- Efficient mathematical routines, especially for 3-D computations
- Stereo reconstruction.
- Particle filtering framework.
- Platform-independent multithreading
- Convenient visualization of images and the integration of a library for the development of Graphical User Interfaces for which Qt was chosen.
- Full support for the operating systems Windows, Linux, and Mac OS X.

Using IVT for the implementation of vision components allows us to develop vision components that are platform- and camera-independent. This is crucial for a successful integration on the demonstration platform.

3.4 Interfaces for Haptics on ARMAR

The latest revision of the anthropomorphic five-fingered hand of ARMAR-III has been extended with a multimodal haptic sensor system and an increased number of individually controlled joints. In detail, this comprises:

- Position sensors at all active and passive joints.

[3] `http://ivt.sourceforge.net`

- Eight actively actuated DoF, partially conducted as coupled joint actuators.
- Pressure sensors for all DoF of the hand.
- Leakage estimation for each DoF as part of in-system diagnostics.
- Model-based force estimation for each DoF from pressure and position sensor measurements.
- Interface to I^2C-based tactile sensor modules.

The software interface is implemented as part of the MCA (see Section 2.2). Due to the availability of new sensors, the interface supports hybrid force-position control for the eight DoF of the hand. Furthermore, the current configuration of the hand and the actuator force measurement values are available from the sensor readings. Contact force readings are also available in the MCA interface coming from I^2C tactile sensor modules, which are installed on the hand. A kinematic model of the hand is integrated in the MCA module, which provides the complete forward kinematics of the hand. Therefore, it is possible to query the location of parts of the hand or distance between parts, e.g. phalanges. Besides direct joint angle trajectory control the hand may be controlled via Virtual Model Control (VMC) [BRAD08] using a physical model, which allows direct setting of target locations for the fingertips.

3.5 Interfaces for Actions

Due to the variety of action representations, which arise from the use of different methods and systems, the definition of a common interface in order to enhance action data exchange is inevitable. The need for a standard interface emerges clearly, when transferring data from various motion perception modules to different robot and action recognition systems. Each module produces data in terms of its own specific model and format, respectively. Hence, one has to deal with a variation of different data formats. Likewise, for reproduction of movements, each robot system requires data in terms of its own kinematics.

Therefore, a standardized interface is established by using the Master Motor Map (MMM) [AAD07b], which features a high level of flexibility and combability. The MMM provides a reference kinematic model by defining the maximum number of DoF that can be used by a perception module and a robot. The arrangement of the joints and the orientation of their axes is based on the H-Anim[4] version 1.1 specification. A depiction of the MMM model is illustrated in Fig. 3.5. In order to represent grasping and manipulation activities, for each hand, the MMM has been extended by 3 DoF for finger and thumb control as well as the tool center point position and the measurements of the manipulator. Based on the MMM, movements can be described as trajectories in joint space consisting of 58-dimensional vectors, each vector describing a joint angle configuration with a floating point number for every single DoF as well as trajectories in Cartesian space. The description of the DoF implemented in the MMM and the equivalent joint descriptions for a humanoid robot are given in Table 3.1.

Due to differences in the Euler conventions, the availability of joints, and the differing order of the joint angle values, a conversion module has to be implemented for each system in order to provide a proper connection to the MMM. This conversion module transforms the module specific data into the MMM file format or vice versa. By implementing a conversion module from the MMM to the humanoid robot ARMAR-III, an interface has been established which enables the reproduction of actions in the MMM format. Furthermore, conversion modules for the connection of the marker-based human motion capture system by VICON[5] and the markerless human motion tracking system [AAD08] developed at UniKarl have been implemented. The framework that connects these two different motion capture systems to ARMAR-III is depicted in Fig. 3.5.

[4]`http://www.h-anim.org`
[5]`http://www.vicon.com`

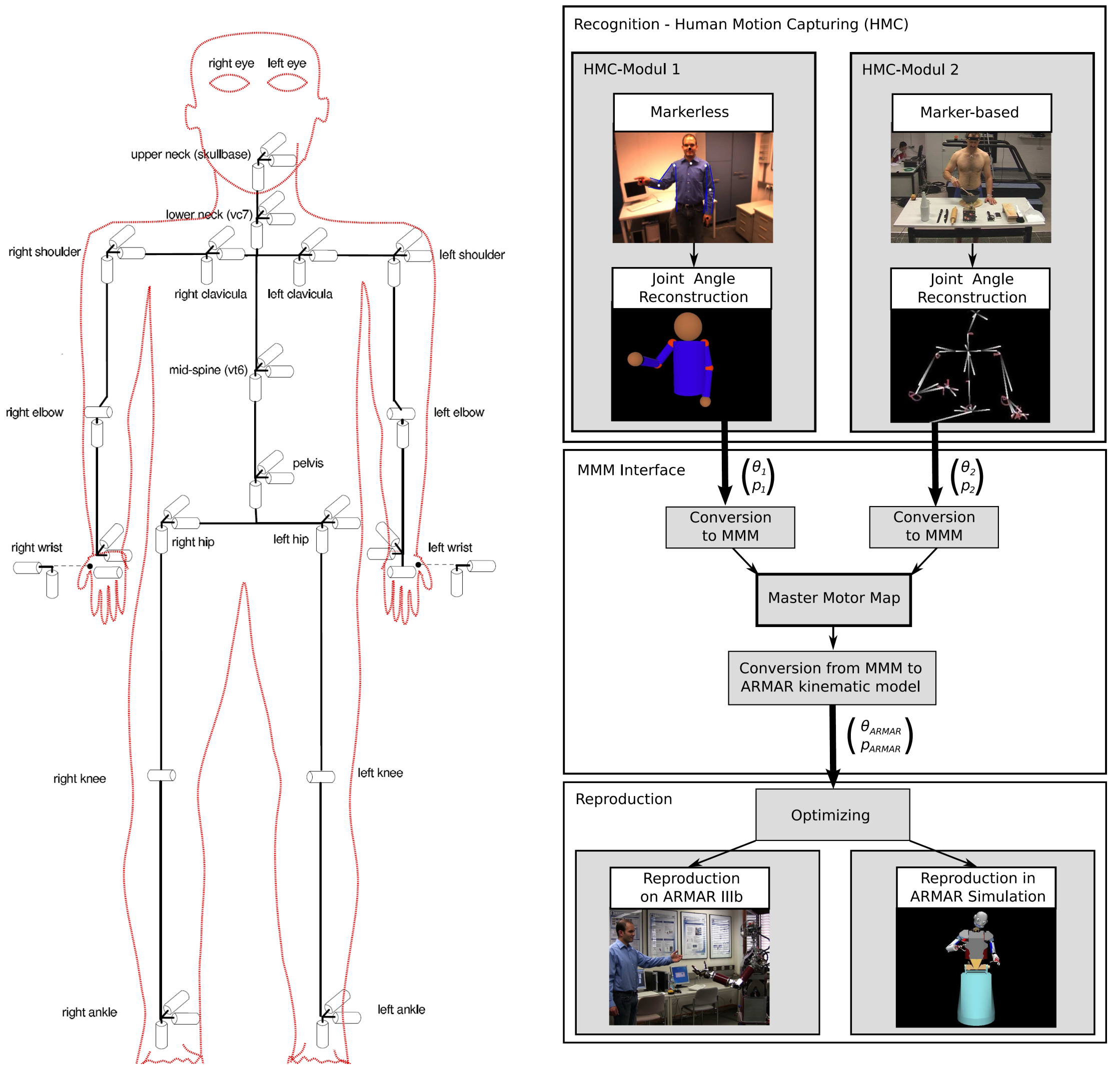


Figure 3.5: Left: The MMM kinematic model including the joint axis of each DoF. Right: MMM interface connecting a marker-based and a markerless human motion capture system to the humanoid robot ARMAR-III.

		DoF	Euler angles	DoF description
HumanoidRoot	Translation	3		
	Rotation	3	$R_{X'Z'Y'}(\alpha,\beta,\gamma)$	
Neck	skullbase	3	$R_{X'Z'Y'}(\alpha,\beta,\gamma)$	nod (SBN) tilt (SBT) rotation (SBR)
Torso	vc7	3	$R_{X'Z'Y'}(\alpha,\beta,\gamma)$	flexion/extension (VCFE) adduction/abduction (VCAA) rotation (VCR)
	vt6	3	$R_{X'Z'Y'}(\alpha,\beta,\gamma)$	flexion/extension (VTFE) adduction/abduction (VTAA) rotation (VHR)
	pelvis	3	$R_{X'Z'Y'}(\alpha,\beta,\gamma)$	flexion/extension (PFE) adduction/abduction (PAA) rotation (PR)
Hip	l_hip r_hip	$3+3$	$R_{X'Z'Y'}(\alpha,\beta,\gamma)$	flexion/extension (HFE) adduction/abduction (HAA) rotation (HR)
	l_knee r_knee	$1+1$	$R_{X'Z'Y'}(\alpha,0,0)$	flexion/extension (KFE)
	l_ankle r_ankle	$3+3$	$R_{X'Z'Y'}(\alpha,\beta,\gamma)$	flexion/extension (AFE) adduction/abduction (AAA) rotation (AR)
Shoulder+Arm	l_sternoclavicular r_sternoclavicular	$3+3$	$R_{X'Z'Y'}(\alpha,\beta,\gamma)$	flexion/extension (SCFE) adduction/abduction (SCAA) rotation (SCR)
	l_shoulder r_shoulder	$3+3$	$R_{X'Z'Y'}(\alpha,\beta,\gamma)$	flexion/extension (SFE) adduction/abduction (SAA) humeral rotation (SHR)
	l_elbow r_elbow	$1+1$	$R_{X'Z'Y'}(\alpha,\beta,0)$	flexion/extension (EFE)
	l_wrist r_wrist	$3+3$	$R_{X'Z'Y'}(\alpha,0,\gamma)$	flexion/extension (WFE) adduction/abduction (WAA) rotation (WR)
Hand	l_finger r_finger	$1+1$	$R_{X'Z'Y'}(\alpha,0,0)$	flexion/extension (FFE)
	l_thumb r_thumb	$2+2$	$R_{X'Z'Y'}(0,\beta,\gamma)$	adduction/abduction (TAA) rotation (TR)

Table 3.1: First column: Name of body part. Second column: Joints related to the body part. Third column: Number of degrees of freedom. Fourth column: Euler angle conventions for the joints of the MMM. Fifth column: Equivalent joint description for a humanoid robot.

Chapter 4

Scene Representation and Object Models

4.1 Scene Representation

To deal with object manipulation and grasping, a robot needs knowledge about both the objects and the environment in which the objects are located. When manipulating one object, other objects in the scene are relevant, too, and have to be taken into account for the execution of a manipulation task. The robot therefore needs a representation of the scene, which describes not only fixed parts of the environment like furniture and room geometry, but also the poses of objects.

For the humanoid robot ARMAR-III, a light-weight scene graph module which can be used to store the kind of information necessary for the execution of complex manipulation tasks was developed. The module is light-weight in the sense that it was specifically designed for the task of storing environment information for a robot, compared to full-fledged scene graph libraries usually intended for visualization purposes. The scene graph module allows fast queries against geometry properties of nodes in the scene, like collision and proximity queries, accelerated through hierarchical bounding volumes (axis aligned and non-axis aligned bounding boxes). Queries can be made for properties of objects in the scene, like a search for certain objects or classes of objects.

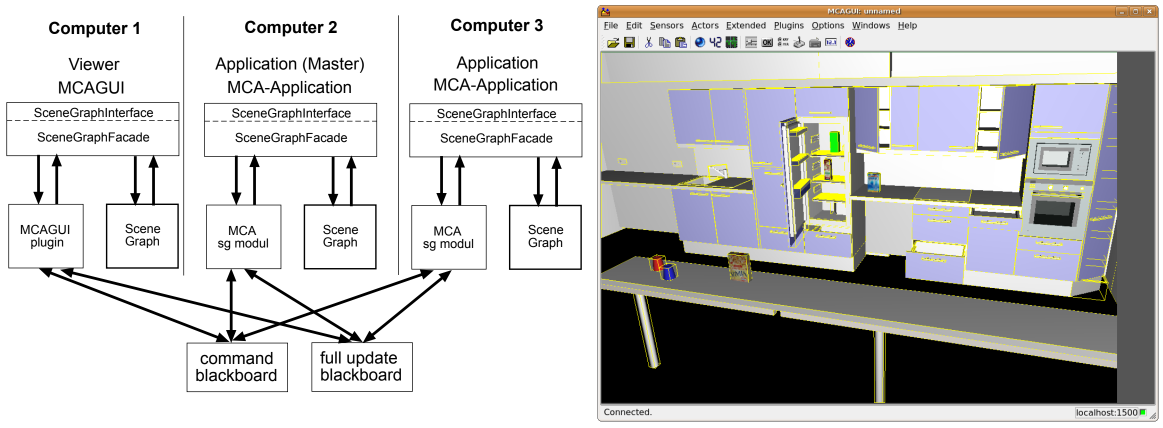


Figure 4.1: Left: The connection of several scene graph modules through MCA using two MCA-blackboards for communication. The scene graph facade hides the synchronization mechanisms and offers a simple interface. Right: Visualization of the scene graph of the kitchen through the `mcagui` application, showing several objects with their bounding boxes. The support surfaces within the fridge are marked in yellow.

The scene graph is integrated into MCA, which enables the scene graph to be distributed over several computers (see Fig. 4.1). A scene graph module automatically synchronizes its state with the other scene graph modules connected through MCA, similar to a cache coherency protocol, which ensures that the local version represents the current state of the environment. The synchronization is only triggered if the scene state is changed, for instance through new percepts. Therefore, queries on the scene graph are very fast, since they are executed locally.

The scene graph can provide abstractions of the objects in the scene as axis aligned and non-axis aligned bounding boxes, which is usually enough for approximate motion planning for grasp execution. The actual geometry of the objects is not stored in the scene graph. Only a reference to the actual geometry is stored, which is used for high-quality visualization and precise grasp planning. The scene graph module can generate and maintain an OpenInventor scene graph, which is used for visualization and export to other geometry formats. The state of the scene graph is completely accessible at runtime through a C++ API.

The scene graph module is used by ARMAR-III in a kitchen scenario for representing the kitchen environment with drawers and doors, as well as the objects that have been detected by the vision system. The module can be used in offline simulations as well; for this purpose, the pose of non-environment objects has to be supplied through a simulated vision process or by the simulation parameters. Motion planning experiments using Rapidly-exploring Random Trees (RRTs) [LaV06, KL00] have been executed using the above-mentioned scene representation, as presented in [VAD07].

4.2 GRASP Object Database

4.2.1 Initial Object Set

To enhance interchangeability of implemented methods and interaction between different systems, all project partners were asked to work with the same identical object set. Hence, a common object set has been defined consisting of items that can be found in an ordinary household. Regarding size and weight the selected objects should have adequate measurements allowing the available manipulators in the project to grasp and lift the objects. However, since one of the main goals of the project is the exploration of different grasp types, the set should contain complex-shaped objects and objects that can have different states (e.g. for a cup: filled or empty). UniKarl has purchased and distributed at least one object set to every project partner. The initial object set is described in Table 4.1 and shown in Fig. 4.2.

Object name	ID	See Fig. 4.2
Detergent bottle with handle	27	b)
White cup with handle	28	c)
Green cup with handle	29	d)
Spray bottle	30	a)
Chocolate box	31	f)
Bread box	32	i)
Zylindric salt box	3	h)
Rectangular salt box	33	g)
Measure cup	44	e)

Table 4.1: Initial object set. Left column: Object name. Center column: Object ID in the database. Right column: Reference to the figure illustrating the object.

4.2.2 Acquisition of Object Models

In addition to the object set, UniKarl has provided all project partners with images and 3D models of the objects, which were acquired with the object modeling center that has been established within the Collaborative Research Center 588[1] (see also [KBSD07, BSZD06]). Models of objects with a simple geometric shape, like boxes or hexagonal shapes, have been created by hand to achieve maximum accuracy. For the modeling of complex objects in the GRASP object set, a high accuracy laser scanner and a stereo camera were used. To ensure optimal and consistent measurements of different views several light sources

[1]www.sfb588.uni-karlsruhe.de

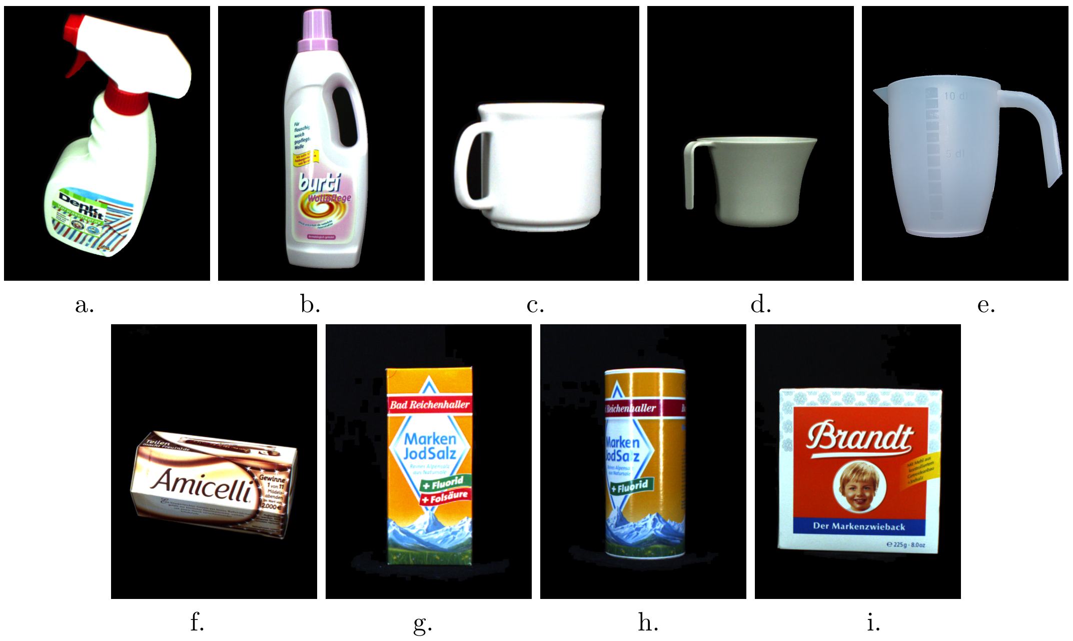


Figure 4.2: Object views from the left camera of the Marlin stereo camera system: a) Spray bottle, b) detergent bottle with handle, c) white cup with handle, d) green cup with handle, e) measure cup, f) chocolate box, g) rectangular salt box, h) cylindric salt box, and i) bread box.

and a rotation plate are installed within the modeling center. During an object modeling process, the specific settings of each frame are stored as depicted in Table 4.2. The object modeling center is shown in Fig. 4.3.

The modeling center uses two main capture devices:

Laser Scanner A high-accuracy laser scanner Minolta VI-900 was used for the acquisition of 3D point clouds. The objects are put on the rotation plate and turned in front of the laser scanner, and a point cloud is taken from different angles. The point cloud of the scene consists of 307,200 points, whereas depending on the object size approx. 30,000-70,000 points are relevant to reconstruct the surface of the object. The acquired 3D data is then post-processed, by means of registration, triangulation, cleanup, and reduction. This ensures the generation of high quality meshes with different resolutions. In order to provide compability, the point clouds and meshes are provided in different formats:

- OpenInventor V2.1 ASCII format (.iv)

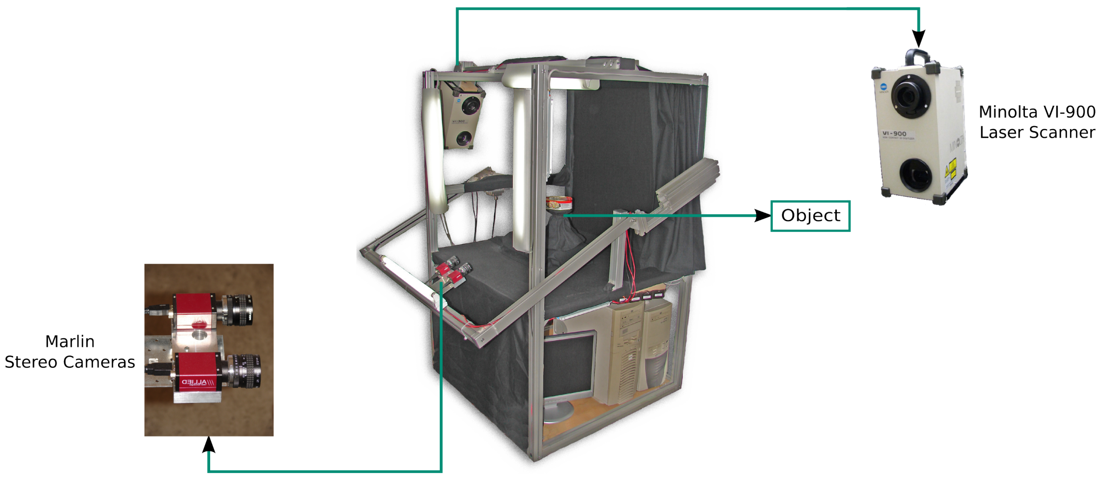


Figure 4.3: Modeling center using the Minolta VI-900 laser scanner and the Marlin stereo camera system.

Setting	Option	Description
Scanner	yes=1/no=0	Laser scanner used in this frame
Picture	yes=1/no=0	Stereo cameras used in this frame
IselTurntable	0-353	Current angle of the turntable in degree
Lightsource_x	0-5	Current brightness of light source x (x = one of the four available light sources)
Powercube	-25-75	Current angle of the camera arm in degrees
Right_picture	*filename*	Filename of the image captured by the right camera
Silhouette	*filename*	Name of the silhoutte file describing the automatically generated silhouette of the object
CameraPosition	$t \in \mathbb{R}^3$, $R \in \mathbb{R}^{3\times3}$	Describes the camera position in the 3D coordinate system via a translation t and a rotation matrix R
Preview_Picture	*filename*	Lower quality jpeg version of the left camera image for preview purposes
Left_picture	*filename*	Filename of the image captured by the left camera
Pointcloud	*filename*	Point cloud file taken during scan
Resolution	0=high/1=low	Resolution of the scan
Reduction	yes=1/no=0	Level of automatic noise reduction
NumberOfPoints	0-307200	Number of vertices of the point cloud/mesh
Mesh	*filename*	Triangulated mesh file, OBJ=Wavefront obj / OIV=VRML, NumberOfPolygons describes the number of triangles/quads

Table 4.2: Settings of the measurements valid during a single object modeling frame.

- VRML V1.0 format (.wrl)
- Wavefront OBJ format (.obj)

In Fig. 4.4 examples of rendered objects are shown. For the registered and triangulated meshes a texture is generated from the stereo camera views of the object. This is provided along with the .obj-files, which contain a matching texture mapping.

Color Stereo Cameras In order to provide visual data corresponding to the generated object models, color stereo camera images are taken from different angles. The different views are generated by rotating the object on a turn table and moving the camera system which is mounted to a rotating bracket which can turn up and down (between about -25 and +75 degrees). This allows us to acquire a broad range of views, e.g. into a cup from above. The object views were retrieved with the following rotation ranges and step widths:

- 360 degree horizontal rotation range, 10 degree stepwidth (of the rotation plate)
- 100 degree vertical rotation range, from minimum -25 degrees to maximum +75 degrees, with 10 degree step width (of the rotation bracket with the stereo camera system)

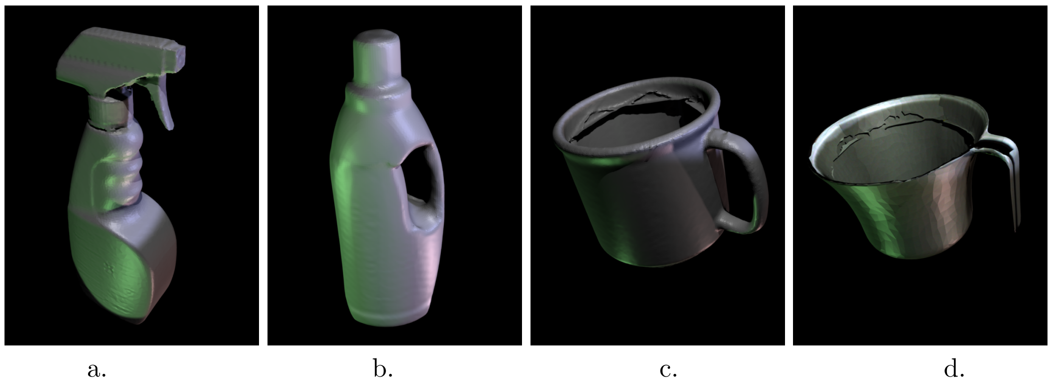


Figure 4.4: Rendered 3D models for complex-shaped objects: a) Spray bottle, b) detergent bottle with handle, c) white cup with handle, and d) green cup with handle.

The images are captured with a Marlin stereo camera in a RGB8 data format at a resolution of 1392 x 1038 pixels. Intrinsic and stereo calibration data files are provided to all project partners.

4.2.3 XML Representation

Along with the 3D and image data additional information about the object and the model acquisition process is provided by an XML file. It contains the filenames of the scans and images along with camera positions for each image. This file format will be extended to contain other information such as material and weight.

At this point, the XML file contains a specification of the object model acquisition process. The system provides the possibility to acquire 3D data in form of scans and image data through the cameras. Each scan or image capture is a single job for the system. Hence, the digitization of a whole object requires several jobs. Each job is represented as an XML file and the data gathered during the job is described therein. The following description of the tags is used:

- ObjectModel: encapsulates a whole object consisting of several jobs.
- Filename: the filename of the XML job file.
- Measurement: the settings during the modeling process as listed in Table 4.2.

For convenience of the other project partners, a sample implementation for parsing the XML job files using the tinyXML parser[2] is provided by UniKarl.

The XML files and the corresponding 3D and image data contain a lot of information about the shape and appearance of the objects. Currently the simplified textured models are used for visualization and the high resolution point clouds are used for box decomposition. The additional information is provided to enable the development of new methods for object detection and localization and object grasping. Currently, we transfer the information from the modeling center to the scene graph manually and add known attributes, but we are working towards the completion of this pipeline to directly use the data from the modeling center and to derive object attributes through cognition, reasoning and instruction. We have elected COLLADA as the carrier format for all this information, as will be described in Section 5.

4.3 Object Representations

Depending on the specific application, an object can be represented as a 3D model consisting of point clouds and meshes as well as in form of camera views of the object. In Table 4.3, different object representations including are given.

Representation	**Source**	**Usage**
Point cloud	Object Modeling Center	Grasping based on box decomposition
Triangle mesh	Point cloud, simplified	Collision detection and visualization
Textured mesh	Additional textures	Visualization (vision simulation possible)
Vision data	Real images, synth. views	Object detection and localization

Table 4.3: Different object representations, their source and usages.

Object models are currently generated offline, either by hand or in an object modeling center with stereo cameras and a depth imaging device (see Section 4.2.2). This results in high resolution point clouds and meshes (400,000 triangles), which are turned into meshes of smaller size for collision detection and visualization (800 - 30,000 triangles). The point clouds are directly used by the box decomposition method (see Section 6.4). Textures have been created for the textured objects, based on the stereo images from the object modeling center. Fig. 4.5 shows a sequence of different models of the same object.

Independently, SIFT features are extracted from real images for textured objects to allow detection and localization of these objects. For objects that can be segmented by color, PCA-compressed synthetic

[2] `http://sourceforge.net/projects/tinyxml`

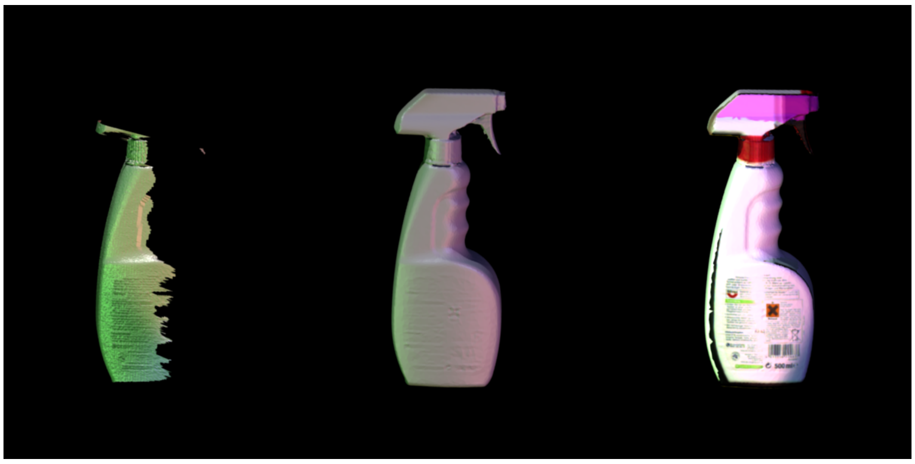
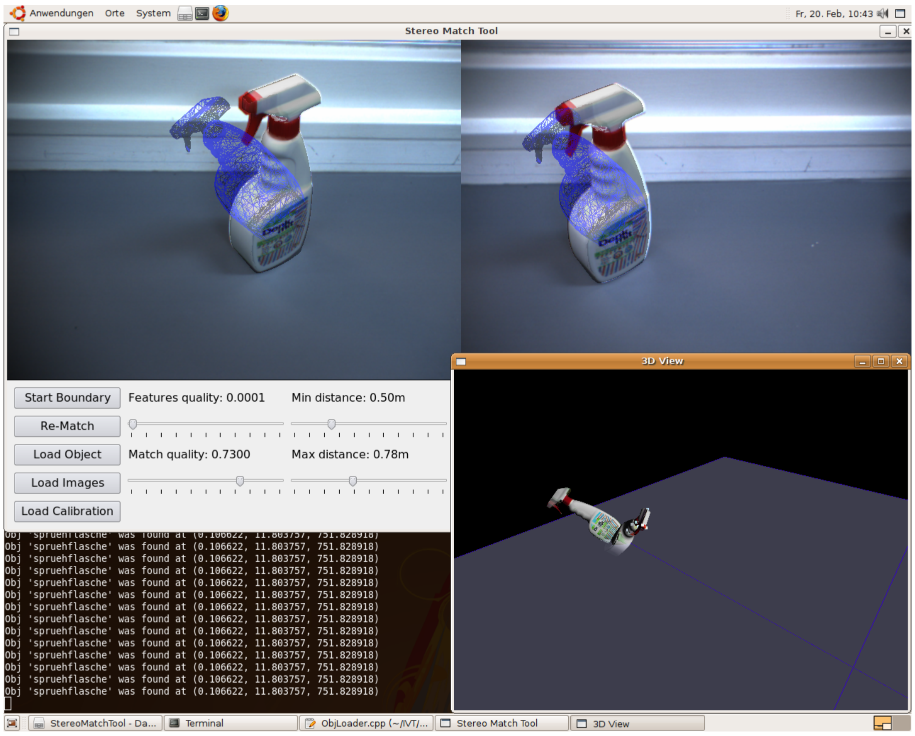


Figure 4.5: Left: Shows the partial mesh as generated from a single scan with the object modeling center. Center: The fusion of all partial scans creates a high resolution mesh. Right: A reduced textured mesh.

views are generated with the mesh representation. Since the pose computed by the vision system is expressed in terms of its own object representation, the constant rigid transformation between the two object representation has to be calibrated. With an interactive tool, which visualizes the objects' 3D model in the stereo images, this transformation is calibrated manually. A screenshot of the GUI of the tool is shown in Fig. 4.6. The used object recognition and pose estimation system as well as the interactive tool are described in D6.



Features quality: 0.0001
Min distance: 0.50m
Match quality: 0.7300
Max distance: 0.78m

```
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
Obj 'spruehflasche' was found at (0.106622, 11.803757, 751.828918)
```

Figure 4.6: The matching tool uses manual controls to determine the transformation from the model coordinate frame (lying wireframe model) to the object recognition frame (the correct standing model).

4.3.1 Object Attributes

When an object is placed in the scene a new node is added to the scene graph. This node has further attributes besides their geometric properties, such as pose, bounding box size and a precise geometric model (external reference). The attributes that are currently implemented are:

- name – human readable name of the object, e.g. red cup
- type – category that describes the use of the object, e.g. food, detergent
- common location – where to find/place the object usually
- handling strategy – known grasping strategies for the object
- support surface – whether something can be placed on the object (used for environment objects like work surfaces)

These attributes represent the subset of attributes that are needed for regular manipulation tasks. However, this list is extensible and new attributes will be added as the capabilities of the robot emerge. We store the attributes of known objects and load these at runtime as a-priori knowledge. All attributes can be modified at runtime through perception, reasoning or interaction with an instructor. An 'apple juice' object would be described as in Fig. 4.7 with further attributes as described in Section 2.3 from D4. The pose of the object in the scene is described through a relative rigid transformation and a parent coordinate frame in the scene graph.

```
name                = applejuice
class               = food
bbox                = (-35.50 -119.00 -31.50)-(35.50 119.00 31.50)
weight              = not_implemented
inertia_tensor      = not_implemented
texture             = not_implemented
inventor_file       = applejuice.iv
common_location     = fridge
handling_strategy   = regular_grasp
grasp_points        = not_implemented
```

Figure 4.7: Example object attributes as stored in a file for the scene graph.

A simple form of geometric reasoning is used, when searching for a position to place an object: The common location attribute of the current object is examined, which is a reference to another node in the scene graph. The nodes within the bounding box of the common location are checked for the support surface flag, which classifies the top surface of such a node as a potential position to place the object. This reasoning is enabled through storing the necessary information in the scene graph and using simple geometric queries.

Chapter 5

GRASP Robot Editor

5.1 Concept and Features

The main goal of the GRASP Robot Editor is to allow the convenient integration of all the various robots used in the project in the simulator. Therefore, a standardized tool called Robot Editor has been developed, which allows the creation of geometrical, kinematic and dynamic robot models as well as their convenient conversion from a variety of CAD formats has been developed. The design principles of the Robot Editor are:

- **Conversion:** Robot models – if available – are usually given in a variety of different file formats that need to be converted to the format used in the simulator.
- **GraspIt! compatibility:** Being an already widely used standard with many conform models available, robot models used in GraspIt! should be convertible into the format used by the simulator.
- **Geometric modeling:** With the integration of new robots also the need for a tool arises that can be used for modeling of the geometrical components (meshes) in a convenient way.
- **Semantic modeling:** More important than the geometric modeling is the ability to describe semantic properties of the robot manipulator, i.e. definition of kinematic chains, sensors, actuators, etc. or even specify algorithms.
- **Dynamics modeling:** Another important aspect is the ability to define physical attribute of the robot's elements and the objects of the scene. At the moment, however, the focus lies solely on the dynamics of rigid bodies.

To our knowledge, there is no existing solution that could meet all these requirements. This is why we decided to develope the Robot Editor based on available open source software. The conceptual design of the Robot Editor hence relies on two technologies that will be presented in the following sections.

5.2 The Blender Project

5.2.1 Framework for the Robot Editor

Being a very powerful and extensible 3D editor, the *Blender* project[1] was chosen among several candidates to serve as the framework of the new Robot Editor because of

- built-in support for many commonly used CAD formats as well as basic support for kinematic structures,
- convenient 3D modeling,

[1]`http://www.blender.org`

- the capability of ray-tracing allows the production of high quality images and sequences that may be used later on, e.g. for generation of hand poses for recognition (see Fig. 5.2-c).
- easy extensibility via the Python scripting language.

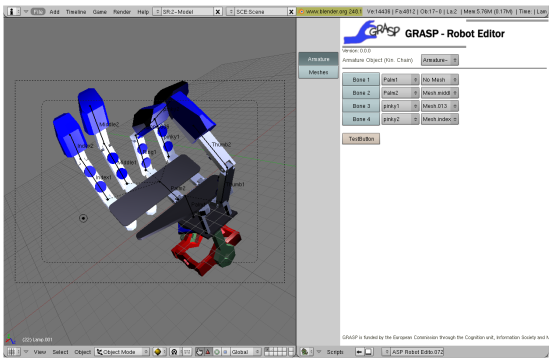
However, Blender on its own cannot be used to generate the input for the simulator. This is due to the fact that the creation of a robot model from scratch or even the connection of existing parts becomes a cumbersome and unintuitive task (from a robotics point of view) that requires the user to undergo a long initial adaption period. The key points lacking in blender are:

- There does not exist any way for defining axes by means of established conventions such as Denavit-Hartenberg parameters. Usually, axes are placed manually decreasing the overall accuracy of the model.
- There is no support for the required custom meta information, i.e. sensors, actuators and algorithms.
- Highly sophisticated dynamics information can not be augmented to the robot.
- Conversions between certain file formats need to be improved or implemented, namely the import of GraspIt! robot models and their underlying geometrical representations, which require manual adjustments and the file format used for the editor itself (see Section 5.3.2).

On the other hand, Blender has been chosen because it comes with the advantage of being easily extendable via the Python scripting language, which allows to overcome the shortcomings mentioned above. Using the scripting language, one has access to all relevant data structures in Blender. The main task of transforming Blender into a powerful Robot Editor hence requires at first the creation of an intuitive and user- friendly user interface with a focus on robotics applications.

The Robot Editor will contain and provide interfaces for essential robotics algorithms, e.g. the computation of dynamics characteristics from the geometric meshes and conversions between kinematics representations. Adjacency information of joints and the impact of joint movements to the robot are additional computational information, which is useful for developers planning algorithms.

A screenshot of the ongoing development of the editor and a rendered image of the Karlsruhe hand performing a precision grip can be seen in Fig. 5.1-a. Although it is still work in progress, the editor already significantly simplifies the task of creating functional robot models.


www.blender.org 248.1 Ve:14436 | Fa:4812 | Ob:17-0 | La:2 | Mem:5.76M (0.17M) | Time: | Lam

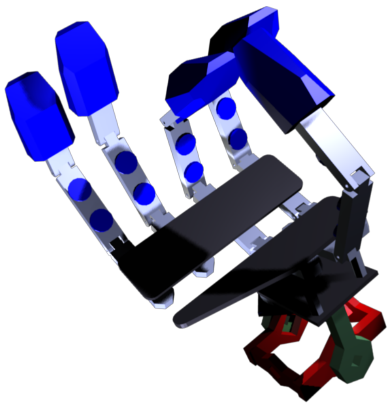
GRASP - Robot Editor

Version: 0.0.0

Armature Object (Kin. Chain) Armature-

Bone 1	Palm1	No Mesh
Bone 2	Palm2	Mesh.middl
Bone 3	pinky1	Mesh.013
Bone 4	pinky2	Mesh.index

a.



b.

Figure 5.1: a) Screenshot depicting the creation of a model of the Karlsruhe hand showing the initial user interface of the GRASP Robot Editor. b) Rendered image of the functional model of the Karlsruhe hand with eight active degrees of freedom (see Section 2.1).

5.2.2 Robot Models in Blender

In order to explore the requirements of the Robot Editor, the development is linked to the creation of an initial set of robot manipulators. For this purpose, UniKarl received hand models from project partners. Models for the following hands are being created during the development of the editor:

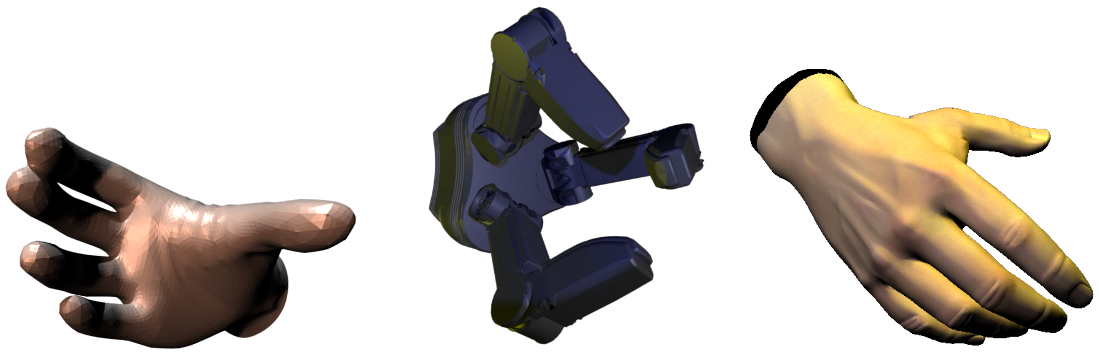


Figure 5.2: Rendered image of the hand produced with the GRASP Robot Editor and showing geometrical robot models used in GRASP: a) the Otto-Bock hand, b) the Schunk hand, and c) a human hand model used for hand pose estimation at KTH.

- the Karlsruhe hand (Fig. 5.1-b),
- the hand prothesis from Otto Bock (Fig. 5.2-a),
- the Schunk hand used at KTH (Fig. 5.2-b), and
- the model of a human hand, which is currently used for hand pose recognition at KTH (Fig. 5.2-c).

All these models will be available in the early stages of the simulator.

5.2.3 Client for Visualization

Finally, the extensibility of Blender includes the creation of graphical runtime applications using the so-called *Game–Blender engine*. This framework will be used in the future for the creation of a visualization tool for both robot simulation and execution. A separation of simulator and the visual representation brings a lot of advantages such as multiple views broadcasted over the network and saving computational resources when the simulator runs directly on the robot. Blender's framework already possesses impressive graphical abilities and can – due to scripting capabilities – be extended by interactive elements and then be natively connected to the software interface proposed in Section 3.1.

5.3 COLLADA

5.3.1 Scene and Robot Models in COLLADA

The COLLADA [Bar06][2] file format in order to exchange both robot and scene models between the Robot Editor and the simulator. COLLADA was chosen for the following reasons:

- **Standardization:** COLLADA is already an accepted standard by the Khronos consortium[3] that is actively developed and finds wide acceptance and support in CAD, game design and industrial applications.
- **Open source:** COLLADA is completely open and royalty free.
- **XML-based:** Being a XML based file format facilitates the handling and processing in applications.
- **Kinematics support:** Since version 1.5 the definition of kinematics is officially supported by the standard.

[2] `http://www.collada.org`

[3] The Khronos Group was founded in January 2000 by a number of leading media-centric companies, including 3Dlabs, ATI, Discreet, Evans & Sutherland, Intel, NVIDIA, SGI and Sun Microsystems, dedicated to creating open standard APIs to enable the authoring and playback of rich media on a wide variety of platforms and devices, `http://www.khronos.org`.

- **Availability:** There are many tools and visualization frameworks already available including open source projects.
- **Extensibility:** The file format can be extended to serve the needs of different applications.

The integration of the standard into the Robot Editor including the definition of extensions to the standard are presented in the following section.

5.3.2 Compatibilty and GRASP-specific extensions

As already mentioned, the COLLADA industry standard was elected by UniKarl and UJI as the only file format to be accepted by the simulator. The acceptance as an industry standard and the wide acceptance in addition to a clear and extensible design lead to this decision. With the introduction of version 1.5 in August 2008 the standard was officially augmented by useful language constructs dedicated to kinematic chains that can be used directly for robot descriptions. The COLLADA import and export script available with the main Blender distribution (see Section 5.2), however, was designed to accept only COLLADA documents of version 1.4 and does not allow the inclusion or modification of the user-definable descriptions needed by the simulator. This leads to the initiation of the development of enhanced COLLADA compatibility that eventually will flow back to the open source community.

In close collaboration with the original developers of OpenRAVE the integration of COLLADA as the standard file format has been accepted and the various aspects of the file format definition are discussed and clarified in an ongoing discussion. COLLADA is an XML-based file format and enables and encourages developers to extend the specifications to their needs without having to violate the underlying schema definition. Every file entry can hereby be enriched by additional data that is specified with respect to a certain application. The only restriction to this data is to be a valid XML document itself that can be validated according to an appropriate schema definition together with the complete document.

There is a clear tendency to include specific existing parts of the former OpenRAVE file definitions into the format definition, however within a separate name space, see the listing in Fig. 5.3.

Hence, all simulator specific entries will be hidden to applications other than the simulator without reducing the validity of the file, i.e. it can still be processed and viewed by any application compatible to COLLADA.

A second aspect being actively discussed is to support the representation of semantic information (algorithms, actuator types) and functional data (geometry, dynamics and kinematics). With the Robot Editor also being a powerful candidate for a front-end to combine also semantic information, our proposition is to accept composed files by default and create modular representations when demanded by the user without abandoning valid COLLADA output.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<COLLADA xmlns="http://www.collada.org/2008/03/COLLADASchema" version="1.5.0">
    ...
    <library_joints id="libjoints">
        <joint id="joint_1" name="Revolute">
            <revolute sid="axis0">
                <axis>-1 -0 -0</axis>
                <limits>
                    <min>-90</min>
                    <max>90</max>
                </limits>
            </revolute>
            <extra type="Sensor">
                <technique profile="OpenRAVE">
                    <OpenRAVE:AttachedSensor name="MyFirstLaser">
                        <OpenRAVE:translation>0 0.2 0.4</OpenRAVE:translation>
                        <OpenRAVE:rotationaxis>1 0 0 45</OpenRAVE:rotationaxis>
                        <OpenRAVE:sensor type="BaseLaser2D" args="">
                            <OpenRAVE:minangle>-135</OpenRAVE:minangle>
                            <OpenRAVE:maxangle>135</OpenRAVE:maxangle>
                            <OpenRAVE:resolution>0.35</OpenRAVE:resolution>
                            <OpenRAVE:maxrange>5</OpenRAVE:maxrange>
                        </OpenRAVE:sensor>
                    </OpenRAVE:AttachedSensor>
                </technique>
            </extra>
            <test/>
        </joint>
        ...
    </library_joints>
    ...
</COLLADA>
```

Figure 5.3: Example for how the COLLADA specification can be enhanced by information specific to OpenRAVE via the `extra` and `technique` elements. In this example, a joint element is defined and augmented by a laser scanner sensor. Its definition uses the former OpenRAVE syntax and is declared within an associated name space. Due to this mechanism, the file can still be opened and used by different applications.

Chapter 6

Further Integration Activities

6.1 GRASP Software Subversion Repository (UniKarl, UIJ)

For a smooth and effective development of software components in the project and in particular the development of the GRASP simulator in which several partners are involved (UJI, UniKarl and LUT), a software repository for the software development in the project has been created at UniKarl. For this purpose, the free/open source version control system Subversion[1] is used to operate across networks, and provide access to people on different computers. UniKarl took the leader role in this activity due to the existing experience in administrating and maintaining huge robot software projects.

The long-term goal with GRASP is to have an open source repository which includes all developed algorithms for grasping, manipulation, visualization, and simulation as well as databases for objects, robot arm and hand models and human grasping. The entire software including the simulator software and the above-mentioned tools will be accessible through this repository.

6.2 Administration of GRASP Wiki Online Content (UniKarl)

UniKarl established the internal infrastructure of the GRASP project necessary for

- the discussion of concepts,
- the presentation of results and
- the administration of collectively used information in general.

For this purpose, the GRASP Wiki been created using the open source Wiki engine MoinMoin[2] which allows all editors to easily create and modify, upload content or initiate related discussions in a very convenient way. It is hosted on the web space of UniKarl.

6.3 Visual Data Recording of Human Grasping (UniKarl, LMU)

As part of the integration work, UniKarl installed the stereo camera system at LMU in order to provide WP1 with a multi-sensorial observation of human grasping trajectories. The system consists of two Point Grey Research[3] Dragonfly IEEE-1394 cameras, which are identical to the cameras used in the vision system of ARMAR-III. Successful installation of the camera setup and the electromagnetic Polhemus Liberty system used at LMU was accomplished by taking the following steps:

- Hardware installation of the vision system on a rack.

[1] `http://subversion.tigris.org`
[2] `http://moinmo.in/`
[3] `http://www.ptgrey.com`

- Software installation of needed software components, namely the Integrating Vision Toolkit (IVT) and the Point Grey Research software, in order to get the system running and to provide interfaces to the cameras.
- Implementation of a customized image capture application in order to synchronize the visual data with the recorded grasping trajectories. Based on a TCP/IP communication channel between both systems, trigger signals are sent from the Polhemus tracking system to initiate the image capture process
- Extension of the image capture application to provide access and control of the camera settings. Since color images are produced in order to provide FORTH with skin color information of the hand, the camera parameters of the vision setup at LMU had to be adjusted according to the environmental conditions, which are mainly influenced by poor lighting. Concerning the camera settings, the implemented application has been extended to allow easy access to the parameters, so that these can be changed and modified by LMU according to their needs.

6.4 Grasping based on Box Decomposition on ARMAR (UniKarl, KTH)

We have integrated the box decomposition approach proposed in [HRK08, HK08] in a model-based scenario, based on a-priori known objects from the Karlsruhe object models database[4]. On ARMAR, texture-based object recognition and pose estimation were implemented using information from that database. If an object is detected, its identification and pose are communicated to a remotely integrated BoxGrasping server. While box constellations of each object in the database are computed beforehand, a selection and ranking of the generated grasps is done dynamically on each request, dependent on the object pose, but also on heuristics and simulated grasp-quality learning described in [HK08]. The final set of ranked hypotheses is sent back to ARMAR and checked for kinematic reachability before a selection is done. If no hypothesis was sent, the object is assumed to be non-graspable.

6.5 Integration of Controller Architecture of LUT at UJI

The low-level control architecture, developed at WP3 (see Deliverable 5), has been integrated through the OpenRAVE architecture (Sec. 3.1.2). In addition to providing high-level interface through OpenRAVE, the architecture includes common low-level interfaces to hardware. Through these standard interfaces, the controller architecture is an attempt to provide a cross-platform implementation. More details about the integration of the controller architecture and the OpenRAVE framework are given in Deliverable 5.

The first version of the controller architecture has been installed and tested on two hardware platforms, the UJI robot platform consisting of PA-10 Mitsubishi arm and a Barrett hand with tactile sensors, and the LUT platform consisting of a MELFA RV-3SB arm with a Schunk PG-70 gripper. In addition to the general framework, also the interfaces between the robot controllers and the hardware independent controllers have been implemented to allow the control of the two separate robot platforms using the standard interfaces defined by the controller architecture.

6.6 Object Database (UniKarl)

As mentioned in Section 4.2.2, in addition to the distributed initial object set, UniKarl proveded 3D models and color stereo images of the objects to enhance and support the development and the evaluation of vision methods for grasping in the project. For this purpose, a database has been established, which can be accessed from a website[4]. The object database is updated regularly and will be extended in the future.

[4]Karlsruhe Object Models Database `http://wwwiaim.ira.uka.de/ObjectModels`, developed within the Collaborative Research Center 588 (SFB588)

6.7 Grasp Taxonomy and Human Grasping Database (OttoBock, LMU, KTH, FORTH, UniKarl)

A grasp taxonomy has been established in a cooperation between OttoBock and KTH. For this purpose, existing grasp taxonomies have been analyzed in order to derive a human grasping taxonomy for daily objects. The results of this study are published under the projects web page (`www.grasp-project.eu`) under "'Resources"'.

In addition, a human grasping database has been established in cooperation between LMU, FORTH and UniKarl. For this purpose, human hand is tracked during grasping movements using two sensor systems: An electromagnetic Polhemus Liberty system which record data at 240 Hz and a stereo camera system running at 30 Hz. A dataset of human hand movement performing graping under different conditions has been established and published under the projects web page (`www.grasp-project.eu`) und "'Resources"'.

Chapter 7

Conclusions and Future Work

This Deliverable presented the work in the first year in WP7 towards the implementation of a cognitive control architecture for grasping and manipulation. The main results are:

- Software interfaces to existing grasp simulators (GraspIt!) and robot software control frameworks (MCA, OpenRAVE) as well as to the GRASP simulator.
- Software interfaces for vision, haptics and action.
- Object database with an initial set of objects for experiments in the project's first year and adequate object representations.
- Robot Editor to allow the creation of geometrical, kinematic and dynamic robot models as well as the convenient conversion of a variety of CAD formats.

In the second year we will continue our work on the extension and improvement of the defined interfaces and developed tools and on their transfer to other robotics platforms in the project. The main focus of WP7 will be identifying the requirements for an implementable cognitive architecture for grasping and dexterous manipulation and fundamental aspects of the transferability of the used representations within this architecture to different robot platforms.

References

[AAD06] P. Azad, T. Asfour, and R. Dillmann. Combining Apperance-based and Model-based Methods for Real-Time Object Recognition and 6D Localization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5339–5344, Beijing, China, 2006.

[AAD07a] P. Azad, T. Asfour, and R. Dillmann. Stereo-based 6D Object Localization for Grasping with Humanoid Robot Systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 919–924, San Diego, USA, 2007.

[AAD07b] P. Azad, T. Asfour, and R. Dillmann. Toward an Unified Representation for Imitation of Human Motion on Humanoids. In *IEEE International Conference on Robotics and Automation*, Rome, Italy, April 2007.

[AAD08] P. Azad, T. Asfour, and R. Dillmann. Robust Real-time Stereo-based Markerless Human Motion Capture. In *submitted to International Conference on Humanoid Robots*, Daejeon, Korea, December 2008.

[AAV$^+$08] T. Asfour, P. Azad, N. Vahrenkamp, K. Regenstein, A. Bierbaum, K. Welke, J. Schröder, and R. Dillmann. Toward humanoid manipulation in human-centered environments. *Robot. Auton. Syst.*, 56(1):54–65, 2008.

[ABB04] A. Albers, S. Brudniok, and W. Burger. Design and development process of a humanoid robot upper body through experimentation. In *IEEE/RAS International Conference on Humanoid Robots*, pages 77–92, 2004.

[ARA$^+$06] T. Asfour, K. Regenstein, P. Azad, J. Schröder, N. Vahrenkamp, and R. Dillmann. ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. In *IEEE/RAS International Conference on Humanoid Robots*, 2006.

[Bar06] Mark Barnes. Collada. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, page 8, New York, NY, USA, 2006. ACM.

[BRAD08] A. Bierbaum, M. Rambow, T. Asfour, and R. Dillmann. A potential field approach to dexterous tactile exploration. In *International Conference on Humanoid Robots 2008, Daejeon, Korea*, 2008.

[BSZD06] R. Becher, P. Steinhaus, R. Zöllner, and R. Dillmann. Design and Implementation of an Interactive Object Modelling System. In *International Symposium on Robotics*, 2006.

[DK08] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. Technical report, CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University, July 2008.

[GSK$^+$08] I. Gaiser, S. Schulz, A. Kargov, H. Klosek, A. Bierbaum, C. Pylatiuk, R. Oberle, T. Werner, T. Asfour, G. Bretthauer, and R. Dillmann. A New Anthropomorphic Robotic Hand. In *IEEE/RAS International Conference on Humanoid Robots*, Daejeon, Korea, 2008.

[GWBW06] D. Göger, K. Weiß, K. Burghart, and H. Wörn. Sensistive skin for a humanoid robot. In *International Workshop on Human-Centered Robotic Systems (HCRS'06)*, Munich, 2006.

[HK08] K. Huebner and D. Kragic. Selection of Robot Pre-Grasps using Box-Based Shape Approximation. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1765–1770, 2008.

[HRK08] K. Huebner, S. Ruthotto, and D. Kragic. Minimum Volume Bounding Box Decomposition for Shape Approximation in Robot Grasping. In *IEEE International Conference on Robotics and Automation*, pages 1628–1633, 2008.

[KAP$^+$05] A. Kargov, T. Asfour, C. Pylatiuk, R. Oberle, H. Klosek, S. Schulz, K. Regenstein, G. Bretthauer, and R. Dillmann. Development of an anthropomorphic hand for a mobile assistive robot. In *International Conference on Rehabilitation Robotics (ICORR 2005)*, pages 182–186, 2005.

[KBSD07] A. Kasper, R. Becher, P. Steinhaus, and R. Dillmann. Developing and Analyzing Intuitive Modes for Interactive Object Modeling. In *International Conference on Multimodal Interfaces*, 2007.

[KL00] J. Kuffner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *IEEE Int'l Conf. on Robotics and Automation (ICRA'2000), San Francisco, CA*, pages 995–1001, 2000.

[LaV06] S. M. LaValle. *Planning Algorithms.* Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[MA04] A. T. Miller and P. K. Allen. GraspIt! a versatile simulator for robotic grasping. *Robotics & Automation Magazine, IEEE*, 11(4):110–122, 2004.

[MAA$^+$06] A. Morales, T. Asfour, P. Azad, S. Knoop, and R. Dillmann. Integrated Grasp Planning and Visual Object Localization For a Humanoid Robot with Five-Fingered Hands. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006.

[SPK$^+$04] S. Schulz, Ch. Pylatiuk, A. Kargov, R. Oberle, and G. Bretthauer. Progress in the development of anthropomorphic fluidic hands for a humanoid robot. In *IEEE/RAS International Conference on Humanoid Robots*, Los Angeles, Nov 2004.

[VAD07] N. Vahrenkamp, T. Asfour, and R. Dillmann. Efficient motion planning for humanoid robots using lazy collision checking and enlarged robot models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3062–3067, San Diego, USA, 2007.

[VSA$^+$08] N. Vahrenkamp, C. Scheurer, T. Asfour, J. Kuffner, and R. Dillmann. Adaptive motion planning for humanoid robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2127–2132, Nice, France, 2008.

[VWA$^+$08] N. Vahrenkamp, S. Wieland, P. Azad, D. Gonzalez, T. Asfour, and R. Dillmann. Visual Servoing for Humanoid Grasping and Manipulation Tasks. In *IEEE/RAS International Conference on Humanoid Robots*, Daejeon, Korea, 2008.