Transformational Planning for Mobile Manipulation based on Action-related Places

Andreas Fedrizzi, Lorenz Mösenlechner, Freek Stulp, Michael Beetz
Intelligent Autonomous Systems, Technische Universität München
{fedrizza|moesenle|stulp|beetz}@cs.tum.edu

***Abstract*— Opportunities for interleaving or parallelizing actions are abundant in everyday activities. Being able to perceive, predict and exploit such opportunities leads to more efficient and robust behavior. In this paper, we present a mobile manipulation platform that exploits such opportunities to optimize its behavior, e.g. grasping two objects from one location simultaneously, rather than navigating to two different locations. To do so, it uses a general least-commitment representation of place, called ARPLACE, from which manipulation is predicted to be successful. Models for ARPLACEs are learned from experience using Support Vector Machines and Point Distribution Models, and take into account the robot's morphology and skill repertoire. We present a transformational planner that reasons about ARPLACEs, and applies transformation rules to its plans if more robust and efficient behavior is predicted.**

I. INTRODUCTION

In everyday activities, opportunities for optimizing the course of action arise constantly, as tasks can often be interleaved or executed in parallel. For instance, when setting the table, plates can be stacked instead of carrying them one at a time, cupboards can be left open during the task, etc. Being able to perceive, predict and exploit such opportunities leads to more efficient and robust behavior.

To enable this approach, the robot must: 1) use least-commitment planning, i.e. not prematurely commit to a specific plan when it is not necessary; 2) have rules for transforming suboptimal plans into more efficient ones. In this paper, we apply plan transformation rules to a mobile manipulation task, in which a robot approaches a table and grasps one or more cups, as depicted in Fig. 1.

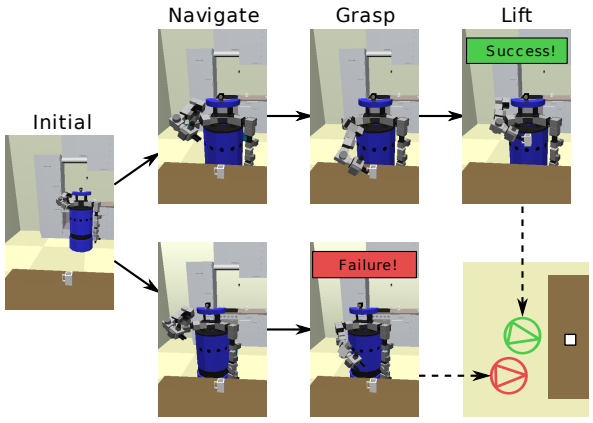


Fig. 1. Mobile manipulation task considered in this paper. Example of a successful and a failed attempt. This figure is explained in more detail in Section IV.

To enable least-commitment planning when navigating *in order* to grasp, we propose a concept of action-related place, denoted ARPLACE, that takes into account the manipulation and navigation skills of a robot, as well as its hardware configuration. The ARPLACE is represented as a probability map that maps positions of the robot and target objects to a probability that the target objects will be successfully grasped from the robot's position. Fig. 2 visualizes ARPLACEs for some given target object positions. The ARPLACE implements a least-commitment realization of positions, meaning that the robot does not commit itself to a specific initial position, but can refine it as the robot learns more about the task context, such as a better estimation of the target object's position, or observed clutter in the environment.

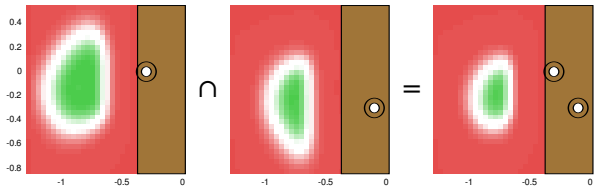


Fig. 2. Left: ARPLACE probability map for grasping cup with left gripper. Center: ARPLACE probability map for right gripper. Right: Grab both cups with left/right gripper respectively. It is the product of the other two probability maps. Green areas mark regions where the probability of a successful grasping action is high.

In this paper, we combine the concept of ARPLACE with a transformational planning system and compose ARPLACEs for multiple actions. The transformational planning system specifies its plans in RPL [12]. RPL is a flexible and powerful language for specifying plans — control programs that cannot only be executed but also reasoned about. Transformational planning performs substantial changes in robot behavior, e.g. by adding plan steps, removing unnecessary ones, or reordering and recombining existing steps. More specific examples are stacking of objects to reduce the number of navigation actions, but also the elimination of more critical failures such as moving objects temporarily out of the way when they are blocking a location the robot needs to navigate to.

We consider a scenario in which the robot has to pick up two cups. By default, this task is solved by navigating to one cup, grasping it, navigating to the second cup, and grasping it. By using the ARPLACE probability map, the transformational planner is able to decide if a position exists where the probability of successfully grasping both cups at

once is sufficiently high. If this is the case, the planner applies a rule for transforming the default plan into the more efficient plan of grasping both cups from one position. This reduces the overall plan length (we save one navigation action), and enables faster task execution.

The main contributions of this paper are: 1) Proposing methods for learning ARPLACE representations from observed experience, using Support Vector Machines and Point Distributions Models 2) Merging different ARPLACEs for independent tasks 3) Integrating ARPLACEs in a transformational planner.

The rest of this paper is structured as follows. In the next section, we discuss related work. In Section III, we describe the concept of ARPLACE. We then explain how a so-called Generalized Success Model (GSM) is learned, and how the GSM is used to compute an ARPLACE with a Monte Carlo simulation in Section IV and V respectively. In Section VI we show how a transformational planning system uses the concept of ARPLACE to optimize plans. Empirical results are presented in Section VII, and we conclude with Section VIII.

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. As they are interested in the optimal start and goal configurations, instead of a probabilistic representation, this approach does not enable least-commitment planning.

The Capability Map is another approach to modelling robot configurations that lead to possible grasps [20]. Capability Maps are used to find regions where the dexterity of a manipulator is high. As they focus on the kinematics of a robot, they are not related to a given skill repertoire or environment. Also, they do not take uncertainties in robot or object position into account.

The robot 'Dexter' learns sequences of manipulation skills such as searching for and then grasping an object [7]. 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 approaching a ball [18]. Our system extends these approaches by explicitly representing the regions in which successful instances are observed, and computing a GSM for these regions.

Friedman and Weld demonstrated the advantages of least-commitment planning in [5]. They showed that setting open conditions to abstract actions and later refining this choice to a particular action can lead to exponential savings. The principle of lazy evaluation is applied to motion planning by Bohlin and Kavraki [4]. They are able to significantly reduce the number of collision checks for building a PRM.

Sussman [19] was the first to realize that *bugs* in plans do not just lead to failure, but are actually an opportunity to construct improved and more robust plans. Although this research was done in the highly abstract symbolic blocks world domain, this idea is still fundamental to transformational planning.

The basis of our transformational planning system is the declarative and expressive plan language RPL, which is described in [2]. The constraints for plan design, especially the specification of declarative goals indicating the purpose of code parts, have been shown in [1]. Besides the modeling of navigation tasks, our system scales with respect to reasoning about perception based on computer vision, the relation between objects and their representation in the robot's belief, as well as reasoning about complex manipulation tasks.

Temporal projection is an integral component of a transformational planning system. McDermott [13] developed a very powerful, totally ordered projection algorithm capable of representing and projecting various kinds of uncertainty, concurrent threads of execution, and exogenous events.

III. Concept of ARPLACE

We propose ARPLACE as a powerful and flexible representation of the utility of positions in the context of action-related mobile manipulation. The concept of ARPLACE is implemented as a continuous probability map that represents the probability of successfully grasping the target object when standing at a certain initial position. Figure 2 depicts three such maps for grasping various cups on a table. Figure 8 shows how variations in the robot's estimation of the cup position influences ARPLACE[1].

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 grasping a cup from a table, this would mean that the concept of ARPLACE finds a solution area that is good enough for the robot to start moving. As the robot approaches the table, new sensor data comes in, and the robot's state estimate is updated (i.e. cup position accuracy, information on clutteredness of regions, etc.). As a consequence the ARPLACE is updated, and becomes 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.

Additionally, the concept of ARPLACE can be easily transfered to a utility-based representation by creating heuristics that optimize for arbitrary secondary constraints such as power consumption, time, end-effector-movement, or torque-change. For further information on the optimization of subgoals with respect to secondary criteria, we refer to [18].

Figure 3 depicts a system overview of how ARPLACE is learned and used for transformational planning. Numerals in

[1]A better impression is given by a video which can be downloaded from `http://www9.cs.tum.edu/people/fedrizzi/icar_09`

the figure refer to sections in this paper. First, a Generalized Success Model (GSM) is learned from observed experience. The transformational planner requests ARPLACEs whenever it finds several manipulation actions in order to find a better location to perform them. These are computed by the PLA4MAN module, which uses the GSM to perform a Monte Carlo simulation. ARPLACEs for individual tasks are merged, to compute ARPLACEs for joint tasks.

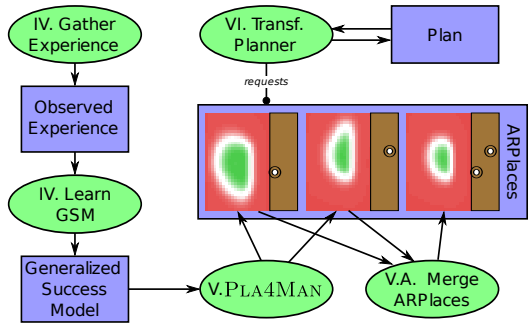


Fig. 3. System overview.

IV. LEARNING A GENERALIZED SUCCESS MODEL

In this section, we describe the implementation of the learning of the GSM, as depicted on the left side in Fig. 3. The rest of this section is structured according to Algorithm 2, 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( ⟨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.

Line 2-6: Acquiring Training Data. The robot first gathers training data by repeatedly executing a navigate-reach-grasp action sequence (see Fig. 1). To acquire sufficient data in little time, we perform the training experiments in the Gazebo simulator. 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, i.e. positions of the cup on the table (cup_{xy}), which are stored in the matrix $\mathbf{T}$. The 12 cup positions with which the robot is trained are depicted in Fig. 4. For each cup position, the action sequence depicted in Fig. 1 is executed (Line 6) 350 ($\#episodes$) times. After approximately 350 episodes, adding gathering more data does not improve the accuracy of the learned model (on a fixed test set with 100 episodes). The initial robot position $robot_{xy}$ for reaching and grasping is randomly sampled (Line 5), and the result $success?$ (whether the robot was able to grasp the cup or not) is stored in a log-file (Line 10). In Fig. 4 depicts succesful and failed grasp positions in green and red respectively.

Line 9: Computing Classification Boundaries. To discern between good and bad places to perform manipulation actions from, the robot needs a compact model of the large amount of data it has acquired in simulation. To do so, we learn a binary classifier for the observed data with Support Vector Machines (SVM), using the implementation by [17]. We used a Gaussian kernel with σ=0.03, and cost parameter C=20.0. Fig. 4 depicts the resulting classification boundaries for different configurations of task-relevant parameters. 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 for testing.

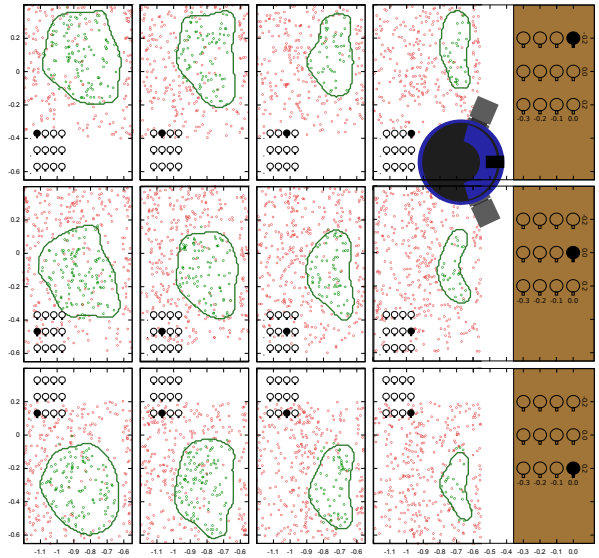


Fig. 4. Successful grasp positions and their classification boundaries. Every sub-image shows the boundary that corresponds to the cup position that is visualized with the black cup. To save space, the table on which the cup is placed is only shown in the right-most graphs, and not all failed data points are drawn. Data points correspond to the center of the robot base.

Line 12-13: Computing the Point Distribution Model. 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. The result is depicted in Fig. 5, where only 4 of the 12 classification boundaries are depicted for clarity. Given the aligned points on the boundaries, we

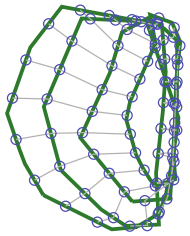
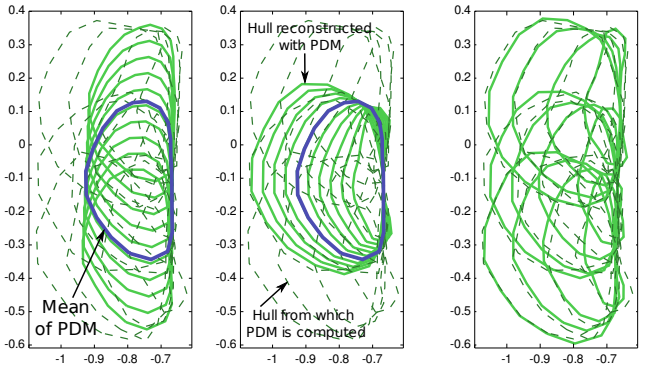


Fig. 5. Point-alignment

compute a PDM. Although PDMs are most well-known for their use in computer vision, we use the notation by Roduit et al. [15], 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 $\mathbf{P}$ 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 extreme values.



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

(b) Reconstructing the boundaries from Fig. 4.

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.

Line 14: Relation to task-relevant parameters. 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 such as cup_{xy} that are varied during data collection. Since the correlation coefficients between the first and second deformation modes and the task relevant parameters $\mathbf{T}$ are 0.99 and 0.97 respectively, we simply compute the linear relation between them with $\mathbf{W} = [\mathbf{1}\ \mathbf{T}]/\mathbf{B}^T$. This approach adheres to the proposed strategy of "learning task-relevant features that map to actions, instead of attempting to reconstruct a detailed model of the world with which to plan actions" [9].

Line 15: Generalized Success Model. The GSM is constructed as a 3-tuple, containing the mean of the classification boundaries $\overline{\mathbf{H}}$, the eigenvectors $\mathbf{P}$, and the mapping from task relevant parameters to deformation modes $\mathbf{W}$.

V. Computing ARPlaces On-line

In this section, we describe how appropriate ARPlaces for manipulation are determined on-line. We call this module 'places for manipulation' (Pla4Man). As can be seen in the computational model in Fig. 3, this module takes the GSM and the estimation of the robot and target object position (as probability distributions) as an input, and returns an ARPlace such as depicted in Fig. 2.

input : gsm = $\langle \overline{\mathbf{H}}, \mathbf{P}, \mathbf{W} \rangle$; *(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 $\mathbf{t}_s$ = samplefromdistribution(objectposition);
5 $\mathbf{b}_s = ([1\ \mathbf{t}_s] \cdot \mathbf{W})^T$;
6 classif_boundary_set.add($\overline{\mathbf{H}} + \mathbf{P} \cdot \mathbf{b}_s$);
7 **end**
8 arplace = $\sum_{i=1}^{\#samples}$ grid(boundary_set$_i$) / #samples;
9 arplace = arplace * robotposition ; *(Convolution)*

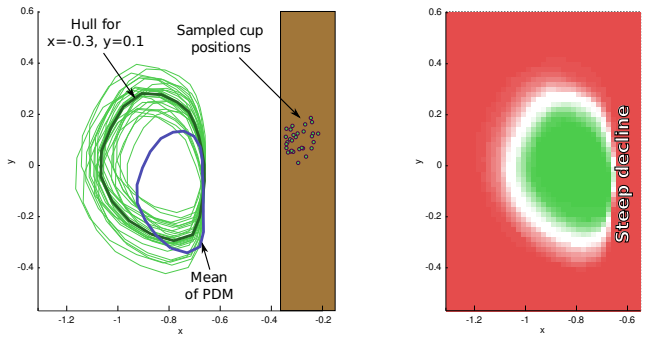
Algorithm 2: Computing ARPlace.

Line 3-6: Monte Carlo simulation. Because of the uncertainty in the estimated cup position, 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 is 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 ($\#samples$) samples from the distribution of the cup position. For each iteration, this yields the task relevant parameters $\mathbf{t}_s = [x_s\ y_s]$. The corresponding classification boundary is determined by first computing the appropriate deformation values from the cup position with $\mathbf{b}_s = ([1\ \mathbf{t}_s] \cdot \mathbf{W})^T$, and then using these to compute $\mathbf{h}_s = \overline{\mathbf{H}} + \mathbf{P} \cdot \mathbf{b}_s$. The boundary $\mathbf{h}_s$ estimates the area in which the robot should stand to be able to make a successful grasp of the cup at position $\mathbf{t}_s = [x_s\ y_s]$.

The distribution of cup position is modeled as a Gaussian with mean $[x\ y]$, and covariance matrix $\begin{pmatrix} \sigma_{xx}^2 & \sigma_{yx}^2 \\ \sigma_{xy}^2 & \sigma_{yy}^2 \end{pmatrix}$, which is provided by our vision-based object localization module [10]. In Fig. 7(a), 30 out of the 100 sampled boundaries are depicted, with a cup position distribution with x=-0.3, y=0.1, σ_{xx}=σ_{yy}=0.05, σ_{xy}=σ_{yx}=0.

Line 8: Summation over classification boundaries. We then generate a discrete grid in which each cell measures $2.5 \times 2.5cm$, and compute the number of classification boundaries that classify this cell as a success. Dividing the result by the overall number of boundaries yields the probability that grasping the cup will succeed from this position. The corresponding probability map, which takes



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

(b) Discretized relative sum of the boundaries.

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

the uncertainty of the cup position into account, is depicted in Fig. 7(b).

Line 9: Uncertainty in robot position. The Adaptive Monte Carlo Localization from the Player project [6] 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 a discretized $(2.5 \times 2.5cm)$ probability distribution of the robot's position[2]. Some results of this convolution are depicted in Fig. 2 (2D) and 8 (3D). Note that this convolution also works for multi-modal distributions as returned by particle filters.

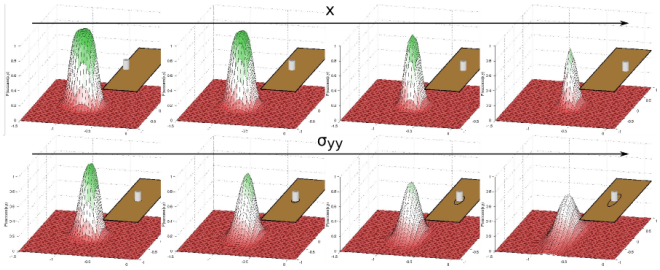


Fig. 8. These images show how varying certain task-relevant parameters affects the shape of the ARPLACE probability map. The table and the cup are drawn to scale in the xy-plane. The video on `http://www9.cs.tum.edu/people/fedrizzi/icar_09` gives an even better impression.

Fig. 8 depicts how the probability map is affected by varying task relevant parameters. Please notice in the first row, how it becomes 'more difficult' (less likely to succeed) to grasp the cup as the cup moves away from the table's edge. The probability maps in Fig. 8 represent the robot's concept of ARPLACE which takes into account the uncertainty in both the pose of the robot and target object.

[2]Note that this uncertainty is in the *current* position of the robot (c), not the position it is navigating to in order to grasp (g). An assumption we make is that as the robot approaches g, the uncertainty in c will become closer and closer to the uncertainty at g, and be equal once $g = c$.

These distributions are generated from a model that is very much grounded in observed experience, as it is learned from observation. Note that this concept 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 concept of successful ARPLACEs. It is the autonomous learning ARPLACE from observed experience that enables us to apply the same algorithm to a wide range of robots and controllers; an advantage over analytical or hand-coded approaches.

A. *Merging* ARPLACEs

So far, we have considered ARPLACEs for single objects O_1. In this case we compute the ARPLACE representation R_{O_1} of grasping O_1 with the right hand and the ARPLACE representation L_{O_1} of grasping O_1 with the left hand. Without further constraints the robot will use the right arm to grasp O_1 if $max(R_{O_1}) \geq max(L_{O_1})$ and the left arm otherwise. In the case where two objects have to be moved, there are several possibilities. The robot can either grasp each object individually by moving to O_1 and grasping it, then moving to O_2 and grasping it. Another possibility is to grasp O_1 and O_2 from one single position. Our ARPLACE representation can handle this generalization easily. Given the ARPLACE representations R_{O_1}, L_{O_1}, R_{O_2}, and L_{O_2}, we can compute the joint ARPLACE probability maps $R_{O_1}L_{O_2}$ (the robot grasps O_1 with its right arm and O_2 with its left arm) and $R_{O_2}L_{O_1}$. This is easily done by a piecewise multiplication of probabilities in the ARPLACE maps, as depicted in Fig. 2. In the next section, we describe how the transformational planner uses merged ARPLACEs to determine the best course of action.

Computing and merging ARPLACEs online as the robot gathers more information about the state of the world is a powerful approach. If a RRT [11] planner is used to compute the optimal configuration C_1 to grasp O_1 and configuration C_2 to grasp O_2, it is not straightforward how to merge C_1 and C_2 to find a configuration C_3, from where both objects can be grasped at once. A RRT approach would have to revise large parts of the prior solutions in a replanning step to solve for a new task. Moreover the configuration space of the new task will be higher, because more joints have to be considered when grasping multiple objects. In our approach the probability of successful execution can be composed from more simple solutions through matrix multiplication, which is a computationally cheap operation.

VI. TRANSFORMATIONAL PLANNING WITH ARPLACE

In this paper, we consider the optimization of a pick-up task: grasping one cup with the left gripper and one with the right gripper. When executing these actions, the robot first drives to a location near the first cup, picks it up, and navigates to a different location to pick up the second cup. This approach works very reliably, and is independent of where the cups are located relative to each other. But it does not perform very well. The overall task could be executed faster by using the same location for picking up both cups.

It is difficult to solve this in a control program without sacrificing generality. The reason is that the two pick-up actions are executed sequentially, and in their default implementation they cannot influence each other or would become less general. In contrast, a transformational planner, as described in this section, is able to detect and locally modify the two locations that cause the sub-optimal behavior.

A. Plan Design

We define plans as robot control programs that cannot only be executed, but also reasoned about. This is important, since it enables a transformational planner to reason about the intention of a specific code part and therefore to infer if a goal has been achieved or not, and what the reason for a failure was. Standard control programs written in RPL [12] are annotated in order to indicate their purpose and make them transparent to the transformational planner. For example, actions that must be performed at a certain location are executed within the context of an *at-location* block. The most important RPL instructions for semantic annotation in the context of pick-and-place tasks are *achieve*, *perceive* and *at-location*. In the context of this paper, we will not give a formal definition of the semantics of these instructions but will describe them only informally.

The *achieve* statement asserts that a successful execution of the statement implies that the logical expression passed as its argument must hold after execution. For example, the statement *(achieve (entity-picked-up ?cup))* states that after executing this instruction, the object referenced by variable *?cup* must be in the robot's gripper[3].

Before manipulating objects, the robot must find the objects and instantiate them in its belief state. The statement *(perceive ?cup)* guarantees that after executing it, the object referenced by *?cup* has been found and a reference to its internal representation is returned.

Manipulation implies the execution of actions at specific locations. Therefore, it must be assured that pick-up actions are only executed when the robot is at a specific location. *(at-location ?location ...)* asserts that code within its context is either executed at the specified location or fails. Please note that transformations which affect the location where actions are performed directly modify the *?location* parameter of such *at-location* expressions. Therefore, *at-location* is the most important declarative plan expression for optimizing ARPLACES.

The declarative expressions explained above form a code tree. Every *achieve* statement can contain several further *achieve*, *perceive* and *at-location* statements. For instance, the goal *(achieve (entity-at-location ?object ?location))* first perceives the object, then picks it up by achieving *entity-picked-up*, which executes the pick-up action within an *at-location* block, and puts the object down by achieving *entity-put-down*, which also contains an *at-location* block. Transformation rules are implemented to replace sub-trees within the code tree by new code. The plan is executed by interpreting every node in the code tree and a so-called task tree is generated. The task tree contains information about plan execution the code tree does not contain, but which is necessary to find behavior flaws.

[3]Please note the lisp syntax, where variables are prefixed with a '?', for example *?cup*, and predicates and functions are pure symbols.

B. Transformational Planning

A transformational planning system consists of three main components: 1) a projection mechanism for predicting the outcome of a plan, 2) a mechanism for detecting behavior flaws within the predicted plan outcome, and 3) mechanisms to fix the detected flaws by applying transformation rules to the plan code. Planning is performed by repeatedly performing these steps until the resulting plan cannot be optimized any more, or a timeout occurs.

Transformational planning enables the robot to detect and fix behavior flaws, such as 1) collisions, e.g. caused by under-parameterized goal locations; 2) blocked goals, e.g. when a chair is standing at a location the robot wants to navigate to; 3) flaws affecting performance, as in our example of picking up two cups from two different initial locations.

1) Plan Projection: A central component of a transformational planning system is an accurate prediction mechanism that generates, based on the plan code, a temporally ordered set of events. For projecting plans, we again use the Gazebo simulator. To not only simulate the execution of plans, but record the events generated by the interaction with the simulated world, we extended Gazebo with plug-ins that signal collisions, perception events, and location changes of the robot and objects. Projection of a plan generates an execution trace that contains the state of the plan, the belief state of the robot and the state of the simulated world for every point in time.

2) Behavior Flaws and Reasoning about Plan Execution: The second component of a transformational planner is a reasoning engine to find pre-defined flaws in the robot behavior. Behavior flaws include not only critical errors in plan execution, like collisions, but also behavior affecting the performance of the executed plan. The latter is investigated in this paper. Behavior flaws are specified using a Prolog-like reasoning engine implemented in Common Lisp. Expressions are written in Lisp syntax and have the form

$$(\langle\text{predicate}\rangle\ [\text{param}]^*)$$

== is unification and *thnot* is a weak inversion predicate, that holds true when it cannot be proven that the passed term holds true.

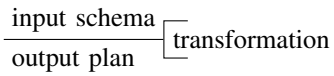
The execution trace generated by plan projection is transparently integrated into the reasoning engine, i.e. the execution trace is queried using Prolog predicates. In combination with a set of facts modeling the semantics of declarative expressions such as *achieve* and *at-location* and concepts of the world, for instance that objects are placed on "supporting planes" (table, cup-board, ...), the information recorded in the execution trace is a central component in order to find behavior flaws.

Behavior flaws can be separated in several classes which can further be specialized. Our planner is aware of the two main classes "detected flaw", which represents errors the robot control program was able to detect and "undetected flaw", flaws that can only be inferred from the execution trace. It includes flaws such as collisions, misperceived objects, but also performance flaws. Listing 1 shows the specification of the flaw generated by two pick-up actions which are executed at different initial locations. The code will be explained in Section VI-C.

Listing 1. Definition of the performance flaw

```
1  (def-behavior-flaw unoptimized-locations
2    :specializes performance-flaw
3    :flaw (and
4            (task-goal ?task-1
5              (achieve (entity-picked-up ?object-1)))
6            (task-goal ?task-2
7              (achieve (entity-picked-up ?object-2)))
8            (thnot (== ?task-1 ?task-2))
9            (optimized-action-location
10             ?object-1 ?object-2
11             ?optimized-location)))
```

3) Plan Transformations and Transformation Rules: After a behavior flaw has been detected, the last step of a planner iteration is the application of a transformation rule to fix the behavior flaw. Transformation rules are applied to parts of the code tree formed by the plan code and cause substantial changes in its structure and the corresponding robot behavior. A transformation rule consists of three parts. The input schema is matched against the plan part to be transformed, the transformation part performs transformations on the matched parts, and the output plan describes how the new code of the respective plan part has to be reassembled.



C. Optimization using ARPLACE

Besides the integration of ARPLACE into the robot control program, it is also integrated into the reasoning engine of our transformational planner. Using two locations for grasping is considered a performance flaw if one would suffice. Informally, we investigate the execution trace for the occurrence of two different pick-up actions, where one is executed at location L_1, and the other one is executed at location L_2. Then we request a location L_3 to perform both actions and the corresponding probability. If the probability of success is sufficiently high, we apply a plan transformation, and replace locations L_1 and L_2 by location L_3.

More specifically, Listing 1 shows the definition of the behavior flaw. The flaw is in the class of performance flaws, i.e. specializing the flaw *performance-flaw* (line 2). In lines 4 to 8, two different pick-up tasks are matched, and the corresponding variables are bound. For that, it uses the predicate *task-goal*, which asserts that a task successfully achieves the corresponding goal according to the semantics of *achieve*. Finally, in line 9, the ARPLACE system is queried for a location to grasp both objects, *?object-1* and *?object-2*. The predicate only holds true when the probability of the new location is sufficiently high (>0.85), i.e. the flaw is detected only if a better location exists. For more detailed information on transformation rules and their application for plan optimization, please see [14].

Note that "sufficiently high" depends very much on the scenario context. In robotic soccer for instance, it can be beneficial to choose fast and risky moves, whereas in safe human-robot interaction, certainty of succesful execution is more important than mere speed. This paper focusses on principled ways of integrating such thresholds in a transformational planner, and relating them to grounded models of the robot's behavior. What these thresholds should be, and how they are determined, depends on the application domain and the users.

VII. EMPIRICAL EVALUATION OF ARPLACE

The hardware platform we use for our experiments is a B21r mobile robot from Real World Interface. Its wheels allow this round robot to move forward and turn around its center axis. Two 6-DOF lightweight arms from Amtec with slide grippers are mounted on this base, allowing for the manipulation of objects at table height.

For localization and navigation, we use several standard modules from the Player project [6], being Adaptive Monte Carlo Localization, pmap for map building, and the AMCL Wavefront Planner for global path planning. These modules use the SickLMS400 laser range scanner and odometry provided by the base. For reaching and grasping, we use a combination of Dynamic Movement Primitives [8] and Vector Fields. The inverse kinematics computations are performed using the Kinematics and Dynamics Library (KDL) from Orocos [16]. Detection and localization of the objects to be grasped is done using the method described in [10]. For debugging and efficient data collection purposes, we also use the Gazebo simulator [6].

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. Fig. 9 shows two images taken during the demonstration. 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.

First, we compared the use of ARPLACE for grasping a single cup (without plan transformations) with another strategy which we call FIXED. FIXED implements the well-in-reach strategy by always moving to a location that has the same relative offset to the target object. The relative location is chosen to be the offset with the best possible overall performance. The cup is placed in three different locations. In one experimental episode, we first determine the real position of the cup, and sample an observed position given the real position p_o of the cup and the covariance matrix $C(p_o)$. 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. 9. A reach-grasp sequence performed at a public demonstration.

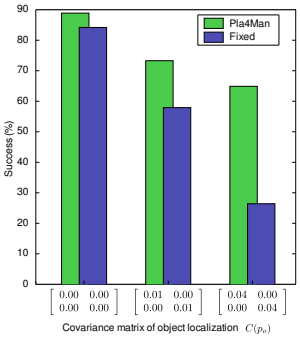


Fig. 10. Result of the empirical evaluation.

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 cup pose. One result is that PLA4MAN always performs better than FIXED. We computed the significance of this performance increase with the χ^2 test; the p-values are depicted in Fig. 10. The only case when the increase is not significant is when there is no uncertainty, a situation that does not arise on the real robot. Another important result is that when the uncertainty rises, the performance of FIXED suffers more than the performance of PLA4MAN. This can be explained by the fact that PLA4MAN tries to stay away from steep declines, when the estimations of the robot get more uncertain.

We then evaluated the merging of ARPLACEs for joint grasping, and applying transformation rules with our RPL planner. Two cups are placed on the table, where the distance between them is varied between 20 and 60cm, with increments of 5cm. Our evaluation shows that grasping two cups from separate positions requires on average 48 seconds, independent of the relative distance of the cups to each other. By applying transformation rules, the default plan is optimized to 32 seconds, which is a significant (t-test: $p < 0.001$) and substantial performance gain of 50%. Above 45cm, two cups cannot be grasped from one position, and plan transformation is not applied.

VIII. CONCLUSION

In this article, we presented a system that enables robots to learn a concept of ARPLACE that is compact, grounded in observed experience, and tailored to the robot's hardware and controller. ARPLACE is modeled as a probability map, which enables the robot to perform least-commitment planning, instead of prematurely committing itself to specific positions that could be suboptimal. We presented a transformational planner that uses ARPLACEs to determine if transformation rules for optimizing plans should be applied. Our empirical evaluation has shown that ARPLACEs improve the robustness of grasping, and when combined with the transformational planner, leads to a substantial improvement in plan execution duration.

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 ARPLACE concepts. New aspects that we are taking into account, are different kinds of objects which require different kinds of grasps. We are also investigating extensions and other machine learning algorithms that will enable our methods to generalize over larger spaces.

ACKNOWLEDGEMENTS

The research described in this article is funded by the CoTeSys cluster of excellence (Cognition for Technical Systems, `http://www.cotesys.org`), part of the Excellence Initiative of the DFG.

REFERENCES

[1] M. Beetz and D. McDermott. Declarative goals in reactive plans. In J. Hendler, editor, *First International Conference on AI Planning Systems*, pages 3–12, Morgan Kaufmann, 1992.

[2] M. Beetz. Structured Reactive Controllers. *Journal of Autonomous Agents and Multi-Agent Systems. Special Issue: Best Papers of the International Conf. on Autonomous Agents '99*, 4:25–55, 2001.

[3] D. Berenson, H. Choset, and J. Kuffner. An optimization approach to planning for mobile manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

[4] R. Bohlin and L. Kavraki. Path planning using lazy prm. In *In IEEE International Conference Robototics and Automation*, 2000.

[5] M. Friedman and Daniel S. Weld. Least-commitment action selection. In *In Proc. 3rd International Conf. on A.I. Planning Systems*, 1996.

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

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

[8] 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.

[9] 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.

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

[11] S. M. LaValle. *Planning Algorithms*, chapter Chapter 5: Sampling-Based Motion Planning. Cambridge University Press, 2006.

[12] D. McDermott. A Reactive Plan Language. Research Report YALEU/DCS/RR-864, Yale University, 1991.

[13] D. McDermott. An algorithm for probabilistic, totally-ordered temporal projection. In O. Stock, editor, *Spatial and Temporal Reasoning*. Kluwer Academic Publishers, Dordrecht, 1997.

[14] A. Müller, A. Kirsch, and M. Beetz. Transformational planning for everyday activity. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 2007.

[15] P. Roduit, A. Martinoli, and J. 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.

[16] R. Smits, T. De Laet, K. Claes, P. Soetens, J. De Schutter, and H. Bruyninckx. Orocos: A software framework for complex sensor-driven robot tasks. *IEEE Robotics and Automation Magazine*, 2008.

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

[18] F. Stulp and M. Beetz. Refining the execution of abstract actions with learned action models. *Journal of Artificial Intelligence Research (JAIR)*, 32, June 2008.

[19] G. J. Sussman. *A computational model of skill acquisition*. PhD thesis, Massachusetts Institute of Technology, 1973.

[20] F. Zacharias, C. Borst, and G. Hirzinger. Positioning mobile manipulators to perform constrained linear trajectories. In *Proc. of the International Conf. on Intelligent Robots and Systems (IROS)*, 2008.