The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Action-Related Place-Based Mobile Manipulation

Freek Stulp, Andreas Fedrizzi, Michael Beetz
Intelligent Autonomous Systems Group, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
{stulp,fedrizzi,beetz}@cs.tum.edu

***Abstract*— In mobile manipulation, the position to which the robot navigates has a large influence on the ease with which a subsequent manipulation action can be performed. Whether a manipulation action succeeds depends on many factors, such as the robot's hardware configuration, the controllers the robot uses to achieve navigation and manipulation, the task context, and uncertainties in state estimation. In this paper, we present 'ARPLACE', an action-related place which takes these factors, and the context in which the actions are performed into account. Through experience-based learning, the robot first learns a so-called *generalized success model*, which discerns between positions from which manipulation succeeds or fails. On-line, this model is used to compute a ARPLACE, a probability distribution that maps positions to a predicted probability of successful manipulation, and takes the uncertainty in the robot and object's position into account. In an empirical evaluation, we demonstrate that using ARPLACEs for least-commitment navigation improves the success rate of subsequent manipulation tasks substantially.**

I. INTRODUCTION

A key aspect of mobile manipulation research is that navigation and manipulation are not considered in isolation, but that the focus is on developing methods to navigate *in order* to manipulate. The close coupling between navigation and manipulation becomes apparent in Fig. 1, where the robot's task is to approach the table in order to grasp a cup. A trivial approach to solving this task is simply going to a position such that the target object 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? Does well-in-reach always imply that the target object can really be reached, given the hardware and control software of the robot? How can such a concept of 'place' take into account uncertainties about the robot's self-localization and estimated target object position?

We address these questions by developing an action-related place, denoted ARPLACE, that takes into account the manipulation and navigation skills of a robot, as well as its hardware configuration. ARPLACE is represented as a probability distribution, that maps (estimations of) the target object's and robot's position to a probability that the target object will be successfully grasped from that position.

Fig. 2 visualizes an ARPLACE for a given target object position. ARPLACE implements a least commitment realization of positions. This means that the robot does not commit itself to a specific goal position, but can refine it as the robot learns more about the task context, e.g. better estimations of the target object's pose, observed clutter in the environment, etc. ARPLACE is very flexible representation of place and can also be used to find good positions for manipulating multiple objects, to optimize the place for executing a sequence of manipulation actions, or to optimize secondary constraints such as execution duration.

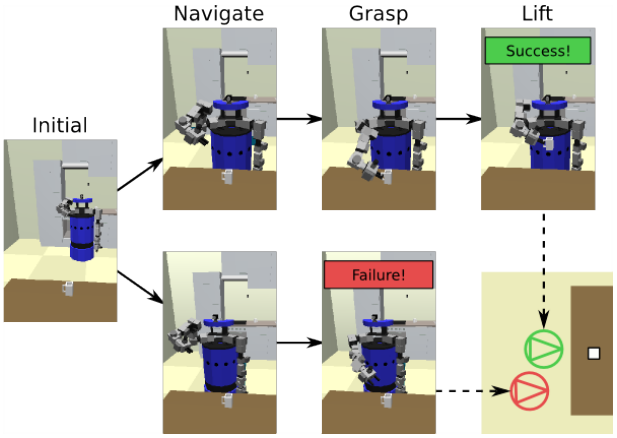


Fig. 1. Mobile manipulation task considered in this paper. Example of a successful and a failed attempt. This figure is explained in more detail in Section IV.

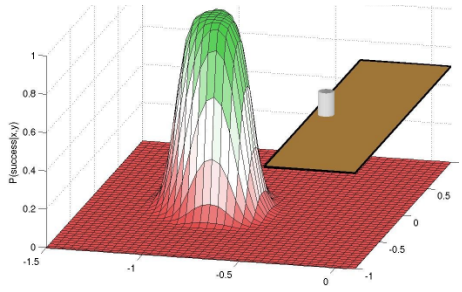


Fig. 2. Probability distribution of successful manipulation, given a pose estimation of the cup. This is the robot's ARPLACE for this task.

By actively considering a multitude of appropriate manipulation positions in a least-commitment way, ARPLACE forestalls and avoids positions from which manipulation is difficult. Our empirical evaluation demonstrates that this leads to more robust mobile manipulation.

Manually designing an explicit model of ARPLACE that takes all the questions in the first paragraph into account is tedious and error-prone due to the large state space. We therefore believe that the robot should acquire ARPLACE 1) autonomously, through learning from interactions with the environment 2) with respect to its own capabilities, which are limited by the hardware and control programs.

As depicted in the computational model of our approach in Fig. 3, the robot learns ARPLACEs for a task by first

gathering experience in simulation. By using Support Vector Machines, we then acquire *classification boundaries*. A classification boundary models regions from which a certain manipulation task was successfully executed. As a final step of model creation we compile the multiple classification boundaries into one *generalized success models* (GSM). This is done by using a Point Distribution Model (PDM). During online task execution, the robot queries the GSM to determine positions that are appropriate for starting the manipulation task.

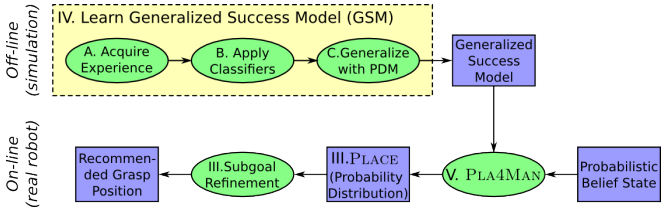


Fig. 3. Computational model (Numbers refer to sections in the paper).

The rest of this paper is structured as follows. In the next section, we discuss related work. We then describe the rationale behind ARPLACE in Section III. How the GSM is learned is explained in Section IV. Section V shows how the GSM is used to compute an ARPLACE. In Section VI we present an empirical evaluation of the system, and we conclude with Section VII.

II. RELATED WORK

Berenson et al. [3] deal with the problem of finding optimal start and goal configurations for manipulating objects in pick-and-place operations. They explicitly take the placement of the mobile base into account. But as they are interested in the optimal start and goal configurations, they do not have a probabilistic representation of the whole space.

The capability map is another option to model robot configurations that lead to successful grasping [16]. Capability maps can be used to find regions where the dexterity of a manipulator is high. As they only consider the kinematics of a robot, they are not optimized for a given skill repertoire or environment; in short, they are not action-related. Although there are methods that use the capability map for computing arm motions to reach a certain pose [16], these methods do not cope with the problem of uncertain pose estimations.

Learning success models is a form of pre-condition learning. In robotics, the focus in pre-condition learning is on grounding pre-conditions in robot experience. For instance, 'Dexter' learns sequences of manipulation skills such as searching and then grasping an object [9]. Declarative knowledge such as the length of its arm is learned from experience. Learning success models has also been done in the context of robotic soccer, for instance learning the success rate of passing [5], or approaching the ball [14]. Our methods extend these approaches by explicitly representing the region in which successful instances were observed, and computing generalized success models from these regions.

Friedman and Weld demonstrated the advantages of least commitment planning in [7]. They showed that setting open conditions to abstract actions and later refining this choice to a particular concrete action can lead to exponential savings.

III. ARPLACE: A PROBABILISTIC LEAST-COMMITMENT REPRESENTATION OF PLACE

We propose ARPLACE as a powerful and flexible representation of the utility of positions in the context of action-related mobile manipulation. Instead of committing to a specific position in advance, an ARPLACE enables least-commitment planning, as a whole range of positions are predicted to be successful, or at least probable. The robot will start to move to a position that is good enough to execute the subsequent manipulation action and will refine the goal position while it moves. In the context of our scenario of grasping a cup from a table, this would mean that ARPLACE finds a solution area that is good enough for the robot to start moving. For instance, the robot could choose any of the positions for which $P(succ|\mathbf{t}) > 0.95 * max(P(succ|\mathbf{t}))$. As the robot approaches the table, new sensor data comes in, and the robot's state estimate is updated (i.e. accuracy of the cup position, information on clutteredness of regions, etc.). As a consequence the ARPLACE is updated, and becomes more and more precise.

The principle of least commitment is especially powerful in real environments, where complete information, required to compute optimal goal positions, is not available. Even if the environment is completely observable, dynamic properties could make an optimal pre-planned position suboptimal or unaccessible. A least-commitment implementation can delay decisions as long as possible, and therefore is more flexible while reducing the need for replanning.

ARPLACE is implemented as a continuous probability distribution that represents the probability of successfully grasping the target object when standing at a certain position. A position p is a 3-tuple given by $p := \{(x, y, a) \mid x \in \mathbf{R},\ y \in \mathbf{R}, a \in]-2\pi, .., 2\pi]\}$. Given an estimation of the target object's pose p_o, the covariance matrix of the pose estimation $C(p_o)$, and the covariance matrix of the robot's localization estimation $C(p_r)$, ARPLACE assigns a probability value to all positions p, so that $P(p) = f(p_o, C(p_o), C(p_r))$. How a representation of an ARPLACE looks like can be seen in Fig. 2. A video that shows how changes in $C(p_o)$ and $C(p_r)$ affects the ARPLACE is submitted as supplementary material to this paper.

The most probable position is not always the one with the highest utility. ARPLACE can be easily transfered to a utility-based representation by providing heuristics that consider arbitrary secondary constraints like power consumption, time, end-effector-movement, or torque-change. Selecting subgoal parameters such that they optimize secondary criteria is known as *subgoal refinement* [14].

Finally, ARPLACEs for multiple actions can be composed by intersecting them. Assume we have computed ARPLACEs for two different actions (a_1 and a_2). If the success probabilities of the ARPLACEs is independent, we can compute the ARPLACE for executing both actions in parallel by multiplying the probabilities of the ARPLACEs of a_1 and a_2. Fig. 4 illustrates this for the task of concurrently grasping

two cups. This composition would be impossible if the robot commits itself to specific positions in advance.

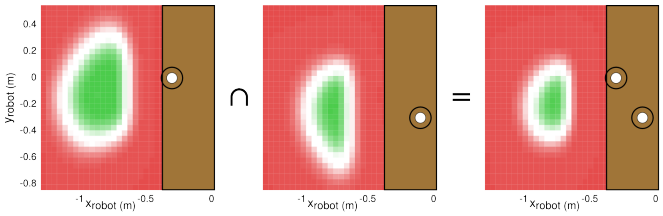


Fig. 4. Left distribution: grasp cup with left gripper. Center distribution: grasp cup with right gripper. Right distribution: Grab both cups with left/right gripper respectively. It is the product of the other two distributions.

IV. Learning a Generalized Success Model for ARPLace

In this section, we describe the implementation of the off-line learning of the GSM, as depicted in the upper row of the computational model in Fig. 3. The rest of this section is structured according to Algorithm 1, which is explained throughout this section.

```
   input  : T ;                    (task relevant parameters (cup positions))
1            #episodes ;           (#experiments per parameter setting)
   output : gsm ;                                (generalized success model)
2  forall cup_xy in T do
3      experience_set.clear( );
4      for i=1:#episodes do
5          robot_xy = randompos(cup_xy);
6          success? = executescenario(robot_xy, cup_xy);
7          experience_set.add( ⟨cup_xy, robot_xy, success?⟩ );
8      end
9      boundary = classify(experience_set) ;          (With SVM)
10     boundary_set.add( ⟨cup_xy, boundary⟩ );
11 end
12 H = alignpoints(boundary_set);
13 ⟨H̄, P, B⟩ = computePDM(H);
14 W = [1 T]/B^T ;     (Mapping from task relevant parameters to B)
15 gsm = ⟨H̄, P, W⟩
```

Algorithm 1: Computing a Generalized Success Model.

A. Acquiring Training Data (Line 2-6)

The robot first gathers training data by repeatedly executing a navigate-reach-grasp action sequence. In Fig. 1, two experiment runs with different samples for the robot position are depicted. 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. To acquire sufficient data in little time, we perform the training experiments in the Gazebo simulator. This allows us to perform approximately 70 training experiments per hour. The model that is used and evaluated in this paper was learned from the experience gathered in 4200 training experiments. So the offline computation time was 60 hours, or 2.5 days. Note that most of the time is spent on restarting software modules for each episode. We are currently optimizing the communication between these modules so they need not be restarted, and we expect this to reduce the off-line time by more than 50%. The robot is modeled accurately, and thus the simulator provides training data that is also valid for the real robot. The action sequence is executed for a variety of task-relevant parameters. In our scenario the robot grasps a cup at the handle with its right arm. The task-relevant parameters were the x, y position of the cup on a table. The 12 cup positions on the table with which the robot is trained are depicted in Fig. 5. 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 log-file. Please note that the logged training data does not contain noise, because the simulator allows us to precisely know the initial cup position and the sampled initial position of the robot. Ground truth data from simulation thus allows us to learn an accurate model of the task. Of course, uncertainties arise on-line due to the robot's perception. How this is dealt with is described in Section V.

B. Computing Classification Boundaries (Line 9-10)

To acquire success models, we compute a classification boundary around the successful samples using Support Vector Machines (SVM), using the implementation by [13]. We used SVM with a Gaussian kernel σ=0.03 and cost parameter C=20.0. The latter controls the trade off between allowing training errors and forcing rigid margins. Fig. 5 depicts the resulting classification boundaries for different configurations of task-relevant parameters. To us, the data and the clusters are shifted a bit more to the right (from the robot's point of view) than we would have expected when grasping the cup with the right arm. This is due to the hardware and kinematics of the robot, which are not very human-like. This supports our approach of experience-based learning compared to hand-coding, as our intuitions about a good 'place' for robot manipulation apparently do not always correspond to the 'place' that is really the best for a particular robot.

The models on average classify 5% of examples wrongly when using a training/test set that contain 66%/33% of the data respectively, and 3% when using the training data as the test data.

C. Computing the Point Distribution Model (Line 12-13)

As input a PDM requires n points that are distributed over the contour. We distribute 20 points equidistantly over each boundary, and determine the correspondence between points on different boundaries by minimizing the sum of the distances between corresponding points, while maintaining order between the points on the boundary.

Given the aligned points on the boundaries, we compute a PDM. Although PDMs are most well-known for their use in computer vision, we use the notation by Roduit et al. [12], who focus on robotic applications. First, the 2D boundaries 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 classification boundary. Each row represents one boundary. 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 boundary $\mathbf{h}_k$ in the set into the mean boundary and a linear combination of the columns of

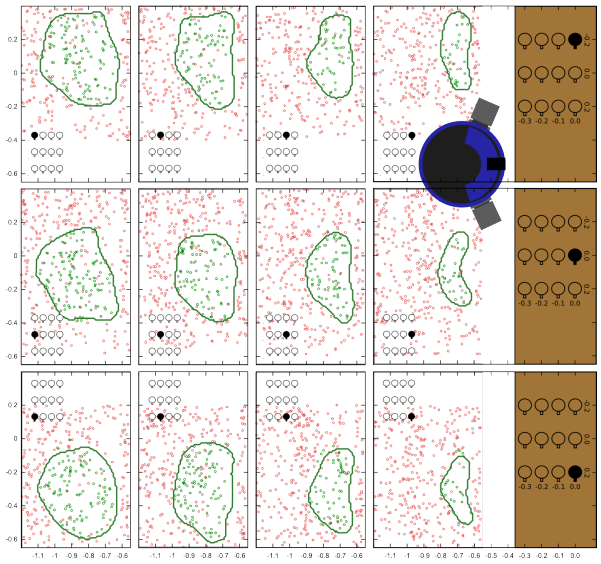
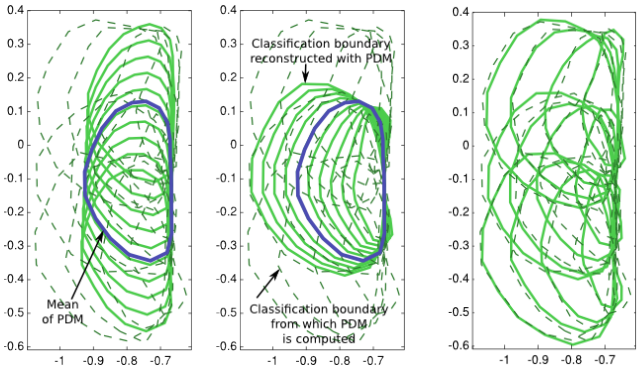


Fig. 5. Successful grasp positions and their classification boundaries. Every sub-image shows the boundary that corresponds to the cup position that is visualized with the black cup. The cup positions on the table are drawn to scale, as is the B21 robot. To save space, the table on which the cup is placed is only shown in the right-most graphs, and not all failed data points are drawn. Data points correspond to the center of the robot base. In this figure and the following ones, the horizontal axis always represents the x coordinate, and the vertical axis the y coordinate of the robot, both in meters.

$\mathbf{P}$ 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} boundary. This is the Point Distribution Model. To get an intuition for what the PDM represents, the first two deformation modes are depicted in Fig. 6(a), where the values of the first and second column of $\mathbf{B}$ are varied between their maximal and minimal value.



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

(b) Reconstructing the boundaries from Fig. 5.

Fig. 6. A generalized success model based on 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 deformation energy. Therefore, we use only the first 2 deformation modes, without losing much accuracy. Fig. 6(b) demonstrates that the original 12 boundaries can be reconstructed well when using combinations of 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 boundaries. It also allows us to interpolate between them in a principled way using only two deformation parameters. The PDM is therefore a compact, general, yet accurate model for the classification boundaries.

D. Relation to Task-relevant Parameters (Line 14)

The final step of model learning is to relate the specific deformation of each boundary, contained in $\mathbf{B}$, to the values of the task-relevant parameters $\mathbf{T}$, which in this case are all 12 combinations of the x and y coordinates of cup position, which are {-0.3 -0.2 -0.1 -0.0} and {-0.2 0.0 0.2} respectively. Since the correlation coefficients between the first and second deformation modes $\mathbf{B}$ and the task relevant parameters $\mathbf{T}$ 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, the GSM allows to quickly compute the area from which a successful grasp can be expected for this *specific* situation. First, we compute the appropriate deformation values from the cup position with $\mathbf{b}_{new} = ([1\ \mathbf{t}_{new}] \cdot \mathbf{W})^T$. Then the boundary is computed with $\mathbf{h}_{new} = \overline{\mathbf{H}} + \mathbf{P} \cdot \mathbf{b}_{new}$. This boundary estimates the area in which the robot should stand to be able to make a successful grasp.

V. Computing ARPlaces On-line

In this section, we describe how appropriate ARPlaces for manipulation are determined on-line. We call this module 'planning for manipulation' (PLA4MAN). As can be seen in the computational model in Fig. 3, this module takes the GSM and the probabilistic belief state as input, and returns an ARPLACE such as depicted in Fig. 2 or Fig. 4. The rest of this section is structured according to Algorithm 2, which is explained throughout this section.

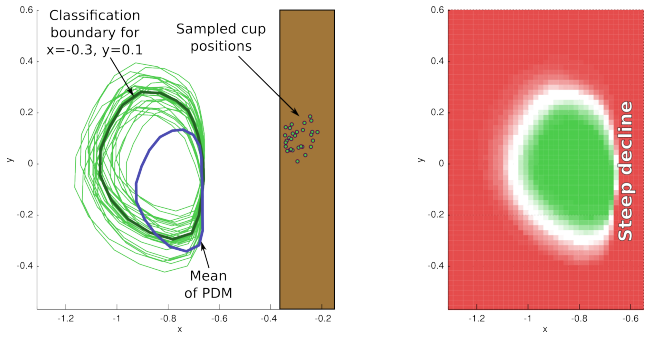
```
   input  : gsm = ⟨H̄, P, W⟩ ;          (generalized success model)
1            objectposition ;          (probability distribution, estimated)
2            robotposition ;           (probability distribution, estimated)
   output : arplace ;                   (probability map)
3  for i=1 to #samples do
4      t_s = samplefromdistribution(objectposition);
5      b_s = ([1 t_s] · W)^T;
6      classif_boundary_set.add( H̄ + P · b_s );
7  end
8  arplace = Σ_{i=1}^{#samples} grid(boundary_set_i) / #samples;
9  arplace = arplace * robotposition ;   (Convolution)
```

Algorithm 2: Computing ARPLACE.

A. Uncertainty in Object Position (Line 3-8)

At the end of the previous section, we demonstrated how a $\mathbf{h}_{new}$ classification boundary is reconstructed, given specific task relevant parameters $\mathbf{t}_{new} = \langle x_{new},\ y_{new} \rangle$. Due to sensor noise and other factors that influence the state estimation, the task relevant parameters can never be known exactly, and uncertainty must be modeled. The belief state therefore also associates a covariance matrix with each position: $\begin{pmatrix} \sigma_{xx}^2 & \sigma_{yx}^2 \\ \sigma_{xy}^2 & \sigma_{yy}^2 \end{pmatrix}$, computed by our vision-based object localization module [11].

Because of this uncertainty, it does not suffice to compute only one classification boundary given the most probable position of the cup as the ARPLACE from which to grasp. This might lead to a failure if the cup is not at the position where it was expected. To solve this problem, 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, given its mean and covariance matrix. This yields a matrix of task relevant parameters $\mathbf{t}_s = [\mathbf{x}_s\ \mathbf{y}_s]$. The corresponding classification boundaries $\mathbf{h}_s$ are computed for the samples by using the method described above. In Fig. 7(a), 30 out of the 100 boundaries are depicted. These were generated from the task relevant parameters x=-0.3, y=0.1, σ_{xx}=σ_{yy}=0.05, σ_{xy}=σ_{yx}=0.



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

(b) Discretized relative sum of the boundaries.

Fig. 7. Monte-Carlo simulation of classification boundaries to compute ARPLACE.

We then generate a discrete grid in which each cell measures $2.5 \times 2.5cm$, and compute for each cell the number of classification boundaries that classify this cell as a success. Dividing the number of successful classification boundaries by the overall number of boundaries yields an approximation of the probability that grasping the cup will succeed from this cell. The corresponding probability map, which takes the uncertainty of the cup position into account, is depicted in Fig. 7(b) (2D), as well as in Fig. 2 (3D).

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 boundaries contain this area, and the variation in $\mathbf{P}$ on the right side of the PDM is low. Variations in $\mathbf{B}$ do not have a large effect on this boundary, as can be seen in Fig. 7(b). When summing over the sampled boundaries, this leads to a steep decline in success probability in the direction of the table.

B. Uncertainty in Robot Position (Line 9)

The Adaptive Monte Carlo Localization from the Player project [8] also returns a covariance matrix for the robot's position. This uncertainty must be taken into account in ARPLACE. For instance, although any position near to the left of the steep incline in Fig. 7(b) is predicted to be successful, they might still fail if the robot is actually more to the right than expected. Therefore, we convolve the ARPLACE as depicted in Fig. 7(b) with the discretized $(2.5 \times 2.5cm)$ probability distribution of the robot's position. The result can be seen in Fig. 8. Note that this convolution also works for multi-modal distributions as returned by particle filters.

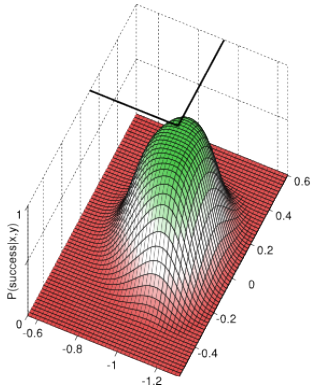


Fig. 8. Final distribution, after convoluting the uncertainty in the robot pose with a distribution as depicted in Fig. 7(b). This distribution represent the robot's probabilistic least-commitment ARPLACE, which is task-related, skill-specific, and grounded in experience. This is a distribution for parameters: x=-0.3, y=0.1, σ_{xx}=σ_{yy}=0.05.

The distribution in Fig. 8 is the robot's ARPLACE for this task, and takes into account the uncertainty in both the pose of the robot and the target object. These distributions are generated from a model that is very much grounded in observed experience, as it was learned from observation. Note that this ARPLACE is also specific for the task context and the skills of the robot. Using a different robot or controller would lead to different observations, and hence to a different ARPLACE. It is the developmental process of learning ARPLACE that allows us to apply it to a wide range of robots and controllers.

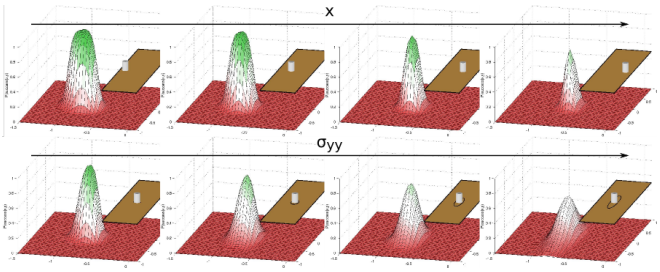


Fig. 9. These images show how varying certain task-relevant parameters affects the shape of the distribution. The table and the cup are drawn to scale in the xy-plane.

Fig. 9 depicts how the probability distribution is affected by varying task-relevant parameters. Row 1 demonstrates how it becomes 'more difficult' (i.e. less likely to succeed) to grasp the cup as the it moves away from the edge.

VI. EMPIRICAL EVALUATION

At a day of open house, our B21 mobile manipulation platform continually performed an application scenario, where it

locates, grasps, and lifts a cup from the table and moves it to the kitchen oven. The robot performed this scenario 50 times in approximately 6 hours, which has convinced us that the robot hardware and software are robust enough to be deployed amongst the general public. After the open day, we ran the same experiment, but this time with the PLA4MAN module included. The focus of this experiment was on our error-recovery system described in [1]. That is why the improved robot performance cannot quantitatively be attributed to the PLA4MAN module or the error-recovery system. However, a major qualitative improvement we certainly can attribute to the PLA4MAN module was that the cup can now be grasped from a much larger area on the table.

An empirical evaluation was done in the Gazebo simulator. We compared the PLA4MAN module to another strategy which we call FIXED. FIXED is implemented by always moving to a location that has the same relative offset to the target object. The relative location was chosen to be the offset with the best possible overall performance. The cup was placed in three different locations. In one simulated experimental episode, we first determine the observed position of the cup, by sampling from the distribution given by the real position p_o of the cup and the covariance matrix $C(p_o)$. The latter two are the controlled variables for each experiment. Given the estimated cup position, the robot then uses the PLA4MAN or well-in-reach module to compute an ARPLACE and performs the manipulation action. When the robot is able to perform the manipulation task after moving to the proposed position we mark the experiment as SUCCESS. Otherwise we mark the experiment as FAILED.

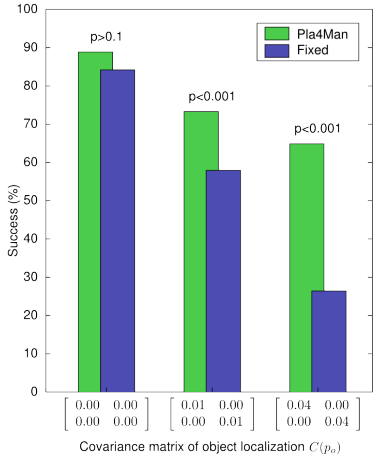


Fig. 10. Result of the empirical evaluation. The x-axis shows experiments with different values for $C(p_o)$. The y-axis shows the success rate.

Fig. 10 shows the results of the evaluation. Naturally, the performance of both methods decreases, as the robot becomes more and more uncertain about the pose of the cup. One result is that PLA4MAN always performs better than FIXED. Another important result is that while the uncertainty rises, the performance of FIXED suffers more than the performance of PLA4MAN.

VII. CONCLUSION

In this paper, we have presented a system that enables robots to learn an action-related concept of place, called 'ARPLACE', that is compact, grounded in observed experience, and tailored to the robot's hardware and controller. ARPLACE is modelled as a probability distribution, which enables the robot to perform least-commitment planning, instead of prematurely committing itself to specific positions that could be suboptimal. Optimizing the probability of successful grasping resulted in more robust behavior on our mobile manipulation platform.

We are currently extending our approach in several directions. We are applying our approach to more complex scenarios, and different domains. For instance, we are learning higher-dimensional ARPLACEs. New aspects that we are taking into account are different kinds of objects which require different kinds of grasps, two-handed manipulation, and using secondary constraints to model more complex utility functions. We are also investigating extensions and other machine learning algorithms that will enable our methods to generalize over larger spaces.

ACKNOWLEDGEMENTS

This work was supported by the DFG project ActAR (Action Awareness in Autonomous Robots) and the cluster of excellence COTESYS (Cognition for Technical Systems).

REFERENCES

[1] M. Beetz, F. Stulp, et al. Generality and legibility in mobile manipulation. *Autonomous Robots Journal, Special Issue on Mobile Manipulation (submitted for review)*, 2009.

[2] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner. Grasp planning in complex scenes. In *IEEE-RAS International Conference on Humanoid Robots*, 2007.

[3] D. Berenson, H. Choset, and J. Kuffner. An Optimization Approach to Planning for Mobile Manipulation. In *Proc. of the IEEE International Conference on Robotics and Automation*, 2008.

[4] R. Bohlin, and L. E. Kavraki. Path Planning using lazy PRM. In *IEEE International Conference on Robotics and Automation*, 2000.

[5] S. Buck and M. Riedmiller, Learning situation dependent success rates of actions in a RoboCup scenario. In *Pacific Rim International Conference on Artificial Intelligence*, 2000.

[6] B. Clement, E. Durfee, and A. Barrett. Abstract reasoning for planning and coordination. *J. of Artificial Intelligence Research*, 28, 2007.

[7] M. Friedman, and D. S. Weld. Least commitment action-selection. In *Proc. of the International Conf. on AI Planning Systems*, 3, 1996.

[8] B. Gerkey, R. T. Vaughan, and A. Howard. The Player/Stage Project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, 2003.

[9] S. Hart, S. Ou, J. Sweeney, and R. Grupen. A framework for learning declarative structure. In *RSS-06 Workshop: Manipulation for Human Environments*, 2006.

[10] 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.

[11] U. Klank, M. Z. Zia, and M. Beetz. 3D Model Selection from an Internet Database for Robotic Vision. In *International Conference on Robotics and Automation (ICRA)*, 2009.

[12] P. Roduit, A. Martinoli, and J. Jacot. A quantitative method for comparing trajectories of mobile robots using point distribution models. In *Proceedings of IROS*, pages 2441–2448, 2007.

[13] S. Sonnenburg, G. Raetsch, C. Schaefer, and B. Schoelkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, 2006.

[14] F. Stulp and M. Beetz. Refining the execution of abstract actions with learned action models. *JAIR*, 32, June 2008.

[15] M. Wimmer, F. Stulp, S. Pietzsch, and B. Radig. Learning local objective functions for robust face model fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(8):1357–1370, 2008.

[16] F. Zacharias, Ch. Borst, and G. Hirzinger. Positioning Mobile Manipulators to Perform Constrained Linear Trajectories. In *Proc. of the IEEE/RSJ International Conf. on Intelligent Robots and Systems*, 2008.