2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Embodiment Independent Manipulation Through Action Abstraction

Janne Laaksonen, Javier Felip, Antonio Morales and Ville Kyrki

Abstract— **The adoption of robots for service tasks in natural environments calls for the use of sensors to allow manipulation of objects under imperfect environment knowledge and the use of knowledge transfer from humans. This paper addresses these challenges by proposing a new abstraction architecture for embodiment independent sensor-based control of manipulation. The aim is to address three specific challenges: hardware independent control of manipulation, use of sensors to alleviate problems of complexity and uncertainty of the environment, and ease of transferring knowledge over different embodiments through a hierarchical abstract representation of manipulation skills. The proposed abstraction architecture is demonstrated for hardware independence and failure detection on two different manipulator platforms.**

I. INTRODUCTION

One of the important changes necessary for adopting robots for service tasks in natural environments is that the robots need to robustly operate in spite of incomplete knowledge of their environment. This necessitates the use of sensors for perception. In addition to perceptual capabilities, robots should also be able to learn from human demonstration. For this learning, embodiment independent representations are necessary as humans and current robotic platforms differ in both perceptual and manipulative skills. A major challenge in this is that the sensors and the embodiments are tightly coupled in sensor-based manipulation. This coupling needs to be decreased to generalize the knowledge. One possible answer to the challenge is to abstract the knowledge and use this abstraction as the basis for the knowledge transfer.

Service robots must be able to cope with several different use cases and tasks, for example setting up a table or placing groceries into a refrigerator. The challenges these tasks present to a robot are twofold. On one hand, the environment poses challenges including a) complexity of the world, such as different number of objects to be handled, b) uncertainty in world knowledge, for example the knowledge of the objects' physical characteristics, and c) dynamic nature of the environment, that is, there are typically other actors in the environment which need to be taken into account. On the other hand, the knowledge transfer from a human to the robot needs to be solved for a particular robot embodiment. The above description demonstrates that there is simultaneously a need for highly embodiment specific information to cope with the uncertain environment and a need to have the embodiment independence to be able to effectively transfer the knowledge between humans and robot embodiments.

In this paper, we propose a new approach, an abstraction architecture, that aims to solve the issue of how the abstract embodiment independent information is used with the embodiment dependent information to cope with the demands of service robotics. The proposed approach uses a hierarchical approach for the decomposition of manipulation skills, which are focused on grasping in this paper, with the ability to use multiple sensors and sensor types. Individual manipulation skills are represented as finite state automata or finite state machine (FSM), with attributes which are used to adapt each skill to a particular use. Using the hierarchical approach ensures that the ability to function in a maximum number of different use cases (for example, different objects, different environment, different hardware) is possible, as the different levels in the hierarchy can be adapted according to the use case. Failure detection is also considered using the finite state automata. The structure of the architecture is shown in Figure 1. The figure shows how the abstract information is completely separated from the embodiment specific information which facilitates the use of multiple embodiments for abstract actions. Combining both the hardware independence and sensor-based manipulation is one of the highlights of the proposed architecture that has not been demonstrated previously, as the hardware independence can not interfere with the real-time requirement of sensor-based manipulation.

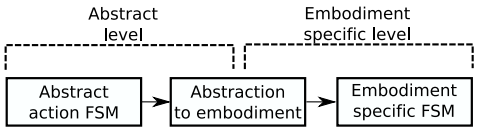

Fig. 1. Levels of the abstraction architecture.

Next, the related work is discussed in Section II. Section III presents the proposed approach. To demonstrate the approach, Section IV shows results of experiments using two different robotic platforms, and Section V concludes the paper with a discussion of the abstraction architecture and future work.

II. RELATED WORK

A number of robot architectures have been presented previously, for both manipulation and for more general

The research leading to these results has received funding from the European Community's Seventh Framework Programme under grant agreement n° 215821, and by Fundació Caixa-Castelló (P1-1A2006-11). V. Kyrki was supported by Academy of Finland grant 114646.

J. Laaksonen and V. Kyrki are with Department of Information Technology, Lappeenranta University of Technology, P.O. Box 20, 53851 Lappeenranta, Finland, jalaakso@lut.fi, kyrki@lut.fi

J. Felip and A. Morales are with Robotic Intelligence Laboratory at the Department of Computer Science and Engineering, Universitat Jaume I, 12006 Castellón, Spain {jfelip,morales}@uji.es

use. Here, we will concentrate on architectures especially targeting manipulation. The concept of primitive skills is central in many of the works as is the use of discrete states to divide a manipulation action into parts.

Milighetti et al. [1] presented an architecture which uses primitive skills, that combine to form a skill, which in turn form a complete task. Each primitive skill is selected by heuristic selection out of many possible primitive skills, based on the sensor signals. A neural network is used to detect the change between the skills. Each primitive skill is based on a separate controller. While the basic idea of hierarchical decomposition is similar to ours, in their approach there is no possibility to adapt the primitive skills themselves.

Haidacher et al. [2] demonstrated an architecture for the DLR Hand II. The architecture is based on different levels of complexity, which handle different aspects of the control. Again, the concept of hierarchical decomposition is central, but the architecture is limited to a single hand and the adaptiveness of the architecture has to be implemented at the highest level as the lower levels are statically defined.

Han et al. [3] present a control architecture for multi-fingered manipulation. As previously, the architecture is based on different levels that handle control from planning to actual joint control. The problem with the architecture is the lack of adaptation as the architecture shows that only predetermined architectural components, such as low level controllers, are available to use. In addition, the architecture does not consider the robotic arm, only the hand.

Hybrid discrete-continuous control architectures for manipulation, such as [4] and [5], separate the control phases according to the state of the manipulator. This is achieved by using discrete events to classify the manipulation configuration and using continuous states to control the dynamic behavior in different configurations. This type of architecture is suitable for both low-level control [5] and for a complete control architecture [4]. Petersson et al. [4] demonstrate a control architecture for a mobile manipulator based on behaviors. The actual manipulator behavior is modelled as a sequence of configurable primitive actions. These primitive actions can be freely defined. These primitives can be chained together using a hybrid automaton to form an action. Although the architecture has some elements desired from a service robotics architecture, such as hardware independence, it lacks the sensor-based approach required to cope in uncertain environments, for example there is no mention of failure detection using the available sensors.

Another mobile manipulator architecture by Chang and Fu [6], is also based on hybrid discrete-continuous control architecture. However, the architecture is more limited than in [4], only consisting of pre-determined set of states, which control the manipulator. These states can be configured for different manipulation tasks. Aramaki et al. [7] have also used automata to control a humanoid robot at a low level.

For a static manipulator, Prats et al. [8] presented a comprehensive system for controlling manipulation. The system also uses automata to control the progress of actions, by separating the primitive actions into the states of the automata. One of the defining features of the architecture is that each state of the automata can be a primitive action or an automaton. This feature can be used to create complex actions. However, the problem of hardware independence is not discussed.

Most of the described architectures have one common element, the use of automata in determining the current state of the control. This approach is also used as part of our proposed approach. However, none of the reviewed architectures describe or demonstrate methods for achieving hardware independence, which is one of the central claims of our approach.

III. ABSTRACTION ARCHITECTURE

Before going in to the details of the proposed approach, we define the terminology used. We use the term *abstraction architecture* for the whole approach. This should not be confused with the control architecture, which is a part of the abstraction architecture related to the actual execution of the actions. Figure 2 shows a general hierarchical decomposition of planning and control. The hierarchy consists of three levels: task, action and primitive. *Task* is the highest level of abstraction, representing a semantically meaningful task such as for example emptying a shopping bag. The task comprises of a sequence of *actions*, which represent subtasks, such as moving an object from one location to another. Actions consist of *primitives*, or primitive actions, which are the lowest level of control in the proposed architecture. More accurate definition for a primitive action used in the proposed architecture is that each primitive action is implemented using a single low-level controller, which is responsible for the actual control of robot hardware.

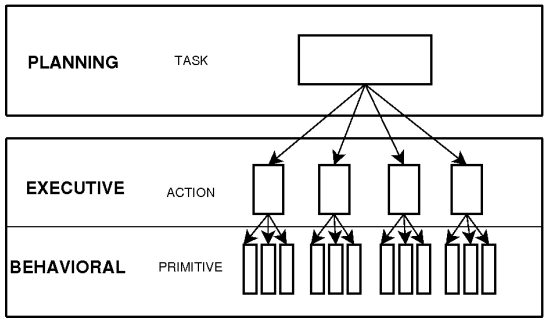

Fig. 2. Planning and control levels.

The abstraction architecture presented in this paper will focus only on the action and primitive levels shown in Figure 2, that is, the actual on-line part of control instead of task planning which might be performed off-line. The architecture itself has elements of both behavioral and executive levels discussed in [9] by Kortenkamp and Simmons. It should be noted that the behavioral control is not considered in the Brooksian sense, instead the behavioral level considers primitive actions which can be executed with traditional control theory. We will show that it is possible to adapt to different tasks and different hardware on the two lower levels

using a set of attributes that are implemented in the actions and in the primitive actions. The focus of the abstraction architecture is on manipulation, especially grasping, by a robotic arm and a robotic hand. Grasping is also used as an example throughout the description of the abstraction architecture.

The control architecture presented in this paper is based on a high level architecture design, which defines the internal structure of the control architecture and the interfaces for hardware and the communication between the controller and outside components. The control architecture itself is not novel, as we have adapted the same idea of using automata for control, seen in Section II. However, the abstraction architecture, i.e., how to combine the abstract actions with the control architecture is novel, and described in detail in sections III-A and III-C. The high level design of the control architecture is depicted in Figure 3 and the actual implementation is detailed in III-B.

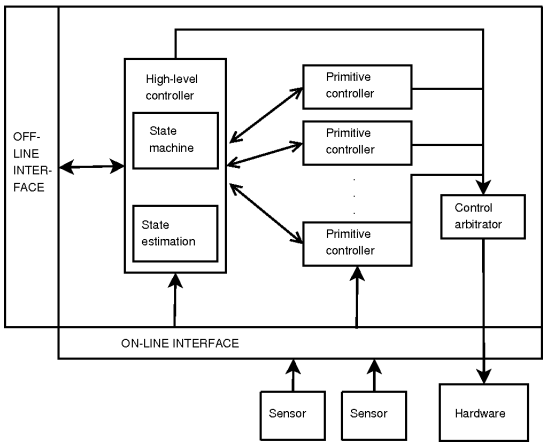

Fig. 3. Control architecture design.

Main features of the control architecture design are the inclusion of two communication interfaces and separation of the high level controller and the primitive controllers. The communication interfaces are for asynchronous "slow" communication and for real-time communication with sensors and actuators. The high level controller handles the internal state of the controller while the primitive controllers output the control signals to the hardware actuator. The primitive controllers can be freely defined. Final component in the design is the control arbitrator which ensures that a single control input from multiple primitive controllers is communicated to the manipulator.

A. Abstract State Machine

The abstract state machine is a hardware independent description of a manipulation action. The abstract state machine uses XML (eXtensible Markup Language) to describe all relevant information, such as the states and transitions of the state machine. Also information about a target object, e.g. pose and mass, and obstacles in the manipulation environment are given through the XML. All properties and definitions in XML are hardware independent.

TABLE I
STATE AND TRANSITION PROPERTIES.

state	transition
movement	success
hand_shape	grasp_stable
trajectory	grasp_lost
	finger_contact
	finger_contact_lost
	timeout
	collision
	hardware_failure

The abstract state machine is described through definition of states and transitions between the states. Current set of properties for both states and transitions are listed in Table I. The transition properties describe the condition when the transition is triggered. While most of the transition properties are self-explanatory, *success* transition denotes that the controller has reached its target, the state properties are: *movement* describing whether the motion of the manipulator is guarded or free, *hand shape* describing the hand shape with abstract concepts, such as closed or open, and finally, *trajectory* describing a trajectory for the manipulator end-effector, using both position and pose definitions.

In addition to the properties, the state also has attributes, which infer the manipulator motion that is desired from each defined state. These attributes are:

- **success:** The success end state of the state machine.
- **failure:** The failure end state of the state machine.
- **move:** Moving the manipulator without an object.
- **transport:** Moving the manipulator with an object.
- **grasp:** Grasp the object.
- **release:** Release the object.

These attributes are designed with grasping in mind, but other forms of manipulation, such as pushing, are possible to define using the abstract state machine. These attributes are the key factor in selecting the primitive controllers during the translation process, which is described in Section III-C. An example XML definition describing a simple grasp and lift manipulation is shown in Table II. Some of the elements have been left out for brevity, e.g. properties of the object and some of the common transitions, e.g. timeout to the failure state.

B. Embodiment Specific State Machine

The embodiment specific state machine is the functional representation of the abstract state machine. The embodiment specific state machine is able to control the manipulator throughout a single action and decide whether the action was successful or a failure.

The embodiment specific state machine follows the structure of the abstract state machine and the high level design discussed earlier. The high level controller presented in the high level design acts as the single most important element in the proposed control architecture. The high level controller consists of the actual embodiment specific state machine, interfaces to the hardware manipulator, i.e., the robotic arm

TABLE II

LISTING OF THE XML ABSTRACT STATE MACHINE.

```
<statemachine>
 <state name="approach" type="move">
  <movement>free</movement>
  <hand_shape>open</hand_shape>
 </state>
 <state name="preshape_hand" type="move">
  <movement>guarded</movement>
  <hand_shape>pinch_grasp_preshape</hand_shape>
 </state>
 <state name="grasp_object" type="grasp">
  <movement>guarded</movement>
  <hand_shape>pinch_grasp</hand_shape>
 </state>
 <state name="lift_object" type="transport">
  <movement>guarded</movement>
  <hand_shape>pinch_grasp</hand_shape>
  <trajectory>
   <position>0.2 0.6 0.25</position>
  </trajectory>
 </state>
 <state name="success_end" type="success">
 </state>
 <state name="fail_end" type="failure">
 </state>

 <transition origin="approach"
    destination="preshape_hand">
  <success/>
 </transition>
 <transition origin="preshape_hand"
    destination="grasp_object">
  <success/>
 </transition>
 <transition origin="grasp_object"
    destination="lift_object">
  <success/>
  <grasp_stable/>
 </transition>
 <transition origin="lift_object"
    destination="fail_end">
  <grasp_lost/>
 </transition>
 <transition origin="lift_object"
    destination="success_end">
  <success/>
  <grasp_stable/>
 </transition>
</statemachine>
```

and the hand and the control arbitrator. Hardware interface is defined to have one unified control method, Cartesian velocity control, for all arms. However, it is possible to define more control methods for both the arm and the hand of the manipulator. The embodiment specific state machine is modelled as a hybrid discrete-continuous automaton, which was proven successful in many of the reviewed architectures [4], [5], [8]. A hybrid discrete-continuous automaton can also mimic human grasping [10], [11], which consists of several sub-actions.

Each state of the automaton has its own primitive controllers and transitions to other states. As the high level design states, the primitive controllers and transitions are freely definable. However, a common interface for primitive controllers and transitions is required. For primitive controllers the common interface is the control output from the controller and for transitions it is the boolean indication whether the transition to another state should be made or not. Another common interface to both primitive controllers and transitions is setting of attributes which means that we can adapt both the controllers and transitions through these interfaces during the execution of the automaton. All transitions and primitive controllers have also access to all the sensors in the system. Sensor access has been implemented through OpenRAVE [12] and the whole high level controller has been integrated into OpenRAVE, which is an open framework for simulating and controlling robots. OpenRAVE is also used as the off-line interface.

While the embodiment specific state machine is designed to closely resemble the abstract state machine to ease the process of translation between them, it is possible to define the embodiment specific state machine manually to suit needs that can not be described by an abstract state machine. It is also possible to create new states with existing primitive controllers and transitions during the execution of the automaton which enables the use of probabilistic methods such as [13].

C. Translation

The translation process is what combines the abstract state machine and the embodiment specific state machine. The translation takes the abstract state machine as an input, and translates the abstract state machine into an embodiment specific state machine. The translation process is depicted in Figure 4.

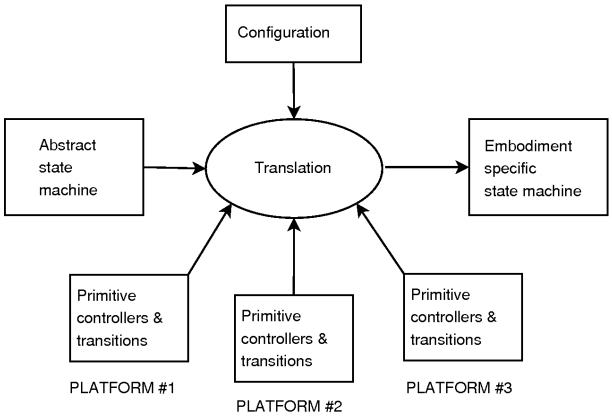

Fig. 4. Translation process.

As can be seen from Figure 4, the translation component needs input defining the configuration of the translation process, i.e. the target platform and the platform specific transitions and primitive controllers used directly in the embodiment specific state machine. The benefit of this arrangement is that the only hardware dependent blocks shown in the figure are the primitive controllers and transitions that are platform specific. Also the critical requirement of real-time operation for sensor-based control is fulfilled as the embodiment specific state machine can be run as is, without any additional overhead from maintaining hardware independence.

The translation process also requires a mapping component which produces the embodiment specific state machine

from the abstract automaton. Currently the mapping itself is done manually per platform, but once the mapping is complete, the translation process from any abstract automaton is performed automatically. This mapping is fairly simple to implement as there are only a limited amount of input properties and the mapping is not aware of the abstract action in any way.

Furthermore, as we have defined a common Cartesian control interface for the arm, we can use primitive controllers that use the arm velocity control for all hardware platforms without modifications. The same applies to some transition conditions, e.g. timeout can be used in all platforms. Thus, building the basic primitive controllers and transitions gives the added benefit of not having to implement all controllers and transitions for each new platform introduced to the system.

IV. DEMONSTRATIONS

The abstraction architecture is demonstrated on two platforms which differ in their kinematics, control, and sensory capabilities. The first platform is a Melfa RV-3SB robot arm with a Schunk PG70 parallel jaw gripper. The arm has 6 DOF (degrees of freedom) and the gripper 1 DOF. In addition to these, a Weiss tactile (pressure) sensor grid is attached to each finger of the gripper. Grasping force is controlled by the feedback from the tactile sensors. Also the stability of the grasp is determined from the tactile sensor feedback.

The second platform consists of a Mitsubishi PA-10 arm with 7 DOF mounted on an Active Media PowerBot mobile robot. The manipulator is endowed with a three-fingered Barrett Hand and a JR3 force/torque and acceleration sensor mounted on the wrist, between the hand and the end-effector. The hand has been improved by adding on the fingertips arrays of Weiss tactile sensors. Each finger of the hand has a built-in strain sensor. The JR3 is a 12 DOF sensor that measures force, torque and acceleration in each direction of the space. The finger-force sensors are used to stop closing the fingers when the object is touched. This sensor is also used to control the force that each finger applies to the object in the final grasp. A complex control grasp primitives that make use of sensor feedback to correct the grasp contacts have been implemented and used for this platform[14].

An executed action is depicted in Figure 5, which shows snapshots of the action being executed on both platforms. The abstract action contains 7 primitive actions: approach, preshape, grasp, lift and move, move down, release and move away. The executions are shown in full for both platforms in the accompanying video. The two objects used in the demonstrations are normal household items, a detergent bottle and a salt container. Both objects have the same mass, 0.5 kg. These objects are shown in Figure 6. In addition to the objects shown, the sensor-based grasping has been demonstrated in the systems with several other similar household objects.

Fig. 6. Grasped objects.

A. Demonstrating Sensor-based Grasping

One of the key challenges that our abstraction architecture addresses is how to combine the need to have sensor based information which is highly coupled to embodiment and abstraction of the action. To show that our abstraction architecture is able to cope with this problem, we demonstrate sensor-based grasping of objects on the two platforms described before. Using the same abstract instructions, i.e., the abstract state machine, we were able to complete the same task on the two platforms, and use the embodiment specific sensors to grasp the object.

As shown in Figure 5 and in the accompanying video, we were able to grasp objects based only on the sensor data from the hand and the arm, no vision was used. Using the same abstract state machine for both platforms shows clearly that we are able to use abstraction and then turn this abstract information to platform specific primitives and transitions used in the sensor-based control.

In the context of the demonstration we were able to use the same controllers for both arms, but the abilities of the hands are too different, in terms of kinematics and sensors, so that both hands had their own controllers. Also the transitions for grasp stability or instability are customized for each of the platforms in order to use the different sensors on the platforms.

B. Failure Detection

Failure detection is an important factor in the proposed architecture. Failure detection can be used to arise surprise and for learning. As the control architecture is focused towards sensor-based control, all the available sensors can be used for failure detection. Failure is also explicitly included in the abstract state machine as one of the end states.

To demonstrate failure detection, the same abstract state machine was used as before with the demonstration platforms. However, the object mass was artificially increased, but this was not reflected on the abstract state machine. Sensor use is critical in detecting failures which can be seen in Figure 7, which shows the result of the demonstration on the first platform. The figure depicts the total force affecting

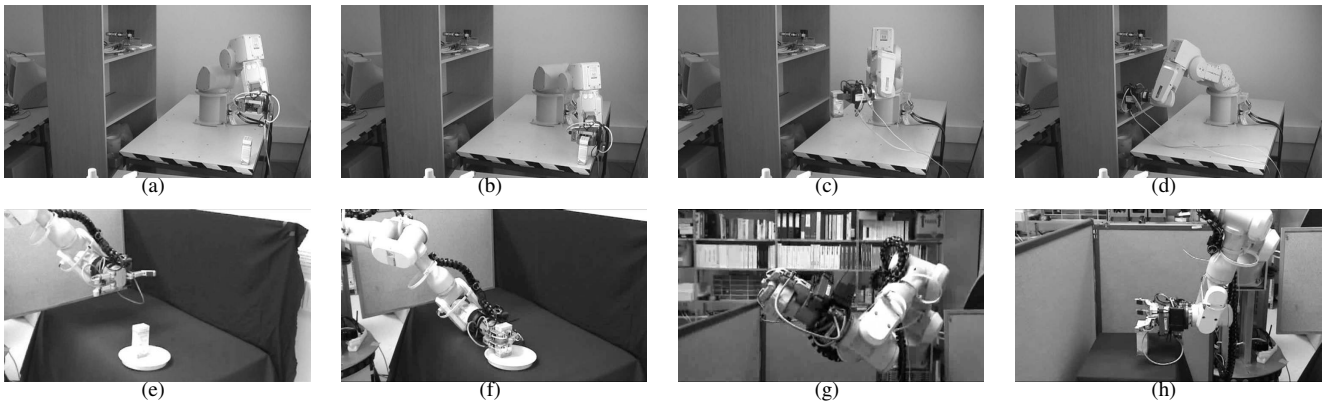

Fig. 5. Action execution on both platforms: (a) Approach; (b) Grasp; (c) Move; (d) Release; (e) Approach; (f) Grasp; (g) Move; (h) Release.

the tactile sensors and the state changes of the state machine as vertical lines. As can be seen from the figure, the state machine was executed normally until the lift and move primitive failed, and the state machine moved into failure state, halting the execution of the state machine. The failure mode is also shown in the accompanying video for one of the platforms. However, both platform have the capability of failure detection, using their platform specific sensors.

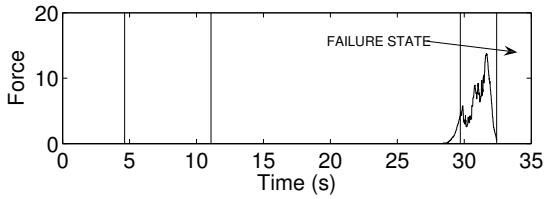

Fig. 7. Measured force during a failed action.

V. DISCUSSION AND CONCLUSIONS

This paper presented an approach for handling embodiment independent knowledge and transferring that knowledge to a more embodiment specific representation which can be used to control the embodiment. Our approach specifically addresses embodiment independence, the use of sensors as an integral part of control, and the modelling of actions as automata of adaptive primitive actions. The embodiment independent knowledge is modelled as a state machine which is then translated to suit each embodiment and its external and internal sensors.

A noteworthy observation is that the use of primitive attributes reduces the problem of learning motions to the learning of primitive attributes. While learning was not demonstrated in this paper, the abstraction would be useful for both imitation learning from a human demonstrator as well as learning by exploration by the robot platform itself. The use of primitives sidesteps the problem of decomposing a trajectory learned from a human to a set of primitives. On the other hand, a known set of primitives must be created and configured for a hand. However by using adaptive primitives, we believe that a wide range of natural motions can be mapped to a limited set of primitives since typical human manipulation is known to consist of a limited number of types of interaction.

Our future work includes implementing the abstraction architecture on more platforms. The abstraction architecture will then form the base for further research on the use of sensors as a part of manipulation and especially as a part of manipulation learning. The focus will be on grasping and how to learn to grasp using the available sensors in both real and simulated environments.

REFERENCES

[1] G. Milighetti, H. B. Kuntze, C. W. Frey, B. Diestel-Feddersen, and J. Balzer, "On a primitive skill-based supervisory robot control architecture," in *Proc. IEEE ICRA'05*, July 2005, pp. 141–147.

[2] S. Haidacher, J. Butterfass, M. Fischer, M. Grebenstein, K. Joehl, K. Kunze, M. Nickl, N. Seitz, and G. Hirzinger, "DLR hand II: Hard- and software architecture for information processing," in *Proc. IEEE ICRA'03*, Sept. 2003, pp. 684–689.

[3] L. Han, Z. Li, J. C. Trinkle, Z. Qin, and S. Jiang, "The planning and control of robot dextrous manipulation," in *Proc. IEEE ICRA'00*, Sept. 2000, pp. 263–269.

[4] L. Petersson, M. Egerstedtt, and H. Christensen, "A hybrid control architecture for mobile manipulation," in *Proc. IEEE/RSJ IROS'99*, 1999, pp. 1285–1291.

[5] T. Schlegl and M. Buss, "A discrete-continuous control architecture for dextrous manipulation," in *Proc. IEEE SMC'99*, vol. 2, 1999, pp. 860–865.

[6] C.-F. Chang and L.-C. Fu, "A hybrid system design of a mobile manipulator," in *Proc. IEEE ICRA'06*, May 2006, pp. 406–411.

[7] S. Aramaki, H. Shirouzu, and K. Kurashige, "Control program structure of humanoid robot," *IECON 02*, vol. 3, pp. 1796–1800 vol.3, Nov. 2002.

[8] M. Prats, A. P. del Pobil, and P. J. Sanz, "A control architecture for compliant execution of manipulation tasks," in *Proc. IEEE/RSJ IROS'06*, Oct. 2006, pp. 4472–4477.

[9] D. Kortenkamp and R. Simmons, "Robotic systems architecture and programming," in *Springer Handbook of Robotics*, 1st ed., B. Siciliano and O. Khatib, Eds. Berlin, Germany: Springer-Verlag, 2008.

[10] U. Castiello, "The neuroscience of grasping," *Nature Neuroscience*, vol. 6, pp. 726–736, Sep 2005.

[11] I. Serrano Vicente, V. Kyrki, D. Kragic, and M. Larsson, "Action recognition and understanding through motor primitives," *Advanced Robotics*, vol. 21, no. 15, pp. 1687–1707, 2007.

[12] R. Diankov and J. Kuffner, "OpenRAVE: A planning architecture for autonomous robotics," Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-08-34, July 2008.

[13] K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, "Grasping POMDPs," in *Proc. IEEE ICRA'07*, Apr. 2007, pp. 4685–4692.

[14] J. Felip and A. Morales, "Robust sensor-based grasp primitive for a three-finger robot hand," in *IEEE/RSJ International. Conference on Intelligent Robots and Systems*, Oct. 2009.