Self-specialization of General Robot Plans Based on Experience

Sebastian Koralewski, Gayane Kazhoyan and Michael Beetz*
{seba, kazhoyan, beetz}@cs.uni-bremen.de

***Abstract*— For robots to work outside of laboratory settings, their plans should be applicable to a variety of environments, objects, task contexts and hardware platforms. This requires general-purpose methods that are, at this moment, not sufficiently performant for real-world applications. We propose an approach to specialize such general plans through running them for specific tasks and thereby learning appropriate specializations from experience. We present a system architecture, which collects data during plan execution for making up supervised learning problems and utilizes the models for specializing the plans in a closed loop. We demonstrate our approach by letting a PR2 robot specialize its general fetch and place plan, whereby learned results are automatically installed into the plan. We show that the specialized plan performs better than the original plan in a statistically significant sense.**

I. Introduction

There is a fundamental trade-off between the specificity and generality of robot plans. Specific plans are tailored for specific tasks, specific robots, specific objects, and specific environments. They can, therefore, make strong assumptions. For example, a robot that is supposed to set a breakfast table can assume that a bowl object should always be grasped from its top side, as grasping from the front or sideways would result in the object not fitting into the hand. This routine – always grasp from the top – is very simple, fast and robust but it only works for this particular context of task, object and environment. The disadvantage of this approach is, that the specific routines are difficult to transfer to other contexts.

The other extreme is to write the plans such that they are very general. To do this, the plans have to include general statements such as "pick up an object", where the object could be any object in any context. The advantage of this approach is that the plans are easily transferable because of their generality but they require very general subroutines such as fetching any object in any context for any purpose. Often these general methods are not sufficiently performant for real-world applications.

In this paper we propose a more promising approach to competently deal with the trade-off between generality and specificity, namely, plan executives that are able to specialize general plans through running them for specific tasks and thereby learning appropriate specializations.

To this end, we frame the robot control problem for performing manipuation tasks in a specific way: we consider it to be the problem of inferring action parameterizations that imply successful execution of underdetermined tasks. For example, in the context of setting a table, relevant action parameters that decide if the outcome of the task is successful would be the inferred locations of the required objects, grasping orientation and grasping points for each object, appropriate object placing locations and corresponding robot base locations. Thus, specializing plans in this setting means learning the distributions of inference tasks that are implied by the specific problem domain.

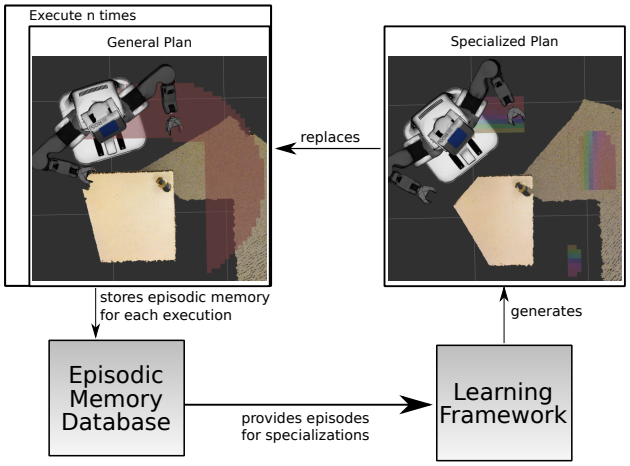

Fig. 1. While the agent performs a general plan, such as picking up a cup, the inference tasks, such as "where to stand to reach the cup", are solved with general methods (red area on the left). To improve the success rate, the plan is specialized based on robot experiences: e.g., successful standing locations are modeled as a probability distribution (heatmap on the right).

To automate this process, we design the general plans such that the inference tasks therein are represented explicitly, transparently and in a machine-understandable way. We also equip the plans with an experience recording mechanism that does not only record the low-level execution data but also the semantic steps of plan execution and their relations. The combination of both allows for autonomous learning and plan specialization (see Figure 1).

The contributions of the paper are, thus, as follows:

- we show that we can design general plans such that they can self-specialize based on experience of a specific task;
- we propose a system architecture, which collects data during plan execution for making up supervised learning problems and utilizes the models for specializing the plans in a closed loop.

We demonstrate the potential power of our approach on a general fetch and deliver plan that is to be applied to the specific task of table setting in a specific environment and with specific objects. We let the robot learn and specialize three different inference tasks. The robot automatically in-

*The authors are with the Institute for Artificial Intelligence, University of Bremen, Germany.

stalls the learned results into the plan. We show that the resulting specialized plan is better than the original general plan in a statistically significant sense. We also discuss and show that the learned information is general enough such that it can be expected to transfer to other tasks and environments.

II. Approach and Architecture

There are many ways to solve the inference tasks for parameterizing mobile manipulation actions. For example, in the context of setting a table for coffee, inferring the likely location of a milk object in the environment can be done by: (1) brute-force search – look on every surface and in every container in the environment, (2) expert system – the expert defined refrigerator to be the likely location of milk, (3) logical reasoning – milk is perishable, so look inside the container for perishable objects, (4) imitation learning – humans always look in fridges when they are searching for milk, (5) learning based on experience – I usually find milk in the fridge. The brute-force solver is the simplest: as long as the domain of each action parameter is known, one can simply iterate over the full domain until a valid solution is found. This approach can be extremely time consuming and requires discretization of continuous domains, on the other hand, it always finds a valid solution if one exists. The other advantage of this approach is that it gives the robotic agent space for exploration.

We apply the brute-force solver in our general plans. However, in order to make this approach sufficiently performant for real-world applications, we rely heavily on domain discretization and heuristics. For example, to infer how to position the robot's base such that it can reach the to be grasped object, we restrict the domain of all positions on the floor to only those that are at robot's arm's length away from the object. This reduces the search space, however, the specifics implied by the problem domain are not being considered. For example, when grasping a bowl from the top, the robot should stand as close as possible to it, however, when grasping a cup from the front, standing too close is counterproductive. In this paper, we apply statistical approaches in order to specialize the plan through learning the specifics implied by the task context.

Figure 2 shows the architecture of the system that implements the approach presented in this paper. The Plan Executive contains General Plans that require inference tasks to be executed. We formulate the inference tasks as questions that are answered by the Question Answering (QA) System. Previously, the queries were answered using the brute-force solver with the Heuristics Collection. In this paper, we implemented a Model Generator that generates learned models stored in the Statistical Model Collection, such that the QA system is able to select relevant models to answer the inference tasks. To be able to generate the models, the Plan Executive contains an Episodic Memory Logger, which is a mechanism for recording robot experiences that does not only log low-level data streams but also the semantically annotated steps of the plan and their relations [1]. The experiences are stored in the Episodic Memory Database,

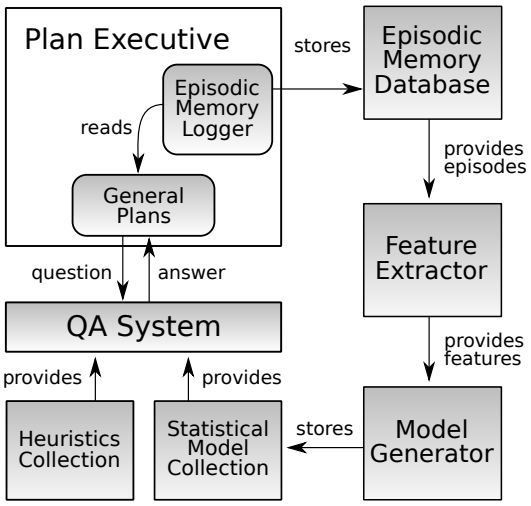

Fig. 2. The architecture of the proposed methodology to transform a general plan into a specialized plan.

from which the Feature Extractor module extracts features for making up supervised learning problems, which are then used by the Model Generator.

First, the robot executes mobile manipulation tasks using the heuristics-based inference engine. Once a significant amount of data is collected, the models are generated. Next, the robot executes its tasks again, now using model-based inference and collects more data. The new data is used to update the existing models, which closes the loop. Thus, by generating the statistical models, we are transforming the general plan into a specialized plan which adapts to the objects and environment in which the agent is performing.

In the following subsections we will go into details of each of the components of our system architecture.

A. General Plans

Plans in our system consist, in a nutshell, from a set of commands to perform actions, executed sequentially or concurrently. The actions are described in an abstract and incomplete way in the format of entity descriptions [2], e.g., an action for transporting an object from A to B is:

```
(an action
    (type transporting)
    (object (an object (type bowl)))
    (to (a location
           (on (an object
                  (type surface)
                  (name breakfast-table))))))
```

The plans implementing higher-level actions contain calls to perform lower-level actions, such that a plan hierarchy is created. For example, the transporting action is implemented by performing three lower-level actions in a sequence:

```
(def-plan transporting (?object ?target-location)
  (let* ((?perceived-obj
            (perform (an action
                         (type searching)
                         (object ?object))))
         (?fetched-object
           (perform (an action
                        (type fetching)
                        (object ?perceived-obj)))))
    (perform (an action
                (type delivering)
                (object ?fetched-object)
                (location ?target-location)))))
```

In action descriptions, *?x* stands for a variable, which gets substituted by its value at runtime, and *x* is an atomic term.

Note that the action descriptions are very abstract: the transporting plan does not tell the robot with which arm to fetch the object, which trajectories to use, where to position the base during manipulation etc. and it does not even say where to search for the object. The missing information is inferred by the plan during execution by issuing queries to the knowledge base. In Figure 3, the three subactions of the transporting plan are shown, along with the parameters of each action that are inferred by asking the QA system.

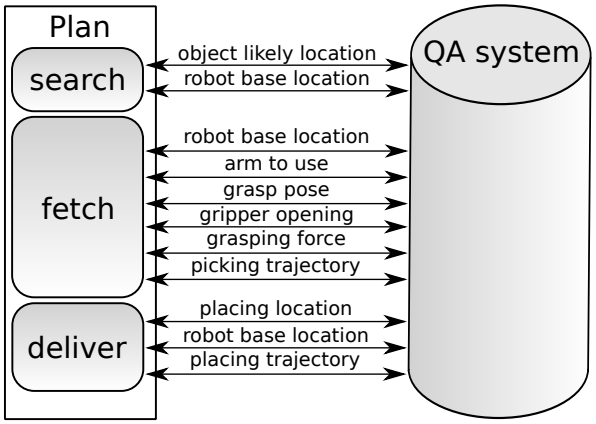

Fig. 3. Subactions of a transporting action infer parameters missing from the plan specification by querying the QA system

Which missing parameters have to be inferred with which queries is defined by the plan designer, such that each action has a number of queries associated with it. Such associations are defined with the Prolog rule *action_grounding*. E.g., for the searching action the predicate is implemented as follows:

```
1  action_grounding(Action,
2            [searching, Object, Loc, RobotLoc]) :-
3    property(Action, [type, searching]),
4    property(Action, [object, Object]),
5    object_likely_location(Object, Loc),
6    RobotLoc = [a, location,
7                [visible-for, robot],
8                [location, Loc]].
```

The code snippet reads as following: for any action description *Action* (line 1), which has a property *[type, searching]* (line 3) and a property *[object, Object]* (line 4), the search location *Loc* is inferred through the query *object_likely_location* (line 5) and the location for the robot to stand to perceive the object is defined as *[a, location, [visible-for, robot], [location, Loc]]* (line 6 – 8). These three parameters – *Object*, *Loc* and *RobotLoc* – are then passed to the *searching* plan (line 2).

The queries that are defined in the plan for inferring missing action parameters of a transporting action and its subactions shown on Figure 3 are listed in Table I. The queries that we consider in this paper are *location_grounding* for reachable locations and *grasp_pose*. The other queries are inferred with the solvers 1 – 4 mentioned above (see Section II).

object_likely_location(Object, LikelyLocation),
location_grounding(Location, Pose)
arm_to_use(Arm)
grasp_pose(ObjectType, GraspPose)
gripper_opening(ObjectType, Distance)
grasping_force(ObjectType, Force)
picking_trajectory(ObjectType, Arm, GraspPose, ObjectPose, Traj)
placing_location(PlacingAction, Location)
placing_trajectory(ObjectType, Arm, GraspPose, PlacingPose, Traj)

TABLE I
QUERIES FOR INFERRING MISSING ACTION PARAMETERS.

B. Episodic Memory Logger

The Episodic Memory Logger is integrated into the plan executive, which allows the logger full access to the plan execution state. Consequently, all data which is processed and generated by the plan execution can be logged and evaluated. As our general plans are the same for real robot execution and execution in a fast simulation environment, the data structures generated from both sources are identical.

C. Episodic Memory Database

To support any possible future learning problem and to be able to reuse old execution logs, our episodic memories contain all the data generated by the agent during an experiment. Our episodic memory format [1] is divided into two parts: symbolic and subsymbolic. The symbolic data is written in Web Ontology Language (OWL). This data includes the plan's complete action hierarchy, its parameters and success status, all questions asked by the agent to the QA system and the corresponding answers and the agent's complete world state. The subsymbolic data currently contains the robot's joint state information and the images which are captured by the robot during the experiment. Even though the subsymbolic data is unstructured, it is linked to the symbolic data through timestamp annotations and is, therefore, easily accessible. By accessible we mean that the subsymbolic data can be queried via RDF query languages.

Episodic memories allow the Feature Extractor module to extract implicit features, which are required to create the statistical models but are not included in the input parameters of the reasoning task. For instance, the *grasp_pose* query, which calculates the grasping orientation and the grasping point (from which side to approach the object and where to grasp) only contains *ObjectType* as an input parameter. The brute-force solver does not need any additional information, as it returns all the known grasping poses for the particular object type. However, utilizing context knowledge, we can say that if the robot is standing very close to the object, it is better to grasp from the top or sideways, and if the object is very far away, only the front grasp would be reachable. This missing information has to be extracted from other parts of the episodic memory. In our case, object's pose relative to the robot can be calculated based on the log of robot's world state at the time point when the grasping action started. Thus, episodic memories allow us to generate data for current and future learning problems without the requirement to modify the logger or the data representation itself.

D. Feature Extractor

Having the episodic memories, it is possible to extract the required features to generate distributions for plan specialization. In this paper we tackle multiple inference tasks. One of the tasks is the grasping pose inference. To determine a successful grasping pose, the Feature Extractor extracts through RDF queries all the successful grasping actions for the given object from the episodic memories, as well as the corresponding robot base poses during the grasping actions. The query results can be afterwards processed into the model designer's desired data structures. Currently, defining the required statistical models for the inference tasks, identifying the required features and writing the RDF queries has to be done manually. In Section VII we discuss possibilities to automate this process.

E. Statistical Model Generator

This module contains templates for defined learning problems. Based on the features provided by the Feature Extractor, the Model Generator decides if it has to generate a new model or update an existing model from the Statistical Model Collection. For instance, when inferring the grasping pose for a particular object, the Model Generator extracts an existing model from the collection if one exists for this object and updates it with new data. Otherwise, if the object has not been encountered before, it takes the defined template and trains a new model. The models designed for our general fetch and deliver plan are discussed in detail in Section IV.

F. Statistical Model Collection

This module stores all the models which were trained with the episodic memories. Currently, each model is represented as a binary file, which is loaded at runtime when the QA System needs it to answer a question.

G. Heuristics Collection

In our current architecture design some heuristics-based inference methods are still present. We use them to generate data with a good coverage of the action parameter domain for learning new statistical models. Additionally, as only 3 inference tasks of a general fetch and place plan are currently covered by the statistical models, heuristics-based methods take care of the remaining 8 inference tasks. Considering that the model definition, feature extraction and training have to be performed manually, generating statistical models for all the inference tasks is considered future work and outside of the scope of this paper.

H. QA System

Based on the received question, the QA System first estimates if it can provide an answer based on learned models or if it has to use heuristics. If a specialized solution exists, the QA system selects, based on the features given in the query and inferred from the episodic memory of the current execution run, the required statistical model. The selection process is based on the same criteria that the model generator uses to decide whether to update an existing model or create a new one.

A special feature of the QA-System is that it can calculate answers for inference questions even before they have been asked. For instance, in our scenario, when the agent starts to perform a fetching action, the QA system gets notified by listening to the episodic memory log of the current execution run, and already starts to infer the solutions for inference tasks of the placing action, which will be asked shortly.

III. Probability Distributions for Fetch and Deliver Tasks

In this section we define the probability distributions, which enable the specialization of general plans. We present probability distributions for inferring object grasping poses and robot base locations for the fetching and delivering actions.

A. Fetch

In this paper, we defined that the following parameters influence the probability of a fetching action being successful: the positioning of robot's base relative to the object, the pose of the robot's gripper w.r.t. the object, robot's arm used for grasping, and different parameters of the object, such as weight, size, material, which we encode in one parameter called *object type*.

The pose of the object w.r.t. the robot is defined by its position and orientation. The orientation is a three degrees of freedom (DOF) parameter with continuous values. To simplify the model of success probability of fetching actions, we restrict the domain of the object orientation, by making the assumption that all the objects, which the robot is manipulating, are supported by a plane. Thus, our model ignores objects that are, e.g., magnetically attached to walls or hanging on a hook. Objects supported by a plane are constrained in two DOF by the plane, and the third DOF is left unconstrained, such that the object can be freely rotated around its upward looking axis. Therefore, we represent the orientation of the object by its supporting face, which is the side of the object that touches the supporting plane, and the robot facing face, which is the face that is oriented towards the robot.

The position of robot's base is also constrained: we assume that the agent cannot grasp an object from any point in the world but only from positions that are of robot's arm's length away from the object. The orientation of the robot base is calculated such that the robot is oriented to always directly face the object.

We call the pose of the robot gripper w.r.t. the object *grasping pose*. It is comprised of the position of the gripper, i.e. the point at which the gripper touches the object, which we call *grasping point*, and gripper's orientation – *grasping orientation*. Each object from robot's manipulation domain has a set of predefined grasping points. These are defined through kinesthetic teaching or by a domain expert. Six grasping orientations are defined for any object: top, bottom, right, left, front and back grasps. The inference task of

determining which grasping orientation to use is based in our system, as most other inference task solvers, on the brute-force approach: the robot tries out random grasp orientations until one works.

Thus, we defined the probability distribution for determining successful grasping poses and robot base locations in the context of a fetching action as follows:

$$\frac{\underset{GO,RFF}{\operatorname{argmax}}\, P(S, GO, RFF \mid RP, OT, SF, ARM)}{\sum_{rp}^{\mathbf{RP}} \underset{GO,RFF}{\operatorname{argmax}}\, P(S, GO, RFF \mid rp, OT, SF, ARM)}$$

where **RP** is the space of all possible robot positions where the robot can still reach the object. We want to determine the max probability of success S, given the robot base position relative to the object RP, grasping orientation GO represented as a discrete variable, object type OT, arm ARM and the object orientation represented as two discrete random variables – supporting face (SF) and robot facing face (RFF) – which take the same values as GO.

The general idea of that probability distribution is that the agent can sample robot base locations for fetching from it. When the agent reached the selected location, its world state is updated and by asking in the next plan step for a grasping pose, the QA system accesses the world state and sends the grasping orientation as answer, which is associated with the sampled location. An additional feature of the probability distribution is that we can incorporate environment information during task execution: we can define that the areas where the robot cannot stand because of collisions with the environment have a probability of zero.

B. Deliver

To be able to deliver an object at a selected location, the agent has to know how to position itself and how to place the gripper with the object attached. We are interpreting a delivering action as an inverse fetching action, since the placing trajectory is equal to a picking up trajectory. Thus, we represent the probability of success of a delivering action similarly to the fetching action:

$$\frac{P(S \mid GO, RFF, RP, OT, SF, ARM)}{\sum_{rp}^{\mathbf{RP}} P(S \mid RFF, GO, rp, OT, SF, ARM)}$$

In the delivering action, the agent already has the object in the hand, so we consider the grasping orientation as given.

IV. Statistical Models for Fetch and Deliver Tasks

To identify which statistical models are required to be able to model the probability distributions presented above, we factorize the distribution as follows:

$$\begin{aligned}
&\underset{GO,RFF}{\operatorname{argmax}}\, P(S, GO, RFF \mid RP, OT, SF, ARM) = \\
&\qquad P(S \mid GO, RFF, RP, OT, SF, ARM) * \\
&\underset{GO}{\operatorname{argmax}}\, P(GO \mid RFF, RP, OT, SF, ARM) * \\
&\underset{RFF}{\operatorname{argmax}}\, P(RFF \mid RP, OT, SF, ARM)
\end{aligned}$$

This representation shows that we need to create statistical models for inferring the robot facing face, the grasping orientation and, afterwards, have a classifier, which uses the previous results as evidence to calculate the probability that the action will be labeled successful.

A. Robot Facing Face

To calculate the corresponding supporting face SF and the robot facing face RFF of the object based on its pose relative to the robot ${}^{r}T_{o}$, we project the x and z axes of robot's coordinate frame (${}^{o}x_r$ and ${}^{o}z_r$ correspondingly) onto the positive and negative x, y and z axes of the object's coordinate frame and take the axis that has the maximum projected value. Here, ${}^{o}x_r$ and ${}^{o}z_r$ are extracted from the orientation of the object's pose:

$$({}^{r}T_{o})^{-1} = {}^{o}T_{r} = \left[\begin{array}{ccc|c} & {}^{o}R_{r} & & {}^{o}t_{r} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} {}^{o}x_r & {}^{o}y_r & {}^{o}z_r & {}^{o}t_{r} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

The resulting vectors are encoded into faces as shown on Figure 4.

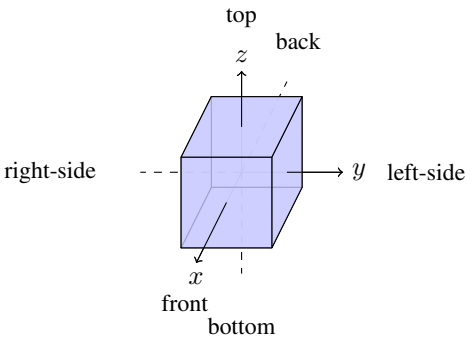

Fig. 4. Encoding of object orientation into discrete "faces"

Given the information of the robot base position, object type and on which surface it stands, we can determine with probability of 1, which object face is facing the robot.

B. Grasping Orientation

To create the probability distribution for the grasping orientation, we decided to use FUZZY-MLNs [3]. Markov logic network (MLN) [4] combines first-order logic with a probabilistic graphical model. A MLN is a set of pairs (F_i, w_i), where F_i is a first-order logic formula and w_i is a real number. Given a finite set of constants $C = \{c_1, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ as follows:

1) For each possible grounding of each predicate appearing in L, $M_{L,C}$ contains one binary node.
   If the ground atom is true, the value of the node is 1. Otherwise it is 0.
2) For each possible grounding of each formula F_i in L, $M_{L,C}$ contains one feature. If the ground formula is true, the value of this feature is 1, otherwise it is 0. The weight of the feature is the w_i attached to F_i in L.

A MLN can be seen as a template to create a Markov network, the probability distribution of which is defined as

$$P(X = x) = \frac{1}{Z} exp\left(\sum_{i=1}^{k} w_i n_i(x)\right)$$

where $n_i(x)$ is the number of true groundings of F_i in x, w_i is a weight associated to the formula F_i and Z is a normalization factor over all possible worlds.

FUZZY-MLN is an extensions of MLN, which uses the fuzzy logic calculus for the evaluation of the formulas. Its distribution is defined as

$$P(X = x) = \frac{1}{Z} exp\left(\sum_{i=1}^{|G|} w_i \pi_x(g_i)\right)$$

where $|G|$ is the number of groundings, $\pi_i(x)$ evaluates the grounded formula applying the fuzzy logic calculus and w_i is the weight associated to the grounded formula g_i.

With the fuzzy calculus, we can utilize taxonomic knowledge, which allows the agent to cope with unseen objects (e.g., a spoon) by relating them to previous seen objects (e.g., a fork). The objects, which the agent is interacting with, are represented as concepts from WordNet[5], where, e.g., spoon and fork are sub-concepts of the cutlery concept. We are calculating the similarity between concepts with the path similarity[1], and by using the FUZZY-MLN we are able to represent this similarity as evidence in our distribution. In any system that is to be used in a real household, the ability to cope with unseen objects is crucial.

An additional advantage of MLNs is that they are white box models and allow us to understand the learned distributions (see V-B for a more detailed discussion).

A large disadvantage of MLNs is that they do not scale well on large datasets. To cope with this problem we decided to apply a multilayer MLN approach for inferring the grasping orientation. The general idea of the approach is that we have one single FUZZY-MLN for each seen object. These are represented in the second layer of our architecture. In the first layer we have a FUZZY-MLN, which, given the object to fetch, infers which MLN from the second layer will infer the most probable successful grasping orientation.

C. Probability of Successful Fetching

The idea of the probability distribution from Section III is that a sample should represent the probability of success for a fetching action, given the provided evidence. Our approach is inspired by[6], where authors bootstrapped SVMs to create a probability distribution for the success rate of standing locations for pick and place tasks. To calculate the success probability, we created a binary classifier, which labels if the fetching action will be successful or not given the evidence such as grasping pose, object type and robot base position. Inspired by our solution for calculating grasping orientation, we decided to create one classifier for each combination of the evidence, excluding the robot position. In theory, it means that we would have to deal with 1296 ($OT \times GO \times RFF \times SF \times ARM$) classifiers, if we would train on 3 objects. However, since, e.g., the spoon's only physically stable supporting faces are bottom and top, the number of classifiers is significantly lower. During our evaluation, 1296 was reduced to 116 binary classifiers that we generated in total. If the QA system is asked for a robot base position for a fetching action, it picks the model based on the evidence and then the selected model uses the robot position as the feature to determine the success rate for the given position. We decided to use generative models (Guassian naive Bayes) due to the small dataset sizes, which is used to train each model individually [7].

[1] `http://search.cpan.org/dist/WordNet-Similarity/lib/WordNet/Similarity/path.pm` (accessed: 2019-01-31)

V. EXPERIMENTAL EVALUATION

One of the advantages of our approach is that one does not need a specifically designed script to generate the data for the training set. The robot can simply execute its usual tasks and collect the data while doing so. Due to general inference methods not being quite optimal, the number of negative samples is quite high. Nevertheless, all the samples – both positive and negative ones – are collected for learning purposes. The aim of this paper was to optimize the process of sampling in order to decrease the amount of negative samples and, therefore, failures. In order not to overfit our training dataset to the evaluation dataset, the pick and place tasks the robot performed varied in object type, object starting and goal poses, and robot poses for fetching and delivering. In the following, we explain how the training set was generated and how we evaluated the system.

A. Data Generation

For the data generation, the robot performed its usual pick and place plans with 3 different objects: *fork*, *cup* and *bowl*. The scenario was as following: (1) all the objects are spawned on one side of the kitchen (*sink-area*) in random positions and with random orientations, (2) the robot picks a random object from the set of four and fetches it with one of its two arms and with one of the possible known grasps for this object, picked randomly, (3) the robot delivers the object to the other side of the kitchen (*kitchen-island*) to a randomly picked point on the surface with a random orientation, (4) the robot fetches the object again and delivers it back to the *sink-area* onto a randomly picked location on the surface, (5) the loop continues with the next randomly picked object. We decided to generate 500 episodic memories for the model generation to evaluate how much improvement gain can be achieved already with a small training set.

B. Model Generation

The Feature Extractor module of our system extracted the required features to train our MLNs and the naive Bayes classifier from the 500 generated log episodes. Since we only had a small dataset, we decided to skip individual validation of the generated models. However, since the MLNs are white box models, we could understand what the agent learned

from the dataset. For instance, it learned that the agent was more successful when grasping the cup from its robot facing side instead of grasping it from the top or sideways. Figure 1 shows how the heuristic-based distribution for robot's base location differs from the learned one. One can observe that the agent learned that in order to successfully grasp a cup from the front, the robot should stand a bit further away from the object, due to joint limit restrictions of robot's elbow.

C. Evaluation in Projection Environment

As evaluation criterion of execution in projection environment we have chosen to count the number of successful fetch and deliver actions as well as the number of failures, i.e. the general number of retries while performing those actions. For the projection, we are not measuring the execution time, since robot movements in simulation can be executed in nanoseconds and, therefore, difficult to compare.

For the test dataset we have chosen 3 objects – *cup*, *bowl* and *spoon*. Compared to data generation scenario, we replaced the fork with the spoon, which is an object unseen in the training dataset. In order to minimize the influence of irrelevant parameters involved in pick and place actions, we have chosen 4 specific object configurations, including the object spawning location and the goal location. We have also fixed the robot arm (left or right) used for the fetching and delivering. The parameters that remain varied are the robot base locations and the grasp orientations, which the robot choses on its own. The scenario is as following: (1) the robot fetches the *bowl* from its initial location at the *sink-area* and delivers it to its goal location on the *kitchen-island*, (2) the robot transports the *spoon* from *sink-area* to *kitchen-island*, (3) the robot takes away the *cup* from the *kitchen-island* and brings it to the *sink-area*.

Initially, we have performed the scenario with the general plan, which resulted in the heuristic-based methods answering the queries. We performed 25 runs of each of the 4 configurations. Next, we performed 25 runs of the 4 configurations again with the specialized plans. Table II shows the performance of all plans over the 4 scenarios.

Plan	Fetch	Deliver	#Actions	#Retries	#Retries/ #Actions
Projection					
G	274/300	239/300	513/600	488	0.95
S	296/300	280/300	576/600	269	0.47
Real Robot					
G	11/12	8/12	19/24	24	1.26
S	10/12	9/12	19/24	9	0.47

TABLE II

OVERVIEW OF THE NUMBER OF SUCCESSFUL ACTIONS AND RETRIES, COMPARING THE GENERAL PLAN WITH THE SPECIALIZED ONE

If all plans executed in projection would perform successfully, there would be 300 successful fetching and delivering actions in total, i.e. a plan could perform 600 successful actions in total. However, some actions failed and some were not executed as a result. Thus, if fetching of an object failed, the delivering action on that object was not performed anymore. This means that the number of retires in the general plan might be significantly higher if the general plan would have the same amount of successful fetching actions.

The evaluation shows that the specialized plan outperformed the general plan in any category. The specialized plan performed more actions successfully and also needed only almost half of the number of retries compared to the general plan.

D. Evaluation on Real Robot

On the real robot the evaluation criteria have been chosen to be the number of successful actions and retries as well as execution time. As it is very difficult to collect enough data points on a real robot to give a meaningful empirical estimate of the costs, we do not claim the experiments on the real robot to be a valid empirical evaluation, rather, these experiments are to prove that the framework is feasible on the real robot and has promising improvements in the number of failures and execution time that we have observed from a few experimental runs.

We have executed each of the 4 configurations once with the general plan and once with the specialized plan. Table II shows the number of successful actions and retries. In total the number of successful actions was the same between the plans, however the specialized plan was able to execute those actions with 60% less retries. Table III shows that the specialized plan reduced the average execution time of fetching on a real PR2 by almost 30%. The delivering execution time got reduced by at least 17% in average.

Plan	Avg. Fetching Time	Avg. Delivering Time
General	151 secs	115 secs
Specialized	109 secs	95 secs

TABLE III

COMPARISON OF AVERAGE ACTION EXECUTION TIME BETWEEN THE GENERAL PLAN AND SPECIALIZED PLAN PERFORMED ON A PR2

VI. RELATED WORK AND DISCUSSION

In this paper, we presented a general learning framework that is capable to improve the performance of mobile manipulation actions by learning answers to queries that infer parameters of the actions. Thereby the emphasis is on being able to learn the answers to any inference query, be it in continuous or discrete domain, independent of the amount and type of features, if the features are explicitly given or have to be inferred, etc. The idea of a general framework that can learn any action parameter has been considered previously [8], [9]. The main disadvantage of such frameworks is that the learning problems and the models used to solve them have to be explicitly stated inside the robot's plan. This means that if the designer decides to change the underlying model or the model requires a new feature, robot's plan has to be rewritten. In this paper, we present a learning framework that is seamlessly integrated into the plan, such that the plan is not even aware if the answers to the inference tasks are coming from the learning framework or from a different, e.g., heuristics-based, method. The training data is, thereby,

automatically acquired during normal robot execution, such that over time the plan performance is improved significantly, without any changes happening in the plan itself. If a model requires a new feature that is not explicitly given in the plan, the framework can infer it based on execution logs and robot's world state.

The state of the art in supervised learning demonstrates extreme improvements in manipulation tasks with deep reinforcement learning approaches [10], [11], [12], [13], mostly concentrating on learning a specific skill, e.g., calculating grasps or in-hand manipulation. Some apply end-to-end approaches to solve longer-horizon multi-stage manipulation tasks [14]. In this paper, we consider more complex mobile manipulation tasks, i.e. combined manipulation and navigation, and solve it with a different approach, which is to split one big learning problem of learning trajectories to execute the full task into smaller and simpler subproblems. As statistical models for our learning problems, we have chosen MLNs, where we can utilize first-order logic to represent complex relations and are able to understand what the agent learned from its own experiences. We consider that MLNs being white box models is an advantage compared to black box models such as neural networks, as it allows to identify important features that are robot-independent or object-independent and transfer this knowledge to other robots and environments. It also allows to reuse full models, e.g., we applied the model of reachability learned for picking up actions also to placing actions with similar success. Transferability is especially important in the context of generalized fetch and place, which is expected to work in a diverse environment such as a human household.

Data-hungry deep learning methods require millions of data points, which is very challenging to collect for such long time horizon tasks as mobile manipulation. We are not claiming that MLNs are outperforming the deep learning solutions at the moment. However, it is interesting to see that already 500 data points were enough to improve the agent's performance significantly and, in combination with FUZZY-MLN, we can transfer the knowledge to unseen objects.

VII. Conclusion and Future Work

In this paper we presented an approach to utilize the robot's experiment experiences to generate specialized plans from general ones. Our approach is an architecture which generates and updates statistical models to enable plan specialization. We compared the general and specialized plan by letting the agent perform various manipulation tasks, using once the general plans and once the specialized. The evaluation showed that a small number of data points was already enough to create a specialized plan, which improved agent's performance of manipulation tasks significantly.

In its current state, the proposed architecture is not able to automatically identify the required statistical models for the learning problems and the required features for improving the answers of the inference tasks of the general plan. Therefore, in the nearest future we will focus on auto machine learning for our framework.

As statistical models we use MLNs, for which, to the best of our knowledge, no incremental learning algorithms exist. As a result, we have to retrain the models every time with the complete training set. Thus, we will additionally focus on online learning algorithms for MLNs to enable incremental learning. Currently, we apply our framework to only three inference tasks, and our goal is to learn answers to *all* queries that the robot can ask in the context of fetch and place.

Acknowledgments

This work was supported by DFG Collaborative Research Center *Everyday Activity Science and Engineering (EASE)* (CRC #1320) and DFG Project *PIPE* (project number 322037152).

References

[1] D. Beßler, S. Koralewski, and M. Beetz, "Knowledge representation for cognition- and learning-enabled robot manipulation," in *Proc. 11th International Cognitive Robotics Workshop (CogRob 2018)*. AAAI Press, 2018, accepted for publication.

[2] G. Kazhoyan and M. Beetz, "Programming robotic agents with action descriptions," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[3] D. Nyga and M. Beetz, "Reasoning about Unmodelled Concepts – Incorporating Class Taxonomies in Probabilistic Relational Models," in *Arxiv.org*, 2015, preprint. [Online]. Available: http://arxiv.org/abs/1504.05411

[4] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[5] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[6] F. Stulp, A. Fedrizzi, L. Mösenlechner, and M. Beetz, "Learning and reasoning with action-related places for robust mobile manipulation." *J. Artif. Intell. Res.(JAIR)*, vol. 43, pp. 1–42, 2012.

[7] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in neural information processing systems*, 2002, pp. 841–848.

[8] A. Kirsch, "Robot learning languageintegrating programming and learning for cognitive systems," *Robotics and Autonomous Systems*, vol. 57, no. 9, pp. 943–954, 2009.

[9] J. Winkler and M. Beetz, "Robot action plans that form and maintain expectations," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 5174–5180.

[10] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," 05 2017, pp. 3389–3396.

[11] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 3406–3413.

[12] J. Tobin, W. Zaremba, and P. Abbeel, "Domain randomization and generative models for robotic grasping," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3482–3489, 2018.

[13] M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. W. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *CoRR*, vol. abs/1808.00177, 2018.

[14] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," in *Proceedings of the 1st Annual Conference on Robot Learning*, vol. 78. PMLR, Nov 2017, pp. 334–343.