Autonomous Robots: Special Issue on Autonomous Mobile Manipulation manuscript No.
(will be inserted by the editor)

Generality and Legibility in Mobile Manipulation

Learning Skills for Routine Tasks

Michael Beetz, Freek Stulp, Piotr Esden-Tempski, Andreas Fedrizzi, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Federico Ruiz

Intelligent Autonomous Systems Group, Technische Universität München, Department of Informatics
Boltzmannstr. 3, D-85748 Garching bei München, Germany

Received: date / Revised version: date

Abstract This article investigates methods for achieving more general manipulation capabilities for mobile manipulation platforms, which produce legible behavior in human living environments. To achieve generality and legibility, we combine two control mechanisms. First of all, experience- and observation-based learning of skills is applied to *routine tasks*, so that the repetitive and stereotypical character of everyday activity is exploited. Second, we use planning, reasoning, and search for *novel tasks* which have no stereotypical solution. We apply these ideas to the learning and use of action-related places, to the model-based visual recognition and localization of objects, and the learning and application of reaching strategies and motions from humans. We demonstrate the integration of these mechanisms into a single low-level control system for autonomous manipulation platforms.

1 Introduction

In this article, we investigate a simple everyday mobile manipulation task: putting typical, easy to grasp everyday objects, e.g. mugs and plates, on the table in order to perform a household task such as setting the table. Our objective is to perform such tasks with a generality and flexibility that is typically required in the context of everyday manipulation tasks, and to exhibit motions and behavior that look natural and are legible for humans.

The objective of performing manipulation tasks already presents a huge challenge for mobile manipulation research. Rosenbaum and his colleagues [34] call this the Turing test for action: "In order to follow a simple instruction to pick up a rock, the robot must somehow answer questions such as: "From what direction should I approach it? Should I grab it on that outcropping closest to my left? Which posture will allow me to reach it?" "

There are two extreme positions on how to approach the Turing test for action. First, reinforcement learn-

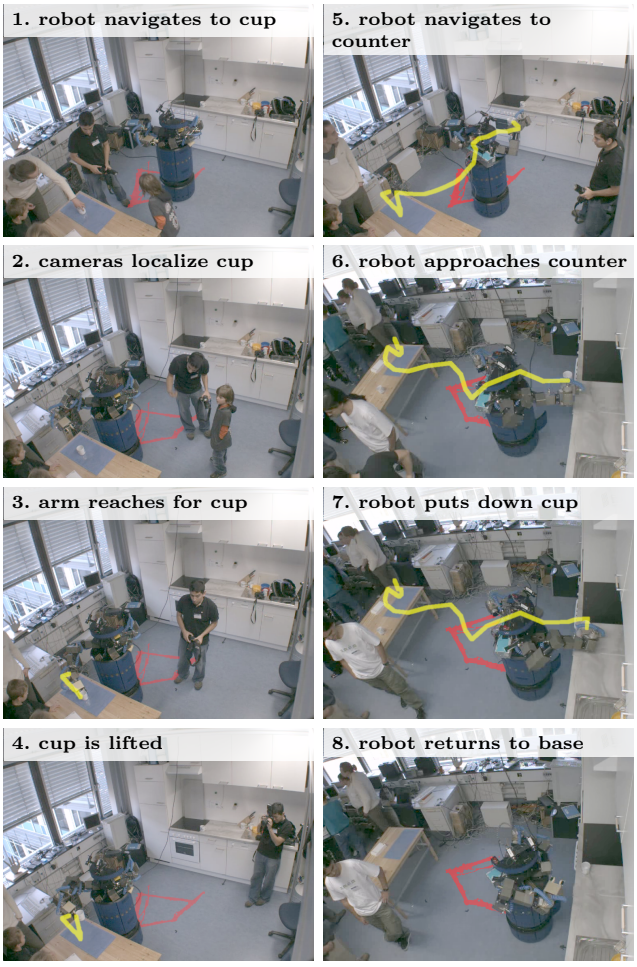


Fig. 1 A reach and grasp trajectory performed during the demo

[0] This article is accompanied by several videos, which can be downloaded from `http://www9.in.tum.de/people/stulp/arvideo/`

ing aims at learning an optimal policy for doing the task [44]. Policies so acquired are very specific for the task context in which they were learned, and it is difficult to port them to new robots and tasks. Motion planning, the second approach, is more flexible, as it uses (directed) search to find a solution to the task [22]. However, this approach treats each task as a novel one, and cannot exploit similarities between tasks. Also, in high-dimensional spaces, a solution is not always found.

Our approach is to have learned standard solutions, or *skills*, for routine tasks, as depicted in Fig. 2. These skills are optimized over time, which leads to stereotypical motion [50]. Such motion is predictable, can be modelled well, and is legible for humans, i.e. humans recognize the intention of the motion. By exploiting the recurrence of many tasks in this way, we are "exploiting the loopholes of life", as Ian Horswill puts it [15]. This approach combines the advantages of both reinforcement learning and motion planning, being optimality and flexibility respectively.

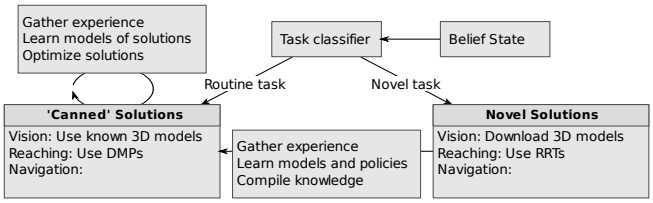


Fig. 2 Learning and optimizing skills for routine tasks.

To exploit the stereotypical character of everyday activity we employ a control system that first classifies tasks into everyday ones for which it has canned solutions and novel ones which have to be planned for and require more sophisticated reasoning before being solved. Having encountered novel tasks that might have to be performed more often, the system gathers experience with the novel tasks to optimize their solution and stores learned models and solutions for later use.

The second issue we want to address in this article is generality. Rather than spelling everything out explicitly and in a detailed manner by hand, we would like to state the control programs more generally and let the program infer the details given more specific information about the current task context, the task itself, and the situation in which the task is to be performed. Thus instead of specifying the exact position from where the robot is to pick up a particular object we ask the robot to go to a place that is suitable for the robot to pick it up. The robot then chooses the most suitable position based on context information such as clutteredness and more accurate information about the object to be lifted.

The application scenario we consider, as depicted in Fig. 1, is an autonomous kitchen assistant [2], in particular setting the table. This task requires the robot to localize itself, to grasp several objects from working surfaces in the kitchen, and to place them on a table.

The main contributions of this paper, which are illustrated in this application scenario are the following ones. We propose

- ... methods for achieving more general manipulation capabilities for mobile manipulation platforms, which produce legible behavior in human living environments by combining two control mechanisms. First of all, experience- and observation-based learning of skills is applied to *routine tasks*, so that repetitive and stereotypical character of everyday activity is exploited.
- ... the concept of *action-related place* to make least-commitment decisions as to where the robot should stay to reach for an object. Action-related places are learned from experience and the robot can revise and refine action-related places during operation in order to better reflect the action context including estimated accuracies of the robot's and the target object's pose.
- ... a reliable object recognition and localization system that can find instances of object categories in its environments by retrieving 3D models from the world wide web and that improves the models to facilitate the subsequent visual tasks concerning this object.
- ... a control mechanism for reaching tasks that learns the motion strategy and behavior from the observation of human reaching behaviors. The result is a legible and predictable reaching behavior of the robot.

We demonstrate the integration of these mechanisms into a single low-level control system for autonomous manipulation platforms.

The rest of this article is structured as follows: First, we present the application scenario and an overview of the system. Then, we discuss related work in Section 2. In Section 3 we describe an algorithm that determines goal positions for navigation, from which manipulation will likely succeed. Section 4 explains how the robot uses its stereo vision system to localize the target object. The stereo-typical reaching trajectories the robot uses to reach to and grasp the object are explained in Section 5. Section 6 presents the task-level controller that is important for high-level failure recovery and allows for flexible and reliable operation of the system. Results are presented in Section 7, and we conclude with Section 8.

1.1 Application Scenario

To illustrate the operation of our system let us consider the following demonstration scenario, which addresses several questions raised by the "Turing test for action". These questions include the ones about where the robot should go in order to perform some manipulation action,

where should the robot grasp the object, and how should the robot reach such that reaching is flexible and natural looking. In all these aspects the robot will make control decisions and parameterize actions in ways that it has learned from experience.

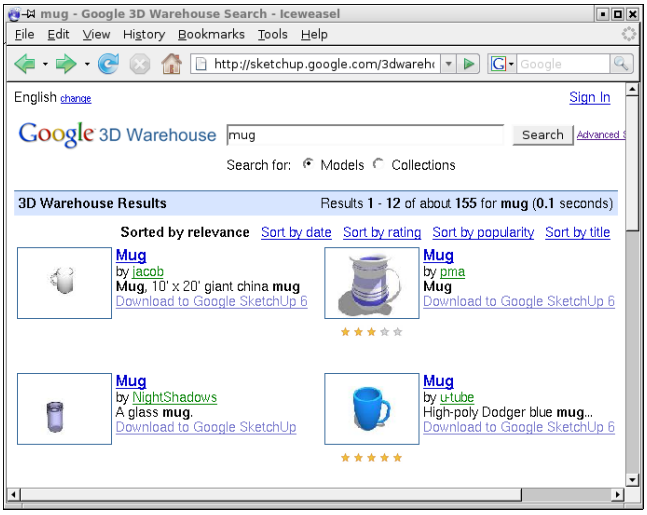


Fig. 3 Result of the querying Google 3D warehouse for the term "mug".

In our scenario the robot is installed in its working environment and automatically learns a semantic 3D object model of its environment [36]. It also learns models of the objects it has to manipulate in the installation phase. To do so, the robot crawls websites and retrieves candidate 3D models of object classes that are relevant for upcoming jobs. Fig. 3 shows the result of a *http-get*-query to *Google 3D warehouse*. The robot filters out incorrect answers assuming that most of the answers returned by *Google 3D warehouse* are models that match the query. It then uses the plausible 3D object models to find instances of the object class in its environment, as depicted in Fig. 4.

Fig. 4 There is no matching object on the table (left) but a matching object is found in the cupboard (right).

Besides acquiring the environment model and models of the objects to be manipulated, the robot also learns models of the effectiveness of its navigation and pick-and-place capabilities in this particular environment. For a particular environment at hand, these effectiveness-models are used to tailor the choice and parameterization of action control.

One particular example of acquiring environment-specific models of its manipulation capabilities is that the robot learns the places where it can easily pick up objects from the table by autonomously collecting data about picking up and placing objects in this particular environment. These models take into account the height and other dimensions of the table, the dexterity of the robot with respect to the challenges imposed by the objects that are to be manipulated, and other aspects that are important for achieving the manipulation task. The models are therefore grounded in experience rather than based on analytical considerations.

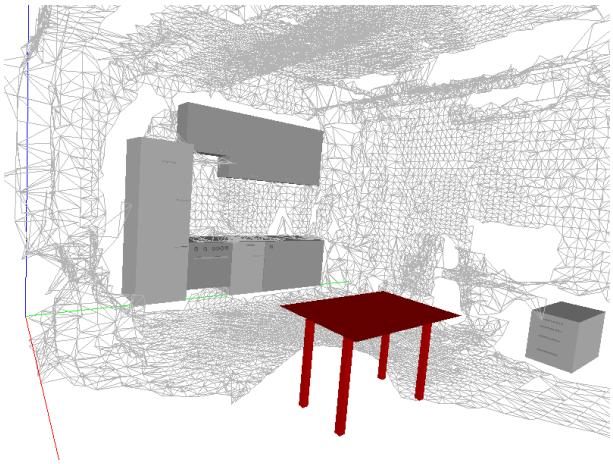


Fig. 5 Results for a query for the location of a piece of furniture, in this case the table.

Thus upon receiving a command to move a cup from the table onto the kitchen counter, the robot uses its environment model to query the information needed to translate the abstract instruction into accurate geometric information required to navigate and look at the right positions. Fig. 5 depicts the result of querying the environment model for the table in the environment.

Based on this information the robot looks in order to determine where the cup is located on the table. Because it could not detect the cup the first time, the robot varies its position and the direction of the camera to get a better look. After having detected the mug, the robot navigates to a suitable position for picking up the object. Upon arrival the robot looks at the cup more carefully to estimate the position of the cup with a precision that is sufficient for reaching and grasping the cup.

Having fixated the cup on the table the robot performs a reaching motion that mimics human reaching trajectories. This motion strategy enables the robot to produce not only behavior that is more legible for humans, but it also to better map motions of people to its own motion apparatus and thereby simplifies imita-

Fig. 6 Two cups detected and localized on the table using the model from Google 3D warehouse. The model has been automatically scaled, but there are still discrepancies between the cup and the model with respect to the handle of the mug.

tion learning. The navigation and the reaching trajectory performed by the robot are depicted in Fig. 1.

Having grasped the object, the robot moves its hand into the carrying position for the object and then navigates to the counter to put the object onto the counter. Noticing the clutter on the counter, the robot adjusts its position for placing the object to better reach the free space on the counter.

1.2 System Overview

In this section, we describe the hardware and software components of our system, required to realize the application scenario described above.

We use a B21r mobile robot from Real World Interface. Its wheels allow this round robot to move forward and turn around its center axis. It has two 6-DOF lightweight arms from Amtec mounted on it. Despite their human-like length they are still much more limited than human arms as they have a very inhomogeneous working space. For the vision system the robot is also equipped with two high dynamic range cameras which are mounted on a pan-tilt unit from directed perception. Localization in the kitchen is done using a SickLMS400 laser range scanner, and the integrated odometry from the wheels.

This robot is not the most advanced hardware platform available. Our goal is to to develop algorithms that run on many different robotic platforms. Therefore, we have designed our software and hardware interfaces such that we expect the same control program to run on our DLR Arm [**?**] and DLR-HIT Hand [**?**], which we are currently installing.

With the increased dexterity and sensing that this platform provides, we expect to be able to improve the performance and robustness of our robot's behavior.

An overview of our system is depicted in Fig. 7. Here, the focus is on processes, not on models they operate on. The system uses a variety of modules that communicate over two middleware systems: Player from the Player/Stage/Gazebo project [10], and YARP [26] that is developed in the RobotCub project. Player provides a variety of hardware drivers and is mainly used for low-level communication with the robot. Examples are receiving laser sensor data, controlling the manipulators, communicating to the base navigation system, and moving the pan/tilt unit of the cameras. Player also comes with some standard robotic algorithms. We currently use the Adaptive Monte Carlo Localization algorithm for localization, pmap for map building, the AMCL Wavefront Planner for global path planning. We added our own interfaces and drivers, where necessary. For instance, we implemented an interface to the Kinematics and Dynamics Library (Orocos-KDL), a navigation controller, and a driver that makes MSL for RRT-based motion planning available to the system. In Fig. 7 all processes that are based on Player modules or external libraries are indicated by a white background.

YARP on the other hand is used for communicating information between higher level systems. YARP and Player co-exist peacefully, and we also built middleware 'bridges' that allow to send messages to both systems.

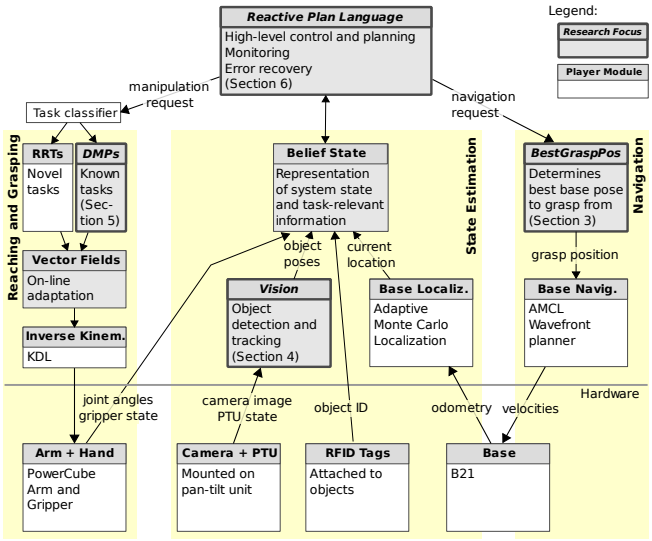


Fig. 7 System overview. The focus of this paper is on the modules indicated with thick edges. For abbreviations, see Section 1.2.

For debugging and efficient data collection purposes, we also use the Gazebo simulator [10]. Gazebo is a high-quality 3D multi-robot simulator for indoor and outdoor environments that uses the Open Dynamics Engine (ODE) library for a proper simulation of rigid body physics. The simulator is especially useful for gathering large training data sets required for learning action models, as described in Section 3.2.

In our experience, using modules from Open Source projects, e.g. Player, YARP, Orocos-KDL, and MSL has

enabled us to quickly solve tasks that are necessary to develop a running system. This has allowed us to focus on tasks that are within our specific research interests. The main research foci we present in this article are indicated by boxes that are surrounded by thicker edges in Fig. 7. These will be discussed in Sections 3 to 6.

The belief state plays an important role in our system. In most systems that use plan-based control, the belief state contains only abstract concepts. This abstraction makes planning tractable. However, this abstraction often abstracts away from aspects of the behavior that are very relevant to understanding the behavior of the robot. For a flexible adaptive planning system, it is for instance necessary to know on a lower level what caused a failure. Only then can the robot understand the failure, and change future plans to forestall the failure. Therefore, our belief state receives all the information from the state estimation modules. The planner chooses when and at what level of abstraction it requires which information from the belief state. Of course, local low-level control loops between different modules also exist, for instance in between the Vector Fields and Inverse Kinematics, and the localization and navigation modules of the base. For clarity, these low-level loops are not depicted in Figure 7.

2 Related Work

In recent years, many novel mobile manipulation platforms have been introduced. Platforms that have been explicitly designed for manipulation and locomotion/-navigation include humanoid robots [1, 4], single manipulators on mobile bases [19, 32], bi-manual torsos on mobile bases [47, 48], and hybrid solutions, where the base is mobile, but the arm is not [40]. Vision is currently the predominant form of perception for manipulation for these and other mobile platforms [20]. Another trend is that many of these platforms use Rapidly-exploring Random Trees (RRTs) [22] or related motion planning algorithms for navigation and/or manipulation planning. As discussed in the Section 1, our focus is rather on learning skills for routine tasks, and optimizing these skills over repeated execution, rather than developing methods that can deal with all possible situations by relying on search. We also focus on generality, for instance by downloading 3D models (Section 4.2) or instructions [27] from the World Wide Web. Another example is learning success models from observed experience (Section 3.2), rather than computing these probabilities based on explicit models of the robot and the environment; models which often require and assume complete and accurate knowledge of the world state [20].

The ability of the service robot ARMAR-III to fill and empty a dishwasher is demonstrated by Asfour et al. in [1]. The impressive robot solves the task well, but under strong assumptions about the environment that allows the usage of a limited perception system only handling simple and well known objects. A more sophisticated perception system for robot manipulation is presented by Kragic et al. in [21]. It is based on a large set of distinct methods to localize known and novel objects. However, they assume the 3D models for most objects as available and lack an ability to acquire new models. The approach also uses a strict heuristic for the order of different visual cues to be extracted. Saxena et al. propose in ([37]) a method to learn grasp points of novel objects purely vision based. In contrast to this, our method tries to find a good 3D model to fit it, while they try to avoid the usage of 3D models. The ability of vision systems to reliably scale so that they can handle a large number of everyday objects has not been demonstrated [20]. We believe that acquiring 3D models from the Internet is an important step in this direction.

Another important research topic in this paper is learning compact models. With regard to learning models of robot capabilities, Zacharias et al. addressed the issue of capturing robot workspace structure [51]. Their capability maps are generated by separating the workspace into discrete regions and trying to solve multiple inverse kinematics queries for every region. A capability map helps to deduce the dextrous workspace of a manipulator, which is useful for positioning the robot relative to an object. However, capability maps do not consider context information and therefore are not optimized for a given environment. Although new versions of the capability map consider arm motion to reach a certain pose [52], our approach explicitly takes the motion system into account and optimizes the initial position so that the following process of trajectory generation is as easy as possible. Most importantly, the success models we learn (Section 3.2) are tailored to the set of motion primitives our robot has (Section 5.2).

Haigh et al. [11] and Belker et al. [3] also developed robots that optimize their high-level plans with action models. These models are used to determine the best path in a hallway environment. Our approach rather focuses on optimizing the parameterization of subgoals at fixed points in the plan (i.e. where to navigate to grasp the object), and is not limited to navigation plans or tasks. A different thread in learning robotic action models from observed experience focusses on models that monitor the execution of actions, rather than predicting their outcome up front given a parameterization [9, 7].

The trajectory imitation technique we use in Section 5 is based on the work by Ijspeert et al. [16] on Dynamic Movement Primitives. Instead of imitating just one example trajectory, we analyze human data to derive stereo-typical trajectories, and imitate (combinations of) those. A related approach for reaching and grasping uses Gaussian mixture models to encode a set of trajectories [6] that are taught to a robot using kinesthetics (the human teacher moves the robot's actuators). These models contain the allowed variance and generalize to

new task settings. Our approach observes data from real human subjects while doing reaching movements in the presence of obstacles, and use different models depending on the situation. The situation (e.g. position of the obstacle) affects the the trajectory in a causal matter, and our model takes into account different strategies for accomplishing the same reaching task (avoiding the obstacle on the left, right, or above it).

3 Navigating to Grasp

Mobile manipulation requires not only navigation and manipulation skills, but also knowledge of the interactions and coupling between them. Simply navigating to a place that is close to the target object is not enough. One of the questions implied by a Turing test for action is the one concerning where the robot should stand in order to perform its manipulation actions. Answering the question seems trivial: simply go to a position such that the target of manipulation is well in reach. However, a more careful look at the question raises some serious issues: What is a good place in the context of an intended manipulation action? Can we have a least commitment realization of places such that the robot can refine a place as it learns more about the context (e.g. the clutteredness) of the surroundings? How can the concept of places deal with the issue that the self localization of the robot might be uncertain and inaccurate? How can we deal with the issue that poses might be more or less appropriate for performing a manipulation task? How can we make the concept of action-related place support conjunctive manipulation actions? That is given two actions a_1 and a_2, what is a place where the robot can perform both a_1 and a_2 and how well can it perform them.

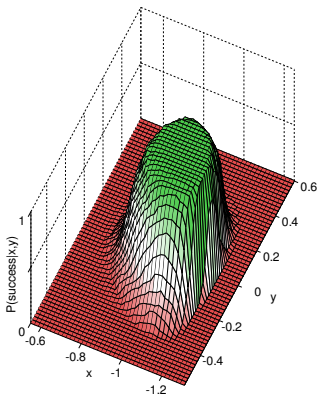


Fig. 8 Probability of successful manipulation, given the robot's position at the table. This is the concept of 'place' for this particula task.

In our system we have realized the concept of action-related place in the following way. An action-related place is a set of regions (robot poses), from which the respective manipulation action can be performed successfully. Fig. 8 shows a distribution that visualizes the probability of success when performing a manipulation action from a given initial position. The plateau on the top consists of a region of connected locations, where the probability of success is sufficiently high. To find the best position arbitrary cost functions can be used to optimize a utility function that considers secondary constraints like time, distance, or quality measures of the arm trajectory. In this paper we take the success probability as our utility function. Additional constraints such as previously unknown obstacles are dealt with by eliminating the poses of a place that violate constraints, for example those that might cause collisions. The uncertainty the robot has about its own location is also taken into account.

Action-related places for multiple actions are then the intersection of the places of all individual actions. Assuming that their success probability is independent of each others, the poses in the intersection are valued with the product of the probabilities of each single action. Fig. 9 illustrates this for the task of concurrently grasping two cups. The yellow hull shows all places, where the yellow cup can be grasped with the left arm. The blue hull shows all places, where the blue cup can be grasped with the right arm. Finally, the green hull shows all places, where the grasping of both cups is possible.

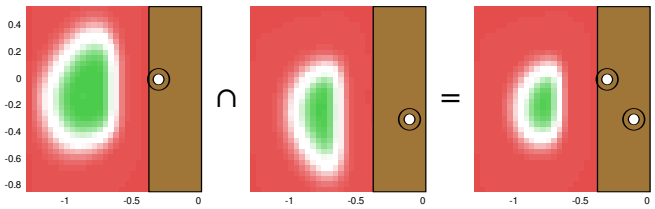


Fig. 9 The concept of 'place' as a distribution of probabilities of successful task execution. The black cell is the maximum of the distribution. Left distribution: grab cup with left gripper. Center distribution: grab cup with right gripper. Right distribution: Grab both cups with left/right gripper respectively. The probability of success of the latter is the product of both individual distributions.

In the remainder of this section we will describe and discuss how places can be represented compactly as a function of the position of objects to be manipulated and how this action-related concept of place can be learned automatically from experience.

3.1 Gathering Training Data

The robot first gathers training data by repeatedly executing a navigate-reach-grasp action sequence. To acquire sufficient data in little time, we perform the training experiments in the simulator presented in Section 1.2. Reaching and grasping are performed using exactly the

same Dynamic Motion Primitives and Vector Fields that we use on our real robot. These methods are presented in Section 5. Moreover we designed a precise model of the real robot in the simulation to ensure that the data acquired in simulation is completely applicable to the real robot.

The action sequence is executed for a variety of task-relevant parameters. In our scenario we tried to grasp a cup and the task-relevant parameters were the x, y position of the cup on a table. The 12 cup positions that the robot used for training are depicted in Fig. 10. For each cup position, the action sequence was executed 350 times. The initial position for reaching and grasping was randomly sampled, and the result whether the robot was able to grasp the cup or not, was stored in a logfile.

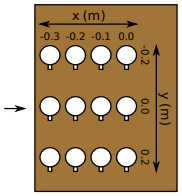


Fig. 10 The positions of the cup on the table.

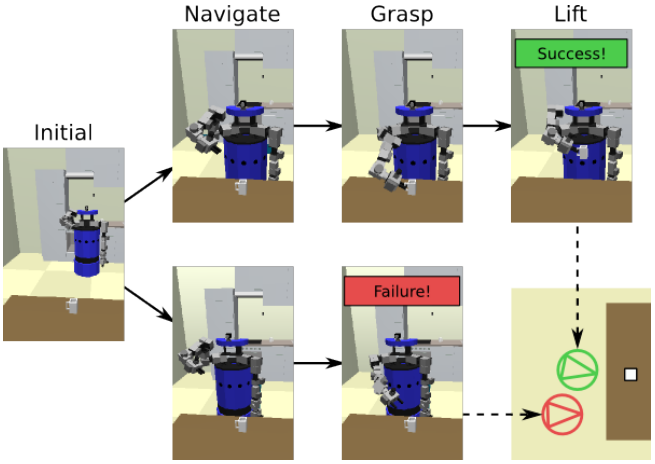


Fig. 11 Two experiment runs with different samples for the robot position. The navigate-reach-grasp sequence in the upper row succeeds. It fails in the lower sequence because the robot is too far away from the cup.

3.2 Learning the Success Model

To learn the success model, we compute a hull around the successful samples. This hull is a simple classifier: successful positions should be in the hull, and failure examples should lie outside. We compute the hull by performing a Delaunay triangulation on the successful points. Since the *convex* hull usually contains failures, we 'tighten' it around the successful samples by only keeping a triangle if each supporting point in the triangle is amongst the thirty nearest neighbors of the other two points. As a result, triangles with very long edges are filtered out. Fig. 12 depicts resulting hulls for different configurations of task-relevant parameters. Here, we make the strong assumption that no failures lie inside the 'tightened' hull around the successes. For our scenario, this approach has proven sufficient, but in future work we will investigate the use of Support Vector Machines and AdaBoost with decision stumps. These approaches provide more accurate classifications, but essentially also provide classification hulls, which can be processed by our algorithm just as the hulls depicted in Fig. 12.

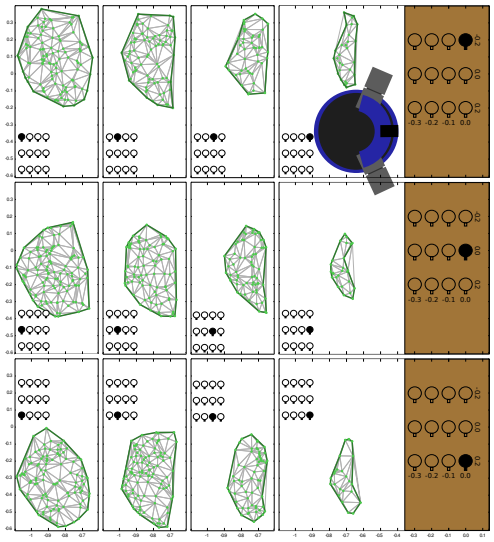


Fig. 12 Successful grasp positions and their 'tight' hulls. Every sub-image shows the hull that corresponds to the cup position that is visualized with the black cup.

For each of the 12 different cup positions, we get one hull. In the next step, we compile all hulls into a generalized compact representation using a Point Distribution Model (PDM), which is a well established method in the field of face recognition [49]. As input a PDM requires n points that are distributed over the contour. We distribute 20 points equidistantly over each hull, and determine the correspondence between points on different hulls by minimizing the sum of the distances between corresponding points, while maintaining order between the hull points. This yields the point distribution depicted in Fig. 13. Only 4 of 12 hulls are shown for clarity.

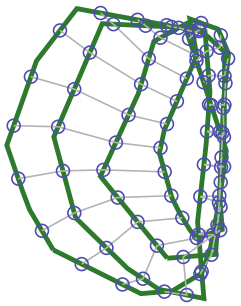
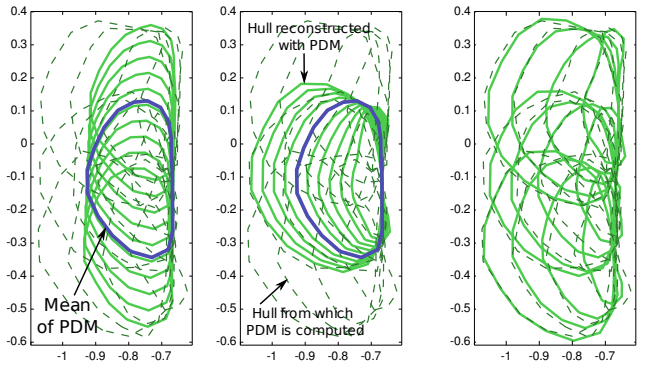


Fig. 13 Aligned points on the hulls.

Given the aligned points on the hulls, we compute a PDM. Although PDMs are most well-known for their use in computer vision, we use the notation by Roduit et al. [33], who focus on robotic applications. First, the 2D hulls are merged into one 40x12 matrix $\mathbf{H}$, where the columns are the concatenation of the x and y coordinates of the 20 points along the hull. Each row represents one hull. The next step is to compute $\mathbf{P}$, which is the matrix of eigenvectors of the covariance matrix of $\mathbf{H}$. Given $\mathbf{P}$, we can decompose each hull $\mathbf{h}_k$ in the set into the mean hull and a linear combination of the columns of $\mathbf{P}$ as follows $\mathbf{h}_k = \overline{\mathbf{H}} + \mathbf{P} \cdot \mathbf{b}_k$. Here, $\mathbf{b}_k$ is the so-called deformation mode of the k^{th} hull. This is the Point Distribution Model. The first two deformation modes are depicted in Fig. 14(a), where the values of the first and second columns of $\mathbf{B}$ are varied between their maximum and minimum value.



(a) First and second deformation mode in $\mathbf{B}$.

(b) Reconstructing the original hulls with the PDM.

Fig. 14 Generalizing over the hulls with a Point Distribution Model.

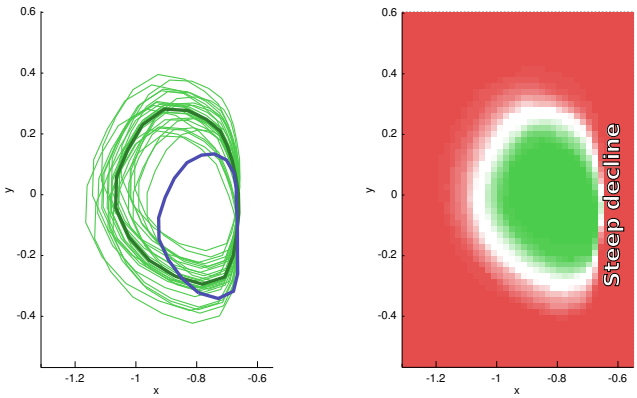
By inspecting the eigenvalues of the covariance matrix of $\mathbf{H}$, we determined that the first 2 components already contain 96% of the energy. Therefore, we use only the first 2 deformation modes, without losing much accuracy. Fig. 14(b) demonstrates that the original 12 hulls can be reconstructed well when using only the first two deformation modes.

The advantage of the PDM is not only that it substantially reduces the high dimensionality of the initial 40D hulls, but also allows us to interpolate between hulls in a principled way using only two deformation parameters. The PDM is therefore a compact, general, yet accurate model.

The final step of model learning is to relate the specific deformation of each hull (contained in $\mathbf{B}$) to the values of the task-relevant parameters varied during data collection (i.e. cup positions). Since the correlation coefficients between the first and second deformation modes and the task relevant parameters $\mathbf{T}$ (the x and y coordinates of the cup) are 0.99 and 0.97 respectively, we simply compute the linear relation between them with $\mathbf{W} = [\mathbf{1}\ \mathbf{T}]/\mathbf{B}^T$. Given a novel position $\mathbf{t}_{new} = \langle x_{new},\ y_{new} \rangle$ of the cup on the table, this model allows us to quickly compute the area from which a successful grasp can be expected. First, we compute the appropriate values of $\mathbf{b}_{new} = ([1\ \mathbf{t}_{new}] \cdot \mathbf{W})^T$. Then the hull is computed with $\mathbf{h}_{new} = \overline{\mathbf{H}} + \mathbf{P} \cdot \mathbf{b}_{new}$. This hull estimates the area in which the robot should stand to be able to make a successful grasp.

3.3 Computing Successful Grasp Position Probabilities

In our system, an uncertainty is associated with the task-relevant parameters. Therefore, it does not suffice to compute only one hull given the most probable position of the cup as a *place* from which to grasp. This might lead to a failure if the cup is not at the position where it was expected. Therefore, we use a Monte-Carlo simulation to generate a probabilistic advice on where to navigate to grasp the cup. This is done by taking 100 samples from the Gaussian distribution of the cup position, which yields a matrix of task relevant parameters $\mathbf{t}_s = [\mathbf{x}_s\ \mathbf{y}_s]$. The corresponding hulls $\mathbf{h}_s$ are computed for the samples by using the method described above. In Fig. 15(a), 30 out of the 100 hulls are depicted. These were generated from the task relevant parameters x=-0.3, y=0.1, σ_x=0.05, σ_x=0.05.



(a) Sampled classification hulls ($\mathbf{h}_s$).

(b) Discretized relative sum of the hulls.

Fig. 15 Monte-Carlo simulation

We then generate a discrete grid in which each cell measures $2.5 \times 2.5cm$, and compute the number of hulls it is contained in. Dividing the result by the overall number of hulls yields the probability that grasping the cup will succeed from this position. The corresponding distribution, which takes the uncertainty of the cup position into account, is depicted in Fig. 15(b), as well as in Fig. 8.

It is interesting to note the steep decline on the right side of the distribution (in the direction of the table). This is intuitive, as the table is located on the right side,

and the robot bumps into the table when moving to the sampled initial position, leading to an unsuccessful navigate-reach-grasp sequence. Therefore, none of the 12 hulls contain this area, and the variation in $\mathbf{P}$ on the right side of the PDM is low. Therefore, variation in $\mathbf{B}$ does not have a large effect on this boundary, as can be seen in Fig. 15(b). When summing over the sampled hulls, this leads to a steep decline in success probability in the direction of the table.

Due to the uncertainty in the robot's position, it is unwise to choose a position close to steep boundaries in the probability distribution. Even if the prediction of grasping the cup is 'success', a steep decline can lead to a failed reaching action, if even only a small localization error causes the robot to be on the wrong side of the decline. The proposed position for grasping an object should therefore take the uncertainty of the robot's position into account. To do so, we convolve the grid of probabilities of grasp success with the discretized $(2.5 \times 2.5cm)$ probability distribution of the robot's position. The result can be seen in Fig. 15(b). Note that this convolution not only works for a Gaussian distributions, but also for multi-modal distributions as returned by particle filters. The goal position recommended to the navigation module is the position that corresponds to the maximum value of this distribution, so that grasp success probability is optimized.

Fig. 16(b) depicts how the probability distribution is affected by varying task relevant parameters. Please notice how in the first row, it becomes 'more difficult' to grab the cup (i.e. less likely to succeed) as the cup moves away from the table's edge.

3.4 Advantages and Shortcomings

We believe our system has three main advantages. First of all, the learned model is very compact, with only 2 (deformation) parameters that are related to task-relevant parameters. On the other hand, as it is learned from observed experience, the model is grounded, and takes into account the robot hardware, its control programs, and interactions with the environment. These aspects do not have to be explicitly modelled manually, as the system is essentially considered to be a black box. Of course, this implies that the models are tailored to the system as a whole, and cannot be ported to other platforms. The method itself however is in principle applicable to any platform. Also, as the data is heavily compacted in the model, querying the model on-line is very efficient. This is an advantage of 'compiling' experience into compact models, rather than running a novel search for each situation.

Second, the output of this model is a *place*, which is an area, instead of a specific position. Rather than constraining itself to a specific place prematurely, this representation enables the robot to optimize secondary criteria within the area. Examples are execution duration, or determining the best position for grasping two objects simultaneously. In previous work, we proposed *subgoal refinement* [41] for optimizing such secondary criteria with respect to subgoals.

Finally, the compact model enables us to compute the probability of successful grasp given a position, which takes the uncertainty of the robot's location and the task-relevant parameters into account. Optimizing the probability of successful grasping leads to more robust and legible behavior. The reason is that unnecessarily complicated initial positions for starting the grasping process are never faced. Therefore, motion planners or motion controllers for reaching and grasping are able to work faster and more effectively, because they find themselves in initial states from where the state space can be explored well. This leads to more natural and legible behavior.

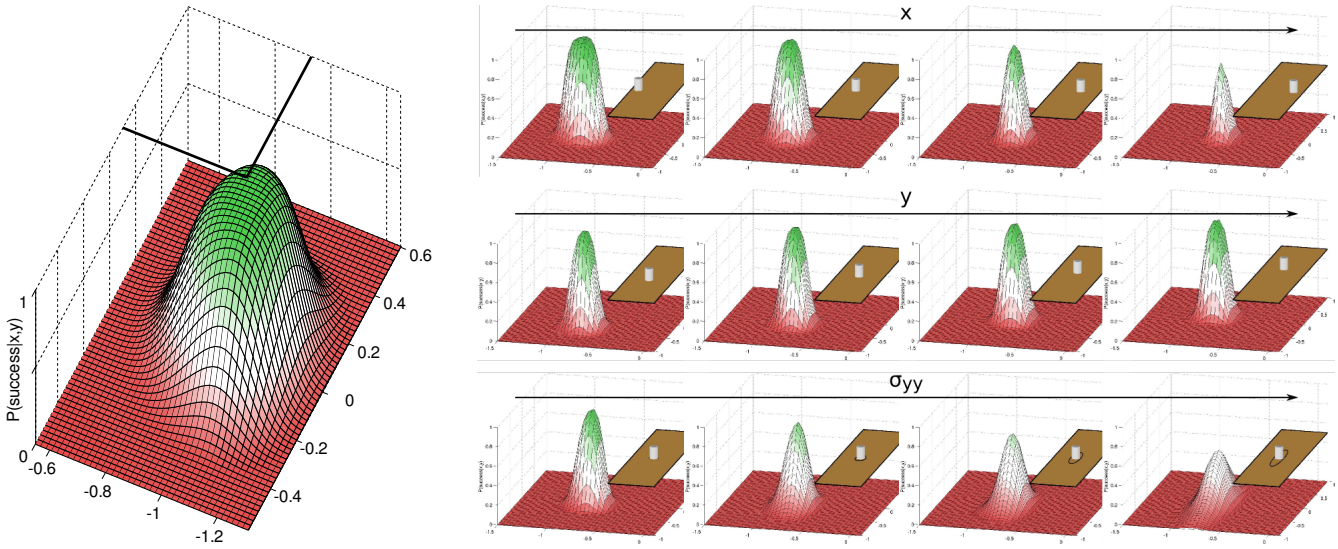
Gathering the data is a time-consuming process. Therefore, we used the Gazebo simulator to automate and parallelize data acquisition. The goal of many current robotic applications is to achieve continuous extended deployment in human environments. When this is achieved, the robot will be able to acquire over time. The challenge is then not so much gathering the data, but recording, classifying and storing it in a principled way.

As previously discussed, we require more sophisticated classification algorithms to determine the classification hull. We are considering Support Vector Machines and AdaBoost with decision stumps. In this scenario, a linear relationship between the deformation modes $\mathbf{B}$ and the task-relevant parameters $\mathbf{T}$ arose. We expect that other data might lead to non-linear functions, and a different mapping will have to be computed or learned.

The methods described in this section are quite general, and not limited to the application scenario of navigating to grasp. We have also done preliminary work on computing PDMs for the classification and generation of successful grasps and reaching motions, in which obstacles are avoided. The main issue here was the high dimensionality of the input spaces. We reduced these spaces to a 2D or 3D manifold using Locally Linear Embedding [35], and computed the PDM on the classification hulls in these reduced spaces.

4 Perception for vision-guided reaching and grasping

To enable vision-guided reaching and grasping, knowledge about the location of the target object and relevant obstacles is required. In this section, we describe the vision-based perceptual system that localizes objects. Depending on the given knowledge about present objects, we distinguish between three different classes of task-related object perception: 1) find and locate a **known object**; 2) recognize and classify a present **new**



(a) Distribution for task-relevant parameters: x=-0.3, y=0.1, σ_x=0.05, σ_x=0.05.

(b) Influence of various task-relevant parameters on the distribution.

Fig. 16 Final distributions, after convoluting the uncertainty in the robot pose with a distribution as depicted in Fig. 15(b).

object; 3) find and locate an unknown object via a semantic **object class** description. While the first task is currently well-studied, the other two tasks lead to several problems which lack a general solution. In the following section we will discuss our solutions for those tasks. We first describe a multi-model approach to treat known objects. Then we show possibilities to index new objects, and we finally show how we model new classes of objects using Internet databases.

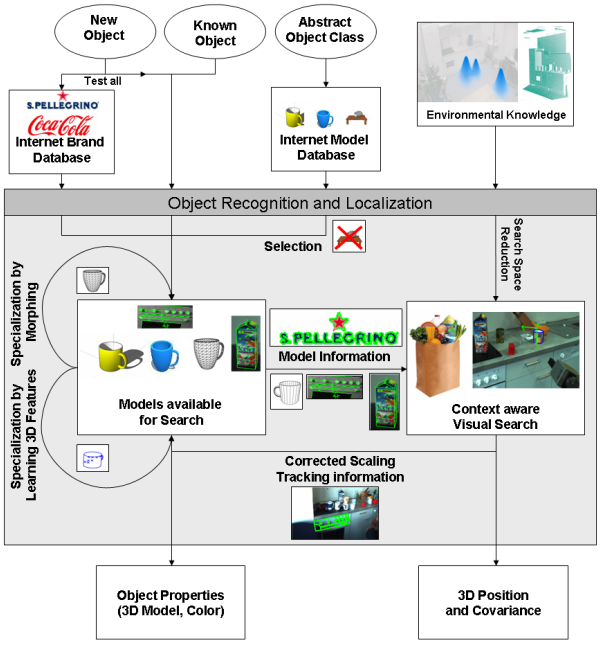


Fig. 17 Overview of the localization methods.

An overview of the perceptual system, including the different classes of object perception, is depicted in Fig. 17. The input of the perceptual system is 1) an object or an object class ID; 2) a position prior defined by a set of possible positions with corresponding covariances. The output is also a position with a covariance matrix. The selection of the algorithm is based on an internal model learned via internal success and time statistics. In case of a query using an object class ID the number of entities to be searched can be specified.

4.1 Localizing Known Objects

The complexity of localizing a known object depends on the available models of the object. The more accurate the model, the more accurately and robustly we can localize the object. We mainly model two kinds of object knowledge: appearances and location priors.

Appearance Models For modelling appearances we use the following kind of models with the corresponding matching algorithms:

3D CAD models, with a matching method based on [46].
3D descriptors, with a matching method based on [23].
2D contour models, with a matching method based on [14]
Color distributions

This list also represents the hierarchy which allows us to infer models top-down, i.e. from a 3D CAD model we derive the other models, given that the object is visible.

Location Models Location priors are modeled using probability distributions in space and rotation. We connect world, robot, actuators and perceived objects with a tree

of possible coordinate frames. 3D positions have 6 degrees of freedom, so we use a 6×6 covariance matrix to propagate uncertainties. The initialization is done by filling the diagonal of the covariance matrix with the estimated deviation of the 6 position DOFs. A propagation is calculated with an unscented transformation, by transforming a sample of possible points into a new coordinate frame, and extracting the major components of the new covariance matrix via a principal component analysis [18]. Movements of the robot cause a reorganization of the tree: all moved parts are copied before the movement is propagated. Therefore, any percept is relative to the next stable parent in the tree, which are mostly world coordinates. For example, other system modules using perception information need to know the state of to the pan-tilt unit (PTU) at the time the localization was performed. As soon as the PTU moves, all localization still connected to the PTU's tree-node must be redirected to the older PTU-state that will be a new node in the tree.

Learning the Optimal Algorithm For optimal localization, we have to consider several properties of the algorithms, search space restrictions and the correlations between them. We learn quality measures in three dimensions in that all algorithms have different properties: reliability, accuracy and computational cost. In this area we still rely on exhaustive tests to observe rates of success and times of calculation in order to select the best algorithm for a task, that has specific requirements for the three quality dimensions. Additionally, the perquisites on the object itself differ from algorithm to algorithm. For example, the matching with 3D CAD models is a relatively slow approach with a high false positive rate, while the contour models are faster and more robust, but only work for partial planar objects. For manipulation, the individual methods are still not reliable enough, so a verification is needed. We use two verification mechanisms to improve the results: either apply the same method to a slightly different view, or apply a different method to the same view. The verification expects similar results in either case. A verification is triggered as soon as the probability of a result being valid is lower than a threshold quality measure. These verification methods are included as additional atomic algorithms during the learning of the optimal algorithm for a certain problem.

4.2 Localizing Novel Objects

In human living environments, novel objects appear on a regular basis. The robot must learn to classify and localize them. For example if a grocery bag arrives in the kitchen it contains most probably novel object. We see two possibilities to deal with new objects: First, we can build an appearance model of them in order to keep track of the object or relocalize and manipulate it later on. Or second, we can try to classify it into a semantic class and apply then the mechanisms that we will explain in Section 4.3. Those novel objects, which we can not classify, can be either recognized by an attention system observing significant changes in the environment or they can be explicitly taught by a user by showing the robot a new object. In such a situation we use stereo cameras to learn a contour model that allows us a 3D localization and tracking of objects with one camera. This method is restricted for partial planar objects.

4.3 Localizing Novel Objects Based on Their Class

For this problem, we propose a novel method, in which 3D models are automatically retrieved from an Internet database. These models are retrieved based on user-annotated labels, and used in our robotic system for vision and manipulation. This method was proposed in [**?**] and will be sketched here briefly. given a certain object, we pass the string that denotes it, e.g."mug", "cooking pot" or "frying pan", to an Internet search engine for 3D models. The results are clustered in order to select the most probable subclass matching our request, based on spatial properties like the shape distribution function [28]. All results in the main cluster are then spatially aligned and used as models in a search for known object that was described before in Section 4.1.

4.4 Results of the Specialization for the Pick-and-Place Task

Our scenario included several tasks for the perceptual system: 1) detect a mug on a table; 2) localize a mug on a table; 3) localize possible obstacles. Instead of an explicit model, the input to the system was a class label "mug" connected with an ID. This class label served as search string for an automatic Internet query at Google 3D warehouse. The results for clustering of the query (see Fig. 3) can be seen in Fig. 18. The first cluster is the largest and contains the model that was finally chosen.

This decision is made by trying to match this model with the 3D CAD based algorithm mentioned before. It is applied on scenes that could contain the queried object. Examples for such location are "on the table" or "in the cupboard", that can be seen in Fig. 4. Pose restrictions are based on a semantic map, and the best fitting models are kept.

Given a well-fitted 3D model, the best algorithm is determined given a set of different views of the object. The resulting best algorithm for our test object was the 3D CAD model based matching with a validation via the same algorithm on a second view/camera with a known external camera calibration. The results for the mugs in our test scenario for the vision system are convincing: We detected nearly all available mugs (about 90%) while we nearly only had false positives when there was no

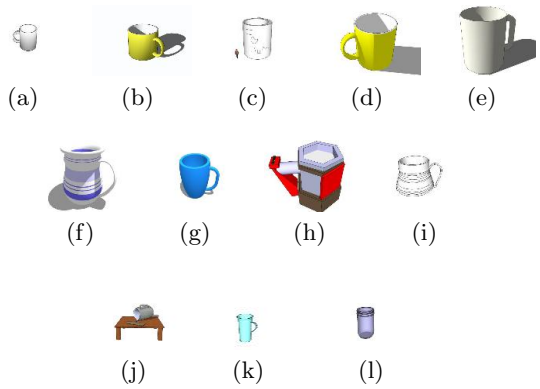
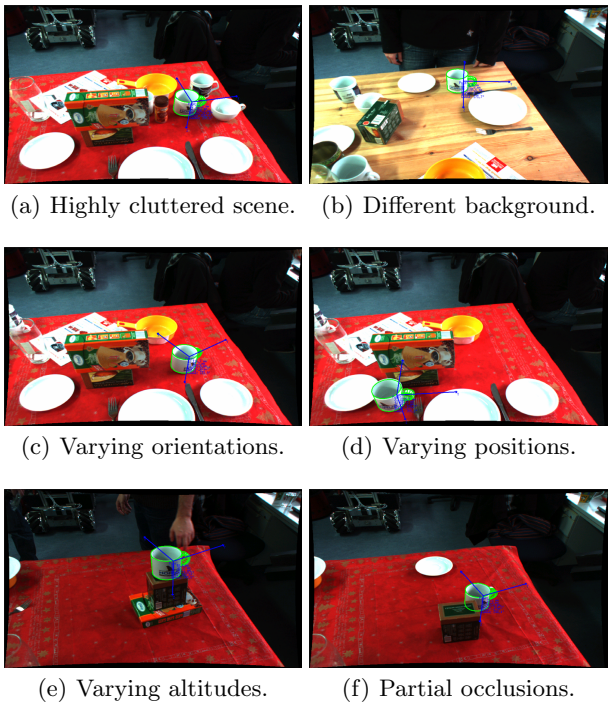


Fig. 18 The result four cluster for the query for "mug", a - e are inliers, all the other are counted as outliers (cluster are f-i, j-k and l).



(a) Highly cluttered scene. (b) Different background.

(c) Varying orientations. (d) Varying positions.

(e) Varying altitudes. (f) Partial occlusions.

Fig. 19 Several matches during our experiments.

	Mug in scene	no mug in scene
Localized	46	19
No mug detected	4	31

Table 1 Evaluation of localization of mugs.

mug available (about 40% false positives in scenes without mugs), see Table 1. The average localization error for true positives was below 2% in space and rotation. The typical scenes we used for testing, can be seen in Fig. 19(a) to 19(f).

5 Human-like reaching and grasping

Humans usually perform manipulation tasks with highly stereotyped movement patterns [50]. The optimal control framework assumes that these typical patterns for reaching and grasping are the result of a very long and multi-facetted optimization process [45]. The optimization criteria used in human motion control are not limited to those typically considered by today's motion planning algorithms, but also include additional aspects such as the accuracy of motions [12].

There are good reasons that the motions of robots for everyday manipulation tasks in the presence of, and in cooperation with, humans should be similar to human reaching behavior. First, robots with human-like motion will enable humans to more easily perform perspective taking and intention recognition [29]. This is necessary to enable implicit coordination (which humans use when coordinating their actions) in joint human-robot tasks. Second, we believe that robots acting in human environments must have the capability to learn advanced manipulation skills through imitation learning [38,5]. The task of transferring observed reaching and grasping behavior into the robot's motion control system becomes much easier if both motion control systems apply similar control strategies. Third, humans can sequence motion primitives seamlessly [8,42]. We expect that robots that have the same motion primitives as humans are able to achieve similar smooth execution of sequences of motion primitives.

5.1 Stereo-typical human reaching trajectories

To acquire stereo-typical human reaching trajectory, we performed an experiment in which a subject was asked to reach and grasp a target glass on a table. A second obstacle glass was placed at different positions on the table in each episode. The goal of this experiment is to answer quantitavely 1) at which positions do obstacles lead to human reaching behavior which is different from the default behavior when no obstacles are present? 2) which reaching strategies do humans use to avoid the obstacle?

The reaching motions were captured with a Polhemus Liberty magnetic position/orientation tracker. One sensor was attached to the hand, as depicted in Fig. 20, and another sensor was attached to the glass to measure the exact time when the lifting movement started. Before performing the experiment we used one sensor to measure the positions of the obstacle grid in the tracker's coordinate frame. All sensors are tracked with very high precision at 240Hz.

In the experiment, the subject sits at a table, and is asked to repeatedly reach for, grasp, and lift a target glass. The hand always starts in the black square in Fig. 20. Before each reaching motion, an obstacle glass is placed on different positions on a grid on the table.

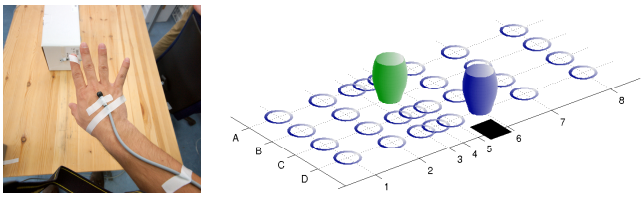


Fig. 20 Location of the sensor (left). Locations of the cups (right). The green glass is the target glass, which is always at position B4. The blue glass is an example obstacle at position D6. The flat black region is the initial location of the fingers.

The grid is 40x80cm. In Fig. 20 for example, the obstacle glass is at position $D6$. The target glass is always at position $B4$. The obstacle glass was placed 10 times on each of the 29 positions. Furthermore, 30 reaching motions were performed without any obstacle glass. These are the '*default*-trajectories'[1]. The order of obstacle placement was random, to avoid learning effects. The total number of reaching motions is therefore 29*10+30 = 320.

To determine the effect of the obstacle on the reaching trajectories, we defined a distance measure d between the *default*-trajectories and the sets of other trajectories, $A1 \dots D8$. We used the trajectory comparison approach described Roduit et al. [33]. Here, the difference measure between two sets of trajectories is computed by 1) computing a point distribution model of the two sets of trajectories 2) taking only the first n components, by inspecting the eigenvalues of the covariance matrix of the merged trajectories 3) computing the Mahalanobis distance between the coefficients of the two sets of trajectories. A more detailed explanation of this method can be found in [33]. In Fig. 21, the height of the cups represents d for that obstacle position.

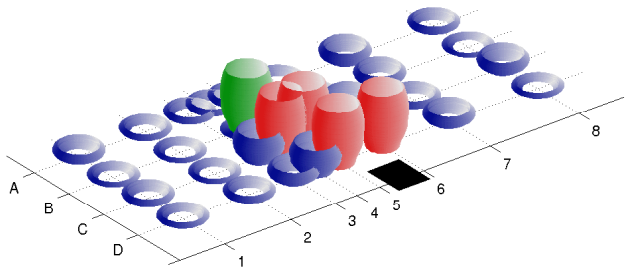


Fig. 21 The height of the glass represents d for that obstacle position. d_{thres} is 16.4 for this graph. The red glasses (at $C5$, $D6$, $D5$, $C4$) lie above this threshold.

We automatically determine an appropriate threshold on d (called d_{thres}) by determining the valley point of the histogram of the d values. The distance d between the $C5$-, $D6$-, $D5$-, and $C4$-trajectories and the *default*-trajectories is higher than this threshold. These four positions are depicted as red glasses in Fig. 21.

The next step is determining prototypical trajectories that represent qualitatively different strategies for avoiding the obstacle. We do so by performing a k-means clustering on the 40 trajectories in the $AVOID$-trajectories set As Jenkins et al. [17], we perform the clustering using several different spaces, and compare the results. We use: 1) a 300D space, in which the 100 x, y and z coordinates are simply concatenated for each trajectory; 2) a the 3D space of a PCA performed on the 300D trajectories above; 3) a 3D space computed with Local Linear Embedding (LLE) [35] from the 300D space (using 10 neighbors). In each space, the distance between two trajectories is determined by the angle between the two n-dimensional vectors representing the trajectories. The number of clusters is set to 3 manually. Clustering in the three spaces yield almost exactly the same clusters (the three spaces only disagree on the categorization of 3 trajectories). This implies that 1) these clusters are good stereotypes, and do not just depend on the clustering space or method; 2) only a very compact representation in 3 dimensions are needed to determine these stereotypes. The clustering algorithm in the 300D space yields the three clusters depicted in Fig. 22.

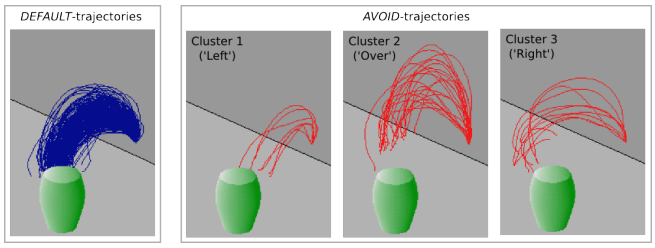


Fig. 22 Clustering the $AVOID$-trajectories.

The average trajectories for these sets are plotted in Fig. 23. Clustering the $AVOID$-trajectories yields stereotypical reaching movements, which we label 'over', 'left', and 'right', denoting the direction in which the obstacle is avoided. We call these the 'principal trajectories'. What is very interesting about these trajectories is that they are not qualitatively different from each other, but are rather variations of the default behavior. From the top view for instance, it is apparent that the 'over' strategy almost perfectly follows the default behavior in the xy-plane, and is therefore simply a version of the default behavior, scaled in the z-plane. Similarly, 'left' and 'right' strategies hardly vary from the default behavior in the z-plane.

Another reason why we believe these trajectories are good stereotypes is because linear interpolations between them are also found in the actual human reaching trajectories [43]. This is actually the reason why we chose

[1] For convenience, we sometimes informally refer to 'the trajectories that arose from the reaching movement that was performed when the obstacle was placed at position $D6$' as 'the $D6$-trajectories'.

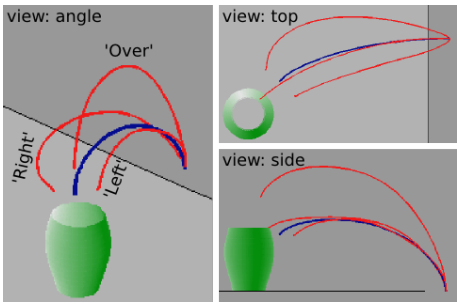


Fig. 23 The four stereo-typical trajectories.

3 clusters. When choosing 4 or 5 clusters, some of the clusters could be reconstructed by choosing linear interpolations of the other clusters. With 3 clusters, there is no redundancy, and the model is as compact as possible.

5.2 Stereo-typical trajectory execution with Dynamic Movement Primitives

We now describe how the compact model is realized on our autonomous manipulation platform to reproduce the principal trajectories, and extrapolate from them using interpolation.

In the control system, each principal trajectory is represented by a Dynamic Movement Primitive (DMP) [16]. One DMP was trained for each principal trajectory with the regression learning algorithm described in [13]. Some advantages of DMPs are 1) within a certain range, they generalize to other goal locations 2) compliance 3) convergence to the goal location is guaranteed. The controller used to execute the trajectories generated by the DMPs is depicted in Fig. 24.

It is worth noting that the compact models contain only kinematic information, more specifically the coordinates from the hand in Euclidean space. This is easily observable from human subjects, in contrast to internal dynamic states like forces and torques. We rely on sophisticated inverse kinematic algorithms and low level controllers for successfully tracking the generated trajectories with the robot.

The trajectory generated by the DMP module is taken and fed to a work space single point attractor, which takes the next intermediate point in the trajectory and pulls the end-effector to this intermediate goal until it is reached. The output of the single point attractor is the desired velocity vector which is given to the velocity based inverse kinematics controller which generates the velocities in joint space. We use the damped least squares inverse kinematics algorithm from [24] as implemented in the Orocos-KDL library [39] which achieves more stable behavior around singularities.

To adapt to changes in dynamic worlds, we use Vector Fields, and integrate them in the DMPs as described in [30]. This will allow us to take advantage of robot arms that have force control (Kuka-DLR arms): In cases

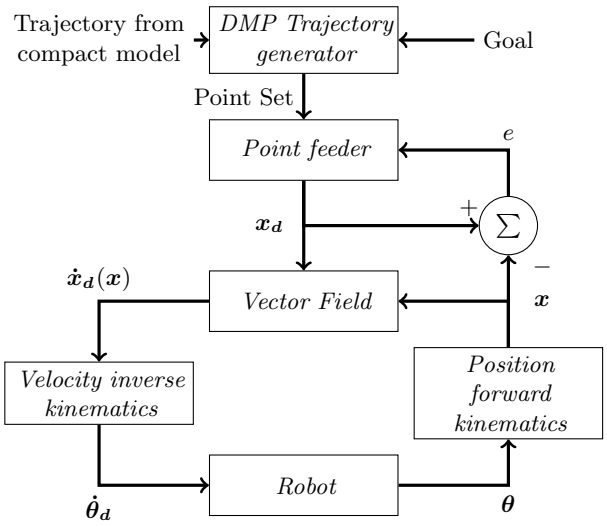


Fig. 24 Block diagram of the motion controller. $\boldsymbol{x}_d$, $\dot{\boldsymbol{x}}_d(\boldsymbol{x})$ and $\boldsymbol{x}$ represent the desired intermediate point, the velocity vector and the current position in work space. $\dot{\boldsymbol{\theta}}_d$ and $\boldsymbol{\theta}$ correspond to the velocity and angular vectors in joint space. e is the error used to determine how near to the current desired point is the end-effector.

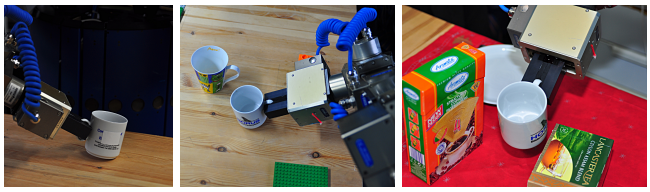


Fig. 25 Several examples of succesful grasps during the experiment.

when the arm is pushed too far away from the stereotypical motion trajectory dictated by the DMPs it will be controlled by the vector field, and will still give a smooth movement to the goal. This fits well within our general approach of assuming that many tasks can be solved with standard solutions (i.e. the stereo-typical DMPs), but that we also need strategies for dealing with unforeseen events (i.e. on-line motion adaptation with Vector Fields). Although not a dynamic object, we also include the table as a repellor in the Vector Field, to avoid collisions of the hand with the table. The system takes care of avoiding obstacles that move, appear or disappear from the arm reaching space. In the example of grasping a cup from the handle, the vector field has proven to be a very practical solution in dealing with avoiding the obstacle that the same cup represents and guiding the precision movement when the gripper is very near to the handle, giving a movement shape that produces a reaching approach movement that avoids crashing against the handle and the cup. The vector field used in the current system takes the current task position given by the position forward kinematics and using information about obstacles and goals it calculates a task space velocity

vector which is then translated to joint speeds by a velocity inverse kinematics module. Instead of calculating a vector field grid and then querying the current value from the grid, it calculates continuously (using point attractors and point repellors) the corresponding velocity vector for every control cycle. The point attractor and repellor forces are shaped depending on the role that each object has. For instance, cups that are to be grasped are composed of two repellors and two attractors, the first repellor is a sphere that decreases the force influence with an exponential decay as the distance to the current position increases, the second repeller is a *shadow* repeller pushing away from the back side of the cup (this gives smoother movements), one attractor is a normal spherical attractor with no decay thus having a global effect of attraction, and the other attractor is a conical attractor with the tip of the cone in the handle of the cup and the cone increasing its diameter away from the cup, also the force of this attractor decays with the distance (Fig. 26). A result of this system is that under the situation where a robot arm link touches a joint limit because the vector field guides the arm outside of the configuration space of the arm (possibly for avoiding an obstacle), the velocity inverse kinematics algorithm still moves the arm in a similar direction and eventually the arm moves away from this limit and returns to the normal trajectory.

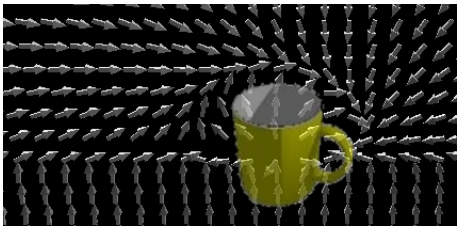


Fig. 26 Vector Field for grasping a cup from the handle.

Player [10] is used for the low-level control of the Amtec Powercube manipulator, and YARP [26] is used for the communication between the control modules described in Fig. 24.

5.3 Outlook

In our current research, we are investigating the optimization of the stereotypical trajectories with reinforcement learning methods. Thus, the robots first acquire trajectories by imitation, and then optimize and tailor them to their own specific hardware, as in [31]. We expect this to lead to optimized execution, with a strong bias towards human-like motion.

A disadvantage of the Powercube arms is that they are not at all anthropomorphic. Although the end-effector might describe a human-like reaching trajectory, the rotation of the joints and arm configurations can be quite unlike humans. This research topic has rather been conducted with the DLR arm in mind. We are currently installing two such arms, and expect much more natural motion from it, especially when executing the stereotypical human reaching motions.

6 Task-level Control: RPL

The task-level controller of our system is designed and realized to achieve flexible and reliable manipulation behavior through compact behavior specifications.

Compactness of the control system is achieved by embedding lightweight reasoning processes to infer situation-specific control decisions and action parameterization. These lightweight reasoning mechanisms free the programmer from spelling out each control decision and parameterization for each conceivable context explicitly. Rather, we can state the routine for the manipulation task in the following way: look at the place where the mug should be. If the the mug is detected then perform a rough position estimate for the cup and determine what it stands on. Use the rough position estimate and the geometric model of the supporting entity in order to determine the place from where the mug can be picked up (as described in Section 3). Upon navigating towards the intended place track the mug to get more accurate pose estimates and update the manipulation place representation according to the refined mug positions, the perceived obstacles, and the uncertainty in the self localization of the robot. Upon arrival at the manipulation place fixate the cup to get a position estimate accurate enough for reaching and grasping (see Section 4). Determine the reaching strategy according to the perceived obstacles on the table, that is whether to reach without considering obstacles, to reach above, or around obstacles. Given the strategy the trajectory is determined as described in Section 5 and the Dynamic Movement Primitive with the respective parameterization is activated. And so on.

Thus the control routine infers many pieces of task-relevant information on the fly. The geometry of the entity supporting the mug and its geometric description, the place where the manipulation is to be performed, the classification of the clutteredness on the table and the expected impact on the manipulation action, etc.

The second key aspect of our task-level control is how the robot achieves the required flexibility and reliability. To do so, the task-level control is specified by concurrent reactive plans implemented in the language RPL (Reactive Plan Language). The RPL plans for the manipulation tasks specify how the robot is to respond to sensory input in order to accomplish its task.

Successful interaction with the environment requires robots to respond to events and to asynchronously process sensor data and feedback arriving from the control processes. RPL provides *fluents*, registers or program

variables that signal changes of their values. Fluents are best understood in conjunction with the RPL statements that respond to changes of fluent values. The RPL statement **whenever** f b is an endless loop that executes b whenever the fluent f gets the value "true." Besides **whenever**, **wait-for** f is another control abstraction that makes use of fluents. It blocks a thread of control until f becomes true.

RPL provides control structures for reacting to asynchronous events, coordinating concurrent control processes, and using feedback from control processes to make the behavior robust and efficient [25].

par $p_1 \ldots p_n$	(**par** (**achieve** (hand-at-pose 'left ...)) (**achieve** (hand-at-pose 'right ...)))
try-all $p_1 ... p_n$	(**try-all** (**achieve** (estimate-pos ... 'camera)) (**achieve** (estimate-pos ... 'laser)))
try-in-order $p_1 ... p_n$	(**try-in-order** (**achieve** (entity-in-hand ... 'left)) (**achieve** (entity-in-hand ... 'right)))
with-policy p b	(**with-policy** (**maintain** (entity-in-hand ...)) (**achieve** (robot-at-pose ...)))

Fig. 27 Some RPL control structures and their usage.

Fig. 27 lists several control structures that are provided by RPL and can be used to specify the interactions between concurrent subplans. The control structures differ in how they synchronize subplans and how they deal with failures.

The **par**-construct executes a set of subplans in parallel. The subplans are activated immediately after the **par** plan is started. The **par** plan succeeds when all of his subplans have succeeded. It fails after one of its subplans has failed. The second construct, **try-all**, can be used to run alternative methods in parallel. The compound statement succeeds if one of the processes succeeds. Upon success, the remaining processes are terminated. Similarly **try-in-order** executes the alternatives in the given order. It succeeds when one process terminates successfully, it fails when all alternatives fail. **with-policy** p b means "execute the primary activity b such that the execution satisfies the policy p." Policies are concurrent processes that run while the primary activity is active and interrupt the primary if necessary. Additional concepts for the synchronization of concurrent processes include semaphores and priorities.

Using RPL's mechanisms for sensor-driven plan execution, for failure detection and handling, and for synchronizing complex concurrent behavior we have realized the task-level control for our manipulation task by using RPLconstructs such as **with-failure-handling** allowing a simple notation for concurrent task monitoring and recovering.

One of the tasks handled by this construct is navigation. The construct **with-failure-handling** consists of three blocks: **perform**, **monitor** and **recover**. All these blocks get executed in parallel. The **perform** block is executing the main plan. In the case of navigation this means that it commands the robot base to drive to a predefined position and waits for a success indicator fluent. In the mean time the **monitor** is observing the current position of the robot. When the intended position is reached the success fluent is set to true and the perform block finishes, indicating to the caller that the plan succeeded. When the **monitor** block detects that the position of the base is not changing anymore but the position is not in the tolerance range, a failure gets generated. The **recover** block gets started by the generated failure and directs the base to move to a random nearby position, and restarts the perform block. The **recover** block is retrying the plan forever, because real life tests showed that the plan is always succeeding after some tries. Still the **recover** block could be simply altered to perform a retry, for example 4 times and indicate that the plan failed. This can trigger high-level failure handling to resolve the error.

An implementation example can be seen in Listing 1.

Listing 1 Example implementation of with-failure-handling for base navigation.

```
(with-failure-handling failure ()
  (recover ((typep failure
                   'navigation-failed)
            (move-to-random-nearby-position)
            (restart-perform)))
  (monitor
    (if (position-not-changing)
      (if (destination-reached-p)
        (pulse continue)
        (fail
          :class navigation-failed))))
  (perform
    (goto-destination)
    (wait-for continue))))
```

7 Results

The platform is evaluated in two sessions. First at a public demonstration, where the main goal was to test the robustness of the system under continuous deployment. A second later session was used for a quantative analysis of the error recovery functionality. In this session,

RPL and the algorithm for determining successful grasp places were also included in the system.

7.1 Public Demonstration

We introduced the system described in this paper to the public on 18.10.2008, at the "Tag der offenen Tür" (open house) of the Technische Universität München. During this demonstration, the robot almost continually performed the application scenario, where it locates, grasps and lifts a cup from the table, and then transports it to a work-surface next to the kitchen oven. It performed this scenario 50 times in approximately 6 hours. Fig. 28 depicts two pictures taken during the demonstration. This successful demonstration has convinced us that the robot hardware and software are robust enough to deploy amongst the general public.

Fig. 28 Some photos of the public demonstration at the Technische Universität München.

7.2 Failure Detection and Recovery

To evaluate the reliability of our overall system and its subcomponents, we test the setup of the public demonstration, which required a human operator to look after the system to detect failures. Then we replaced the simple control program by a more sophisticated RPL kernel, and run a test again under similar conditions. The errors we counted occurring in the system are denoted in Table 2. It can be seen that errors occur frequently in such a complex system, especially the navigation component failed often by not raising the event of a successful arrival, which then caused the system without error recovery to wait for this event forever, or until the human operator intervened. This problem was completely solved by monitoring the navigation. Only once in the tests an unhandled hardware problem in the navigation system caused a failure in the system with error recovery. The errors in the vision system are mainly false positive detected objects or localization with an spatial or rotation error over 2%, which exceeds the tolerance for a successful grasp. Those failures were recovered by monitoring the success of the grasp following the localization. Grasping errors are of two kinds: First, singularities in the vector field forcing the arm to be repelled before reaching the target in a repetitive way and second, by moving the object before the robot was able to grasp it. The first problem could be solved by monitoring the end-effector of entering already visited volumes. The second is detected by monitoring the gripper position on closing, whether it contains an object or not. For the second trial we also added the module that computes a position from which grasping is likely to succeed (Section 3), and allowed us to place cups on the entire table. This enabled the robot to grasp cups that were out of range in the first trial. Twice the system with error recovery did not notice that the pick up failed until it moved away from the table while there was still an RFID signal of the mug on the table, those are two detected errors, which actually occurred in the grasping system but were detected using the RFID antennas.

8 Conclusion

In this article we have presented the current state of our investigations of methods for achieving more general manipulation capabilities for mobile manipulation platforms, which produce legible behavior in human living environments. We have proposed an integrated system that performs simple vision-guided everyday manipulations in a human living environment. The manipulation task is to put typical, easy to grasp everyday objects, e.g. mugs and plates, on the table in order to perform a household task such as setting the table.

In this context, we have extended the basic manipulation capabilities of a mobile manipulation platform in several important ways that increase the generality and compactness of the robot control systems and the legibility of the behavior they produce. These extensions include place- instead of position-based mobile manipulation, object recognition and localization methods that can automatically learn environment specific 3D models for object categories and use the specialized models for improved object recognition and localization, and a method for learning reaching strategies from humans and realizing them through dynamic motion primitives.

	Runs	Succ.	Errors (Occured/Detected)				
			Total ⇒	Navi	Vision	Grasp	RFID
Without ER	32	15	20/0	6/0	4/0	8/0	0/0
With ER	32	31	22/19	7/6	5/5	10/8	0/2

Table 2 Execution episodes with and without error recovery (ER).

In order to achieve generality we have based navigation for manipulation on the concept of place rather than position. A place was realized as the set of positions from where the object can be successfully be grasped weighted with the probability that the respective pick up will be successful. The advantage of using places instead of positions are that places provide more flexibility, that places can take the inaccuracies of position estimates into account, and that places for multiple manipulation tasks can be combined to compute places where all tasks can be performed successfully. In addition, places are learned from experience in performing everyday manipulation tasks. The second method that increases generality is our method for object recognition and localization. This method can recognize objects of particular classes using 3D object models retrieved from the world-wide web. In order to optimize object perception and localization for stereotypical tasks the robot then uses the specific models that best match the objects in its environment.

In our ongoing research we investigate further aspects of increasing generality. These aspects include the use of partial, inaccurate and uncertain object descriptions and the intergration of probabilistic reasoning. Probabilistic reasoning is necessary to enable the robot to perform tasks such as put a clean, unused cup on the table. To this end, the robot must infer that a cup in the dishwasher before running is probably dirty, a cup standing on the table is probably used by somebody else. Another aspect is to generalize pick and place tasks for more complex situations like get a cup out of the cupboard or put a knife into a drawer. These situations require foresight such as determining the right place considering that the drawer will be opened and the object placed afterwards.

Legibility of the behavior of our robot is increased through place-based mobile manipulation as well as the learning of reaching strategies. Because the robot learns the places that are appropriate for performing the respective actions, going to the respective places already indicates the manipulation actions to be performed. Likewise learning the reaching strategies of humans and mapping them into robot control enables people to infer what the robot is doing through perspective taking.

Besides further scaling the generality of our control programs we also investigate the integration of the manipulation capabilities described in this article into more comprehensive control systems that are capable of performing complete activities such as setting the table or cleaning up. These aspects are described in a companion paper [**?**].

References

1. T. Asfour, K. Regenstein, P. Azad, J. Schroder, A. Bierbaum, N. Vahrenkamp, and R. Dillmann. Armar-iii: An integrated humanoid platform for sensory-motor control. In *Proceedings of the IEEERAS/RSJ International Conference on Humanoid Robots (Humanoids06)*, pages 169–175, 2006.
2. Michael Beetz, Freek Stulp, Bernd Radig, Jan Bandouch, Nico Blodow, Mihai Dolha, Andreas Fedrizzi, Dominik Jain, Uli Klank, Ingo Kresse, Alexis Maldonado, Zoltan Marton, Lorenz Mösenlechner, Federico Ruiz, Radu Bogdan Rusu, and Moritz Tenorth. The assistive kitchen — a demonstration scenario for cognitive technical systems. In *IEEE 17th International Symposium on Robot and Human Interactive Communication (RO-MAN), Muenchen, Germany*, 2008. Invited paper.
3. Thorsten Belker. *Plan Projection, Execution, and Learning for Mobile Robot Control*. PhD thesis, Department of Applied Computer Science, University of Bonn, 2004.
4. Dmitry Berenson, Rosen Diankov, Koichi Nishiwaki, Satoshi Kagami, and James Kuffner. Grasp planning in complex scenes. In *IEEE-RAS International Conference on Humanoid Robots*, 2007.
5. A. Billard, S. Calinon, R. Dillmann, and S. Schaal. *Springer Handbook of Robotics*, chapter 59. Robot programming by demonstration. Springer, 2008.
6. S. Calinon, F. Guenter, and A. Billard. On learning, representing and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man and Cybernetics, Special issue on robot learning by observation, demonstration and imitation*, 37(2):286–298, 2007.
7. A. Dearden and Y. Demiris. Learning forward models for robotics. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1440–1445, 2005.
8. T. Flash and B. Hochner. Motor primitives in vertebrates and invertebrates. *Current Opinion in Neurobiology*, 15:660–666, 2005.
9. M. Fox, J. Gough, and D. Long. Using learned action models in execution monitoring. In *Proceedings of UK Planning and Scheduling SIG*, 2006.
10. Brian Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage Project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*, pages 317–323, 2003.
11. Karen Zita Haigh. *Situation-Dependent Learning for Interleaved Planning and Robot Execution*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1998.
12. C. M. Harris and Daniel M. Wolpert. Signal-dependent noise determines motor planning. *Nature*, 394(20):780–784, 1998.

13. Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamic movement primitives for movement generation motivated by convergent force fields in frog. In Roy Ritzmann and Robert Quinn, editors, *Fourth International Symposium on Adaptive Motion of Animals and Machines*, Case Western Reserve University, Cleveland, OH, 2008.
14. Andreas Hofhauser, Carsten Steger, and Nassir Navab. Harmonic deformation model for edge based template matching. In *Third International Conference on Computer Vision Theory and Applications*, volume 2, pages 75–82, 2008.
15. I. Horswill. Integrated systems and naturalistic tasks. In: Strategic Directions in Computing Research AI Working Group, 1996.
16. A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *International Conference on Robotics and Automation (ICRA2002)*, 2002.
17. O. Jenkins, R. Bodenheimer, and R. Peters. Manipulation manifolds: Explorations into uncovering manifolds in sensory-motor spaces. In *International Conference on Development and Learning (ICDL 2006)*, 2006.
18. Simon Julier and Jeffrey K. Uhlmann. A general method for approximating nonlinear transformations of probability distributions. Technical report, Dept. of Engineering Science, University of Oxford, 1996.
19. D. Katz, E. Horrell, Y. Yang, B. Burns, T. Buckley, A. Grishkan, V. Zhylkovskyy, O. Brock, and E. Miller. The UMass Mobile Manipulator UMan: An Experimental Platform for Autonomous Mobile Manipulation. In *Workshop on Manipulation in Human Environments at Robotics: Science and Systems*, 2006.
20. C. Kemp, A. Edsinger, and E. Torres-Jara. Challenges for robot manipulation in human environments. *IEEE Robotics and Automation Magazine*, 14(1):20–29, 2007.
21. D. Kragic, M. Björkman, H.I. Christensen, and J.O. Eklundh. Vision for robotic object manipulation in domestic settings. *Robotics and Autonomous Systems*, 52(1):85–100, 2005.
22. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.
23. V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(9):1465–1479, Sept. 2006.
24. A. A. Maciejewski and C. A. Klein. The singular value decomposition: Computation and applications to robotics. *International Journal of Robotics Research*, 8(6):63–79, 1989.
25. D. McDermott. A Reactive Plan Language. Research Report YALEU/DCS/RR-864, Yale University, 1991.
26. G. Metta, P. Fitzpatrick, and L. Natale. YARP: Yet Another Robot Platform. *International Journal of Advanced Robotics Systems, special issue on Software Development and Integration in Robotics*, 3(1), 2006.
27. Daniel Nyga, Moritz Tenorth, and Michael Beetz. Understanding and executing instructions for everyday manipulation tasks from the world wide web. Submitted to the IEEE International Conference on Robotics and Automation (ICRA). Under review, 2009.
28. R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Shape distributions. *ACM Transactions on Graphics (TOG)*, 21(4):807–832, 2002.
29. E. Oztop, D.W. Franklin, T. Chaminade, and G. Cheng. Human-humanoid interaction: Is a humanoid robot perceived as a human? *International Journal of Humanoid Robotics*, 2(4):537–559, 2005.
30. Dae-Hyung Park, Heiko Hoffmann, and Stefan Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *International Conference on Humanoid Robots*, 2008.
31. J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21:682–697, 2008.
32. Anya Petrovskaya and Andrew Y. Ng. Probabilistic mobile manipulation in dynamic environments, with application to opening doors. In *International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007.
33. Pierre Roduit, Alcherio Martinoli, and Jacques Jacot. A quantitative method for comparing trajectories of mobile robots using point distribution models. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2441–2448, 2007.
34. David A. Rosenbaum, Rajal G. Cohen, Ruud G. J. Meulenbroek, and Jonathan Vaughan. Plans for grasping objects. In Mark L. Latash and Francis Lestienne, editors, *Motor Control and Learning*, pages 9–25. Springer US, 2006.
35. Sam Roweis and Lawrence Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
36. Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3D Point Cloud Based Object Maps for Household Environments. *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge)*, 2008.
37. A. Saxena, J. Driemeyer, and A.Y. Ng. Robotic Grasping of Novel Objects using Vision. *The International Journal of Robotics Research*, 27(2):157, 2008.
38. Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.
39. Ruben Smits, Tinne De Laet, Kasper Claes, Peter Soetens, Joris De Schutter, and Herman Bruyninckx. Orocos: A software framework for complex sensor-driven robot tasks. *IEEE Robotics and Automation Magazine*, 2008.
40. Siddhartha Srinivasa, David Ferguson, Michael Vande Weghe, Rosen Diankov, Dmitry Berenson, Casey Helfrich, and Hauke Strasdat. The robotic busboy: Steps towards developing a mobile robotic home assistant. In *International Conference on Intelligent Autonomous Systems*, 2008.
41. Freek Stulp and Michael Beetz. Refining the execution of abstract actions with learned action models. *Journal of Artificial Intelligence Research (JAIR)*, 32, June 2008.
42. Freek Stulp, Wolfram Koska, Alexis Maldonado, and Michael Beetz. Seamless execution of action sequences. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3687–3692, 2007.
43. Freek Stulp, Ingo Kresse, Alexis Maldonado, Federico Ruiz, and Michael Beetz. Compact models of human

reaching motions for robotic control in everyday manipulation tasks. Submitted to the IEEE International Conference on Robotics and Automation (ICRA). Under review., 2009.
44. R. Sutton and A. Barto. *Reinforcement Learning: an Introduction*. MIT Press, 1998.
45. Emanuel Todorov. Optimality principles in sensorimotor control. *Nature Neuroscience*, 7(9):907–915, 2004.
46. Christian Wiedemann, Markus Ulrich, and Carsten Steger. Recognition and tracking of 3d objects. In Gerhard Rigoll, editor, *Pattern Recognition*, volume 5096 of *Lecture Notes in Computer Science*, pages 132–141, Berlin, 2008. Springer-Verlag.
47. A. Wilhelm, J. Huissoon, W. Melek, C. Clark, M. Fuchs, and G. Hirzinger. Design of a wheeled mobile robotic platform with variable footprint. In *IEEE/RSJ Conference on Intelligent Robotics and Systems*, 2007.
48. PR2 Robot, 2008. `www.willowgarage.com/pages/robots/pr2-overview`.
49. Matthias Wimmer, Freek Stulp, Sylvia Pietzsch, and Bernd Radig. Learning local objective functions for robust face model fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(8):1357–1370, 2008.
50. Daniel Wolpert and Zoubin Ghahramani. Computational principles of movement neuroscience. *Nature Neuroscience Supplement*, 3:1212–1217, 2000.
51. F. Zacharis, Ch. Borst, and G. Hirzinger. Capturing robot workspace structure: representing robot capabilities. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3229–3236, 2007.
52. F. Zacharis, Ch. Borst, and G. Hirzinger. Positioning mobile manipulators to perform constrained linear trajectories. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2578–2584, 2008.