From Low-Level Trajectory Demonstrations to Symbolic Actions for Planning

Nichola Abdo and **Henrik Kretzschmar** and **Cyrill Stachniss**
University of Freiburg
Department of Computer Science
Georges-Köhler-Allee 079
79110 Freiburg, Germany

Abstract

Robots that should solve complex manipulation tasks need to reason about their actions on a symbolic level to compute plans comprising sequences of actions. Planning, however, requires knowledge about the preconditions and effects of all the actions. In this work, we present an approach that allows a robot to learn manipulation skills from teacher demonstrations. Our approach enables the robot to learn to physically execute the motion needed to perform the actions, and, most importantly, to infer the preconditions and effects. Our system can express the acquired manipulation action as symbolic planning operators and thus can use any modern planner to solve tasks that are more complex than the individual, demonstrated actions. We implemented our approach on a PR2 robot and present real world manipulation experiments that illustrate that our system allows non-experts to transfer knowledge to robots.

Introduction

Future service robots must be flexible enough to carry out a variety of day-to-day tasks under diverse conditions. It is, however, practically impossible to preprogram a robot for all kinds of situations that occur in the real world. Therefore, we need means for easily instructing robots and teaching them new skills by non-experts.

Planning for solving complex manipulation tasks can be done using low-level motor commands or on a symbolic level. Computing solutions based on low-level motor commands is infeasible due to the high-dimensionality of the resulting planning problem and thus robots need to reason about their actions on a higher level. Computing plans of high-level actions to achieve some goal, however, requires a high-level symbolic representation describing the preconditions and effects of the robot's actions.

In this work, we aim for a fast and intuitive learning process that allows the robot to learn new actions such that it can later on reason about the actions on both, the motion level and a symbolic level. Our approach is based on learning by demonstration. The robot observes a human teacher and learns how to physically execute the movements in order to solve a manipulation task. In addition to that, the robot learns the preconditions and effects of the actions, which are both needed for planning. While using the learned actions to solve manipulation tasks, the robot monitors its performance and reacts to unexpected changes. In summary, our system (i) encodes the low-level movements, (ii) estimates the preconditions and effects of the individual actions, and (iii) generates a planning problem definition that allows state-of-the-art planning systems to solve new tasks using the learned actions. We implemented our approach and carried out extensive experiments using a PR2 robot to illustrate the capabilities and flexibility of our system. We demonstrate that our approach enables the robot to autonomously solve tasks that are more complex than the basic actions that have been demonstrated to the robot.

Related Work

In the literature, there are various approaches for transferring task knowledge from humans to robots. In recent years, imitation learning methods have become popular to encode robot motion. See Billard et al. (2008) for an overview. The key idea is to speed up the learning process by exploiting demonstrations given by a teacher. For example, Bentivegna et al. (2004) demonstrate to a humanoid robot how to play air hockey by learning primitives that the robot can use in new situations. The robot learns how to choose a primitive in a given situation and practices these primitives to improve its performance. Asfour et al. (2006) use hidden Markov models to encode and reproduce demonstrated actions. Dynamic movement primitives (DMPs) are popular to learn control policies for robotic manipulators from demonstrations and to generalize the movements to new situations. Our approach also relies on these movement primitives as proposed by Pastor et al. (2009) to encode the low-level movements of the actions. Calinon and Billard (2008) propose Gaussian mixture models to represent the variance over time in the demonstrated trajectories of a manipulator to exploit this information in the reproduction step. Also Eppner et al. (2009) consider the variance to guess less relevant parts of the demonstrations. Our method analyzes the variations in the state during the demonstrations to identify the preconditions and effects of the individual actions. This allows our approach to generate a symbolic representation of the actions, which is then used for planning purposes.

There are also a number of approaches that aim at teaching robots skills on a symbolic level for task planning based on teacher demonstrations. Veeraraghavan and Veloso (2008)

demonstrate sequences of actions to teach a robot a plan for solving sequential tasks that involve repetitions. They instantiate preprogrammed behaviors and then learn the corresponding preconditions and effects. Pardowitz, Zöllner, and Dillmann (2006) extract task-specific knowledge from sequences of actions. The robot extracts the relevant elementary actions and task constraints from teacher demonstrations of pick-and-place actions when setting a table. Manipulation skills are arranged in a hierarchical manner with macro actions encompassing elementary ones. The preconditions and effects of actions are expressed by predetermined properties like the relative positions of the objects. Ekvall and Kragic (2006) also provide a robot with demonstrations of tasks related to setting a table. The robot incrementally learns the constraints for each task with respect to the order of executing the actions. This knowledge is then used to choose the best strategy for solving a new task. To identify the different states observed during the demonstrations, they apply k-means clustering to the relative positions and orientations of the objects and inspect the cluster variances. Zhuo et al. (2009) learn action preconditions and effects for hierarchical task networks from given observed decomposition trees. Such trees describe how a task can be broken down into smaller subtasks.

Similar to Ekvall and Kragic, our system applies k-means to features to identify preconditions and effects of actions. By inspecting the variance within the extracted clusters, our system additionally tries to determine if a certain feature or aspect of the action is relevant as a precondition or effect and to recognize similar states across different actions. Unlike the approaches above, we do not require to demonstrate sequences of actions to the robot or to provide task decomposition information. Instead, our system learns the individual actions by demonstration, and uses the identified conditions to chain the actions in plans for solving a variety of tasks.

Many researchers adopt object action complexes (OACs), as presented by Krüger et al. (2009), as a representation that combines low-level robot control and high-level planning. OACs consider objects as important to the robot in terms of the actions that can be applied to them. Pastor et al. (2009) suggested adding a symbolic meaning to DMPs such that they can be used for high level planning in the context of OACs. However, this was not realized in their work. There are approaches that learn simple cause-effect rules based on simulated actions or exploration so that they can be integrated into the OAC frameworks (Petrick et al. 2008). Furthermore, Omrcen, Ude, and Kos (2008) present an approach that allows a robot to learn the effect of poking different objects by exploration. The approach uses a neural network to learn the relation between pushing actions and the predicted motion of the object. This is then used to plan for applying several poking actions to move objects to desired locations.

In contrast to these techniques, our approach does not rely on exploration or simulation to learn the effects of carrying out actions. Instead, our system learns both the preconditions and effects of the actions from a few teacher demonstrations and represents the actions as high-level planning operators. From the same demonstrations, our approach learns the trajectories of the manipulator using dynamic movement primitives.

Overview

Our approach allows a robot to acquire and combine actions to solve complex tasks. By observing a teacher, the robot learns how to physically execute individual actions. The robot then infers symbolic information that allows it to combine the learned actions via a planning system to solve complex tasks that have not been shown to it.

Our work enables a robot to identify preconditions that have to be satisfied to carry out a certain action as well as the effects of an action. An example of such a precondition is the fact that the gripper of the robot needs to be open before an object can be grasped. Identifying preconditions and effects is done by estimating the distribution of world states to identify patterns while the teacher repeatedly demonstrates the same action. These patterns lead to logical predicates, allowing the robot to translate low-level sensory data into a high-level logical representation and verify when the preconditions or effects are satisfied. Consequently, our robot can associate the low-level movements of its end effector with symbolic information and to derive a definition of the action in the Planning Domain Definition Language PDDL. Given the PDDL description, the robot is able to use any modern planning system to solve tasks that are more complex than the individual actions that have been demonstrated to it.

Perception and Predefined Features

Our approach assumes that the robot can identify relevant objects in the scene along with their poses. For this work, we attached checkerboard markers to the relevant objects and used an out-of-the-box detector that is available in the robot operating system ROS. The detector provides the robot with the types of the objects (e.g. block, table, . . .) and their poses. The robot also uses its laser scanner, for example to estimate the state of doors or for 2D obstacle avoidance. Note that our approach is orthogonal to the perception problem. Therefore, our method should be applicable in the same way when using a system for marker-free detection of objects.

To encode the state of the world, our approach relies on features, which can take continuous or discrete values. We derive the preconditions and effects of the different actions using the values of these features during the demonstrations. So far, we applied our system to solve blocks world-like tasks and to operate doors. We defined continuous features such as the opening of the gripper, the relative poses between the gripper and manipulated objects, and the relative poses between objects. We furthermore defined discrete features such as the visibility of objects, and the state of the door. Depending on what the user demonstrates, new features may need to be defined. This can be done easily and does not affect already learned actions.

Recording and Encoding Demonstrations

To teach the robot basic actions, we use kinesthetic training, *i.e.*, the teacher moves the manipulators of the robot, as illustrated in Fig. 1. This method is rather accurate and does not require extra sensors since the robot can directly record the movements using its own encoders. Our approach allows


Figure 1: Examples of kinesthetic training showing how to place a block on another one and how to operate a handle.

for demonstrating individual actions one by one and does not require demonstrating sequences of actions.

A popular way to encode movements of a manipulator are DMPs. Our approach uses DMPs as described by Pastor et al. (2009) to encode the trajectory of the robot's end effector as observed in the demonstrations. DMPs allow us to easily adapt the movement to different situations such as new starting or goal points. Our system groups the learned DMPs together so that multiple DMPs for each action are available to the robot. In our experiments, we recorded 10 demonstrations per action. Moreover, we propose in the next section a method for extracting the preconditions and effects from these demonstrations.

Identifying Preconditions and Effects

The preconditions of actions and their effects are expressed in terms of features. To identify the preconditions and effects based on a set of demonstrations, we inspect the recorded values of all features at the beginning and at the end of each demonstration. For each feature, we then seek to find patterns in its values to decide whether or not it is important for an action.

General Problem and Assumptions

In the most general case, the robot cannot be sure that an action can be carried out unless the current state of the world is identical to a state observed in one of the demonstrations. Otherwise, a precondition might not be satisfied and executing the action might fail. Finding the preconditions only based on successful demonstrations does not lead to satisfying results without further assumptions. The resulting unsupervised learning problem can be viewed as a one-class classification problem in which only positive examples are provided. In our case, the examples correspond to demonstrations in which the preconditions and the effects are satisfied.

Such problems can be addressed using density estimation methods or by only estimating the boundaries of the distribution (Schölkopf et al. 2000). A simple nearest neighbor approach considers all states to be fulfilling the conditions that are similar under a distance function to the states observed during the demonstrations. In our setting, a key disadvantage of the nearest neighbor approach is the fact that a large subset of the features are not relevant as preconditions and effects of most actions. As a result, the entire feature space would have to be populated by samples to make the robot ignore irrelevant feature values. This is infeasible in practice since only a few demonstrations can be provided by a teacher. In contrast to the nearest neighbor approach, one-class classification methods such as single-class support vector machines (SVMs) could be more appropriate approaches in this setting (Schölkopf and Smola 2002).

To tackle the above mentioned problem, we assume that the preconditions and effects of the actions can be expressed in terms of the predefined features and their corresponding values, and that the individual features are independent of each other. We consider that a feature is relevant as a precondition or an effect of an action if its values follow certain patterns throughout the demonstrations. Moreover, we assume that the teacher provides demonstrations with variations. If the teacher demonstrates actions with too few variations, the robot may identify additional preconditions or effects that are irrelevant in reality.

Estimating Preconditions and Effects by Analyzing the Variations in the Demonstrations

In our approach, three questions have to be answered: First, which features are relevant for an action as a precondition or as an effect? Second, if a feature is regarded as relevant, which values are typically observed and how to derive a logical predicate that encodes the decision whether the precondition or effect is fulfilled? Third, given two logical predicates, how to decide whether both represent the same condition? The last question is important to allow a planning system to verify beforehand whether the effects of an action match the preconditions of another one. This is essential for planning.

As preconditions, we consider features that take the same or similar values at the beginning of all demonstrations of an action. The same holds for the effects: features that always take similar values after having executed an action are considered to be an effect of that action. Informally speaking, for each action independently, we estimate for each feature the region of the feature space that covers the samples corresponding to the demonstrations. By analyzing the volumes of such regions, we can decide whether the feature is relevant or not. This allows us to derive the logical predicates and to estimate whether two predicates model the same condition or not.

Note that for an action, we only consider those features that involve objects or things that are close to the robot during the demonstrations or that involve only the robot itself. This allows us, for example, to ignore the state of a window, potentially in a different room, while the teacher shows the robot how to operate the door.

Features Taking Continuous Values There exist multiple ways for estimating the boundaries of regions in a potentially high-dimensional space that are populated by samples. Approaches to one-class classification belong to this class of algorithms such as single-class SVMs (Schölkopf et al. 2000). Alternative approaches are the one-class k-means, the one-class PCA, and the one-class k-nearest neighbor (Kennedy, Namee, and Delany 2009).

Compared to most other learning problems, we suffer from having only a small number of training examples. A user is expected to provide around ten demonstrations of one action. This will lead to ten sample points in feature space. With so

few training examples, applying techniques such as SVMs is likely to provide results that do not generalize well. For example, in the context of image classification based on a small number of training images, simpler methods such as nearest neighbor approaches are reported to perform better than SVMs (Boiman, Shechtman, and Irani 2008). Since we consider training sets in the order of 10 sample points, we propose to not use single-class SVMs but follow a simpler approach and apply one-class k-means. Note that one-class k-means does not mean that $k = 1$ but that all centroids represent the single class jointly, which allows for covering multiple modes. Additionally, we consider the variance of the samples within each of the identified clusters.

If all samples are concentrated in one cluster, we can directly compute the variance in the individual feature values over multiple demonstrations. If the variance is small, we can regard the feature as relevant and thus to be a precondition or an effect. There are, however, situations in which such a simple criterion is not successful. For example, before grasping a block, the gripper must be open and the gripper must either be on top of the block or at its side (top grasp or side grasp). For the robot, it can be advantageous to consider this as two distinct actions, but the teacher may teach that as one grasping action. To allow for considering such situations in which feature values can be centered around multiple possible values, we apply k-means clustering to the individual feature values and then analyze the variances in each cluster.

Since the teacher demonstrates an unknown number of ways of performing an action, the system typically does not know the number of clusters k to look for in advance. We therefore perform multiple iterations of k-means with increasing values for k from 1 up to $\sqrt{N/2}$, where N is the number of data points. Note that this upper limit is a heuristic, as suggested by Mardia, Kent, and Bibby (1979).

For a cluster to be considered as representing an important aspect of the action, its average squared intra-cluster distances should not exceed a certain limit. This predefined limit reflects the desired accuracy of executing the manipulation action. For a cluster c with mean μ_c, this condition can be expressed as

$$\frac{1}{N_c}\sum_{i=1}^{N_c} dist(v_i^c, \mu_c)^2 \leq \varepsilon_1, \tag{1}$$

where N_c is the number of data points assigned to cluster c and v_i^c is the value of the i^{th} data point in the cluster. Here, $dist(.,.)$ is a distance measure for the feature under consideration. This can either be the Euclidean distance or the angular difference based on an angle/axis representation in case of a rotation, *i.e.*,

$$dist_{rot}(R_v, R_\mu) = angleOf(R_v R_\mu^{-1}), \tag{2}$$

where R_v and R_μ are the corresponding rotation matrices. The value ε_1 is the maximum allowed variance for each cluster (that is separately defined for the Euclidean and the angular distance function). If the condition in Eq. (1) is satisfied for all clusters, our system could identify a potentially multi-modal pattern in the input data and considers this pattern as a precondition or effect. Then, no further increase of k is needed. However, if this criterion fails for all values of k, the system determines that the feature is irrelevant to the action since no pattern could be found.

To finally make the decision if a state satisfies a precondition or an effect, we have to check, according to the one-class k-means formulation, whether the minimum distance between the centroids and the current feature value v is smaller than a threshold or not. We represent this fact by so-called predicates that are used by the planning system. A predicate $\mathcal{P}_f$ is defined for each action for which the feature f is relevant as a precondition or effect. We may generate an individual predicate for the precondition and effect as well as for each cluster c. The predicate is defined as:

$$\mathcal{P}_{f,c}(v) = \begin{cases} true & \text{if } dist(v, \mu_c) \leq d_{max} \\ false & \text{otherwise,} \end{cases} \tag{3}$$

where d_{max} is a threshold defining the maximum allowed distance to the centroid.

Features Taking Discrete Values Besides features taking continuous values, we also consider features taking discrete values. An example of such a feature is object-is-visible, which can be $true$ or $false$. To decide whether a discrete-valued feature is relevant for an action, we compute the entropy of the distribution of the feature values during the demonstrations. The entropy H is a measure of uncertainty and is defined as

$$H(f) = -\sum_{l=1}^{L} P(f = v_l) \log_2 P(f = v_l), \tag{4}$$

where $P(f = v_l)$ is the probability that the feature f takes the value v_l (out of L possible values). The distribution over the values is computed based on the observations. A low entropy indicates that the probability mass is concentrated in one state (or a few states, depending on the number of possible states) and thus indicates a low variation of the feature value over the demonstrations. To avoid overfitting in case of few demonstrations, a Dirichlet prior can be added.

In our approach, we consider a feature f as relevant if $H(f) < \varepsilon_2$, where ε_2 is a threshold specifying the certainty the system should have about the value of this feature. The value that the feature has to take to satisfy the precondition or effect is then given by

$$v^f = \underset{v_l \in \{v_1 \ldots v_L\}}{\operatorname{argmax}} P(f = v_l). \tag{5}$$

In most cases, the discrete features are binary variables taking $true$ and $false$ as possible values, but there exist also features that can take more than two values. An example of a feature that we found useful in our experiments is a three-state representation of a door: the door can be completely open so that the robot can go through it, or it can be partially open so that the robot first needs to open it further to pass through without having to operate the handle, or the door may be closed completely.

Similar to the continuous case, we can derive a boolean predicate $\mathcal{P}_f(v)$ that is later on used in the planning process

to test whether a discrete precondition is fulfilled as:

$$\mathcal{P}_f(v) = \begin{cases} true & \text{if } v = v^f \text{ and } H(f) < \varepsilon_2 \\ false & \text{otherwise.} \end{cases} \quad (6)$$

Identifying Identical Predicates

Whenever the user teaches actions individually and not as a sequence, the predicates have to be learned for each action individually. To allow a planner to compute a plan, we need to identify which predicates from one action are the same as the predicates from other actions. Consider two predicates $\mathcal{P}_f^{a_1}$ and $\mathcal{P}_f^{a_2}$ generated from two different actions a_1 and a_2 but from the same feature f. To decide if they represent the same condition, we consider the feature values from the demonstrations of a_1 and a_2 as a merged sample set. The predicates $\mathcal{P}_f^{a_1}$ and $\mathcal{P}_f^{a_2}$ will be merged into one predicate if the merged sample set still fulfills the criterion given in Eq. (1) (or the entropy criterion for the discrete case). Otherwise, $\mathcal{P}_f^{a_1}$ and $\mathcal{P}_f^{a_2}$ remain individual predicates.

Generating the PDDL Description

Over the last 15 years, the Planning Domain Definition Language PDDL has been established as a standard language for defining planning problems. Therefore, we developed a system that automatically derives a PDDL description which allows us to easily use most out-of-the-box planning components.

To generate a PDDL description, we first need to define the objects involved in the planning process and their types. This is obtained from the perception system as mentioned before. Second, we need the predicates that define the state of the planner, and which are computed using the method described above. Third, the start and goal states need to be specified. The start state is simply obtained by evaluating all predicates according to the current observations. The goal, obviously, has to be provided by the user in terms of the predicates. Finally, the actions with their preconditions and effects on the state have to be provided.

For expressing each action in terms of its preconditions and effects, we consider the different possible cases for each relevant feature f. Since f could be relevant as a precondition, an effect, or both, our system adds the appropriate predicate, $\mathcal{P}_f$, or its negation in the preconditions or effects part of the PDDL operator. An example of a generated PDDL operator for approaching a block from the top to grasp it is:

```
(:action reachBlockTop
 :parameters (?b-block ?g-gripper)
 :precondition (and (visible ?b)
         (gripperOpen ?g)
 :effect (and (not (visible ?b))
         (gripperAroundBlockTop ?g ?b)))
```

The operator has been learned from demonstrating the reaching motion to the robot. The parameter block represents the typed variables `?b` and `?g` that are involved in the predicates. The types of objects are not learned but are provided by the perception system during the demonstrations. The terms in the precondition and effect blocks correspond to learned predicates. Here, we replaced the automatically generated names by meaningful ones.

Accounting for Physical Constraints

After implementing our approach, we identified that the robot misses background knowledge about its capabilities and the physical world. For example, the robot should not move away from a door if its gripper is still grasping the handle of the door. The robot simply cannot move the door although that might be a valid plan from the PDDL definition point of view. Such constraints could in theory be identified based on a physical simulation system that operates in parallel to the planner and verifies that a plan does not violate any physical constraints. However, such simulations are considerably expensive and complex. We therefore added a few additional constraints manually to the PDDL description. In particular: (i) The robot cannot move away from a door while grasping its handle. A similar rule needs to be added for any object that the teacher grasped during the demonstrations but that cannot be carried away. (ii) The robot is not allowed to release an object from its gripper without placing it somewhere, for example on a table. Otherwise, the object may break or the robot may not be able to pick it up again from the ground—this actually happened during our first experiments. (iii) The robot cannot reach any object that is further away than 70 cm from its torso without navigating first. This encodes the size of the workspace which is given by the size of the robot's arm.

Planning using the Acquired Actions

Given the collection of actions including the PDDL description, the planning problem can be outsourced to any standard symbolic planning system capable of interpreting PDDL. In our system, we use the fast downward planner proposed by (Helmert 2006). We used Helmert's implementation and integrated it into a ROS module.

To execute the next action of a computed plan, the robot has to choose one of the DMPs from its library that belongs to the corresponding action. The DMPs can be adjusted easily to situations that have a different starting or goal point compared to the learning phase. The DMP will generate a new trajectory whose shape resembles the demonstration but generalizes to the new situation. To only enforce minor adaptations due to a new start and goal point, our approach selects the DMP for which the relative pose of the end effector between the goal point g and start point s during training was most similar to the current situation. The relative pose is described by its translational $t(s, g)$ and rotational component $r(s, g)$. We select a DMP using the cosine similarity measure

$$d_t(s, g, i) = \frac{t(s, g) \cdot t(s_i, g_i)}{||t(s, g)||\ ||t(s_i, g_i)||}, \quad (7)$$

where s_i and g_i are the start and goal pose during the demonstration from which the i-th DMP has been learned. Similarly, $d_r(s, g, i)$ is defined for the rotational component and takes into consideration the angle and direction of rotation.

Additionally, we simulate the trajectory for the i-th DMP after setting the new start and goal to check for collisions between the end effector and obstacles. The term $d_o(s, g, i)$ is the minimum distance to the closest obstacle along the

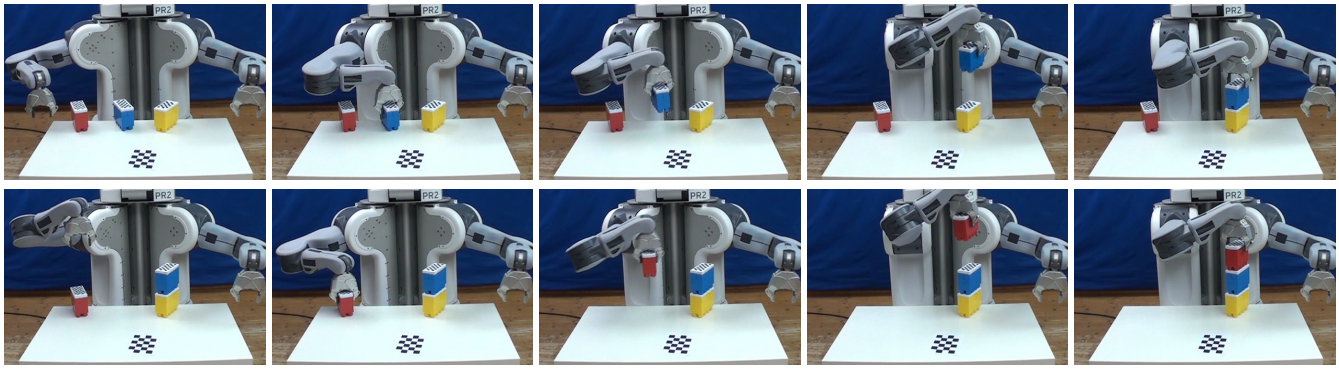

Figure 2: The robot builds a tower of three blocks. To do so, the robot only uses the basic actions that it has learned from demonstrations and combines them in a new way.

trajectory. Then, we choose the DMP with the index

$$i^* = \underset{i}{\operatorname{argmax}}\, \alpha_t d_t(s,g,i) + \alpha_r d_r(s,g,i) + \alpha_o d_o(s,g,i), \quad (8)$$

where α_t, α_r, and α_o are scaling coefficients chosen to reflect the relative importance of the different criteria. Finally, the chosen DMP is instantiated with s and g and executed.

Note that even when properly selecting appropriate DMPs, in the real world a robot may not carry out all actions as expected or the environment may change. Especially for long action sequences, it is unlikely that the individual steps can be executed without corrections. For example, if the gripper slips off the door handle, the robot should be able to detect that and compute a new plan. We therefore implemented a separate module that monitors the execution of the plan, computes the current values of the features, updates the values of the individual predicates, and compares the actual state to the expected effects of the actions. In case something unforeseen happens, the execution monitor triggers replanning using the current state of the world as the start state. The fast downward planner is efficient enough to compute a new plan online so that the robot can proceed without significant interruptions.

Experiments

The evaluation is intended to show the capabilities of our approach. All components of the system have been implemented as ROS modules and our experiments are carried out with a real PR2 robot. We considered tasks from two different manipulation domains: blocks world-like tasks like moving and stacking blocks as well as operating and opening doors. Videos covering the experiments can be found at: http://www.informatik.uni-freiburg.de/%7Estachnis/videos/tampra/

Training and Learning Preconditions and Effects

The first set of experiments is designed to illustrate how our system can learn individual actions from multiple demonstrations and is able to identify the preconditions and effects reliably. Teaching was done by kinesthetic training as illustrated in Fig. 1. We demonstrated 11 different actions to the robot and provided 10 demonstrations per action. Actions include reaching for objects and grasping them, placing an object on a target, turning a door handle, pushing a door, etc. In all our experiments, the DMPs were learned without any problems and stored in the robot's action library. Moreover, the correct set of preconditions and effects was identified by our system, *i.e.*, no necessary conditions were missing and all relevant ones were identified.

Table 1: Success rate for learning preconditions and effects.

#demonstrations	5	6	9	10	>10
success rate	17/20	19/20	19/20	20/20	20/20

For example, for the action `reachHandle`, the system correctly identified as preconditions that (a) the gripper has to be open and that (b) the handle must be visible. As effects, it identified that (a) the door handle is inside the gripper, (b) the gripper stays open, and (c) the handle is still visible. At the same time, the system correctly identified that all other features, like the relative pose of the gripper to the robot's torso and the exact distance of the door handle relative to the robot, are irrelevant.

To provide a more quantitative evaluation, we recorded 20 demonstrations for the action `reachBlock`. We then randomly sampled demonstrations, performed the learning step, and compared the extracted preconditions and effects to the real ones. We repeated this process 20 times and obtained the results shown in Tab. 1. When using 10 or more demonstrations, the system produced the correct results in all cases. With less than 10 demonstrations, the system often failed 1 to 3 out of 20 times in the sense that our approach found at least one false positive precondition or effect. This is due to too little variations in the feature values in the small number of demonstrations.

Building a Tower of Three Blocks

The second set of experiments is designed to show how the robot can use the learned actions to solve novel tasks, *i.e.*, tasks that have not been demonstrated to it beforehand. In this example, the robot was placed in front of a table with three

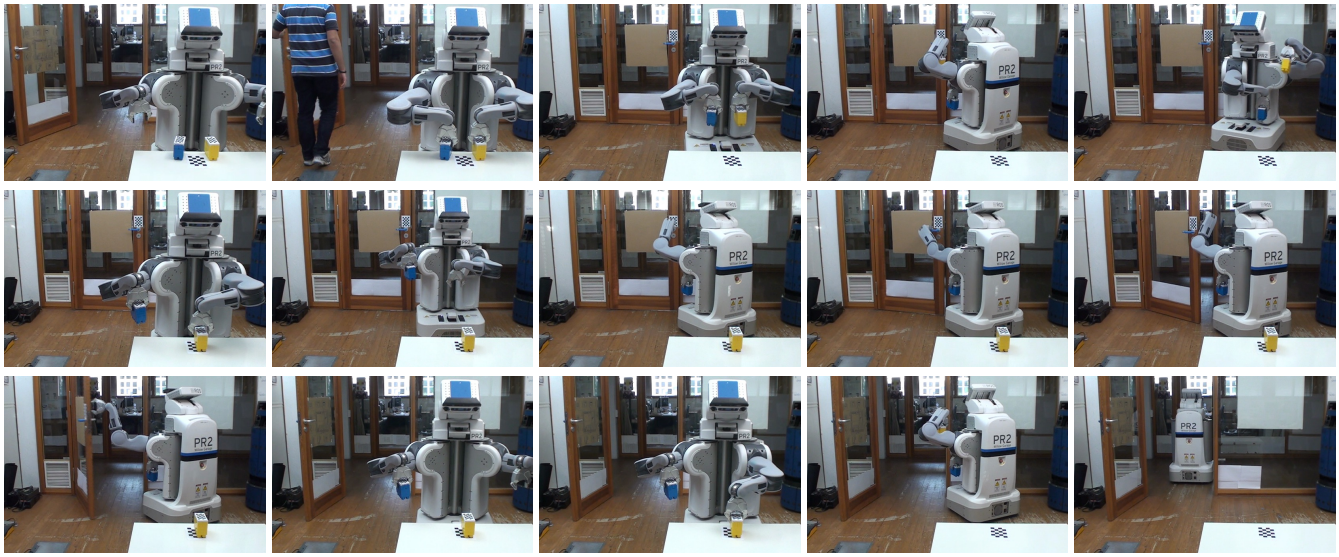

Figure 3: The robot computes a plan to grasp the two blocks and go through an open door. Once the robot has grasped the two blocks, a person closes the door. After having detected that, the robot computes a new plan to clear its left gripper by first going back to the table and placing the yellow block there. The robot then moves back to open the door, and then back to the table to grasp the yellow block again. Finally, the robot leaves the room with both blocks and reaches the goal state. See also http://www.informatik.uni-freiburg.de/%7Estachnis/videos/tampra/ for a video of this experiment.

blocks on top of it. As the goal configuration, the three blocks should be stacked on top of each other (yellow-blue-red). The planner computed a plan using the learned pick-and-place operators. This involves reaching, grasping, placing, and releasing blocks. Furthermore, the system correctly merged identical predicates. For example, the effect of grasping is equivalent to the precondition of placing. Moreover, each step was executed by choosing a DMP and adapting it to the new situation. Fig. 2 depicts the plan execution.

We repeated this experiment 20 times. In all cases, the robot was able to generate a valid plan to the goal. The only sources of failure that occurred during the execution were checkerboard markers not being detected by the perception system, or an error in estimating the pose of a block.

Reacting to Unexpected Changes in the Environment

This experiment is designed to illustrate how the robot can deal with unexpected changes in the environment while executing plans. The goal was to take two blocks and bring them to the corridor outside the room. Initially, both blocks lie on a table in the room and the door is open. While the robot picks up the two blocks with its manipulators, a person closes the door. After detecting that change, the robot computes a new plan and decides to bring one block back to the table to free one gripper. It then opens the door with the free hand, moves back to the table, picks up the block again, and finally brings both blocks outside the room. Pictures from this experiment are shown in Fig. 3.

Maintaining a Goal State

The last experiment illustrates how the robot can use the learned actions to plan for reaching a goal state from different possible starting states. We placed the robot in front of a door and instructed it to keep it fully open. Then, a human repeatedly closed the door and the robot opened it from basically any possible configuration.

During this experiment, the robot came up with three different plans depending on the current configuration of the door and the visibility of its handle. If the door is completely closed, our robot needs to carry out the following actions: `reachHandle; graspHandle; turnHandle; pullDoor; releaseHandle; moveArmToInnerSide; pushDoor`. If the door latch was not locked, it is sufficient to execute: `reachHandle; graspHandle; pullDoor; releaseHandle; moveArmToInnerSide; pushDoor`. If the door is already partially open but the robot does not see the handle, then executing: `moveArmToInnerSide; pushDoor` is sufficient. Some of these actions are visible in the second and third rows of Fig. 3 showing the previous experiment. Further images had to be omitted due to limited space but the reader may consider the video showing parts of this experiment. We conducted this experiment for more than 20 min without any critical failures. It may happen that the execution of an action fails but the execution monitor always detects that and compensates for it immediately by replanning.

Limitations

Despite these encouraging results, there is space for further improvements. Currently, our system is able to identify only a limited variety of patterns in the feature values to find the

preconditions and effects. To find more complex patterns, which are needed to symbolically represent more complex actions, more sophisticated pattern recognition algorithms than one-class k-means clustering are needed.

Our system furthermore assumes that preconditions and effects can be expressed based on a set of predefined features. It would be interesting to substantially extend the list of features available to the robot, such as features that capture physical aspects like forces and dynamics. As the number of features increases, we expect to require more teacher demonstrations. Alternatively, the robot could explore preconditions and effects in simulation to reject irrelevant features that resulted in learning false positive conditions. For instance, a robot that is taught in front of a table may regard the color of the table as an important aspect, although this is obviously not the case. Such a simulation could also allow the robot to explore physical constraints without having to manually provide them. Finally, our approach relies on several thresholds, which reflect the desired accuracy of performing the actions. These thresholds are currently set manually. Learning them should be considered.

Conclusion

We addressed the problem of learning a library of manipulation actions based on demonstrations provided by a teacher. Our approach requires only few demonstrations of actions and identifies the preconditions that need to be fulfilled for each action to be applicable, as well as the effects that are always fulfilled as a result of executing it. These conditions are represented by logical predicates, leading to a symbolic representation in the Planning Domain Definition Language. Therefore, the robot can use existing state-of-the-art planners to solve manipulation tasks which are in sum more complex compared to the taught actions. Furthermore, from the same demonstrations, the robot learns how to physically execute the actions by encoding the observed trajectories as dynamic movement primitives. We implemented our approach and presented experiments using a real PR2 robot to illustrate the capabilities and flexibility of our system, including its ability to react to unexpected changes in the environment.

Acknowledgments

This work has partly been supported by the EC under grant FP7-ICT-248258-First-MM. We thank Malte Helmert for providing his implementation of the FD planner, Peter Pastor for making his DMP implementation available, and Luciano Spinello for the fruitful discussions about single-class SVMs.

References

Asfour, T.; Gyarfas, F.; Azad, P.; and Dillmann, R. 2006. Imitation learning of dual-arm manipulation tasks in humanoid robots. In *Int. Conf. on Humanoid Robots*.

Bentivegna, D.; Atkeson, C.; Ude, A.; and Cheng, G. 2004. Learning to act from observation and practice. *Int. Journal of Humanoid Robotics*.

Billard, A.; Calinon, S.; Dillmann, R.; and Schaal, S. 2008. Robot programming by demonstration. In Siciliano, B., and Khatib, O., eds., *Handbook of Robotics*. Springer.

Boiman, O.; Shechtman, E.; and Irani, M. 2008. In defense of nearest-neighbor based image classification. In *IEEE Conf. on Computer Vision and Pattern Recognition*.

Calinon, S., and Billard, A. 2008. A probabilistic programming by demonstration framework handling skill constraints in joint space and task space. In *Int. Conf. on Intelligent Robots and Systems*.

Ekvall, S., and Kragic, D. 2006. Learning task models from multiple human demonstrations. In *Intl. Symposium on Robot and Human Interactive Communication*, 358–363.

Eppner, C.; Sturm, J.; Bennewitz, M.; Stachniss, C.; and Burgard, W. 2009. Imitation learning with generalized task descriptions. In *Int. Conf. on Robotics & Automation*.

Helmert, M. 2006. The fast downward planning system. *Journal on AI Research* 26.

Kennedy, K.; Namee, B. M.; and Delany, S. 2009. Learning without default: A study of one-class classification and the low-default portfolio problem. In *Conf. on Artificial Intelligence and Cognitive Science*.

Krüger, N.; Piater, J.; Wörgötter, F.; Geib, C.; Petrick, R.; Steedman, M.; Ude, A.; Asfour, T.; Kraft, D.; Omrcen, D.; Hommel, B.; Agostino, A.; Kragic, D.; Eklundh, J.; Krüger, V.; and Dillmann, R. 2009. Formal definition of object action complexes and examples at different levels of the process hierarchy. Technical report.

Mardia, K.; Kent, J.; and Bibby, J. 1979. *Multivariate Analysis*. Academic press.

Omrcen, D.; Ude, A.; and Kos, A. 2008. Learning primitive actions through object exploration. In *Proc. of the Int. Conf. Humanoid Robots*.

Pardowitz, M.; Zöllner, R.; and Dillmann, R. 2006. Incremental acquisition of task knowledge applying heuristic relevance estimation. In *Int. Conf. on Robotics & Automation*.

Pastor, P.; Hoffmann, H.; Asfour, T.; and Schaal, S. 2009. Learning and generalization of motor skills by learning from demonstration. In *Int. Conf. on Robotics & Automation*.

Petrick, R.; Kraft, D.; Mourão, K.; Pugeault, N.; Krüger, N.; and Steedman, M. 2008. Representation and integration: Combining robot control, high-level planning, and action learning. In *Int. Cognitive Robotics Workshop*.

Schölkopf, B., and Smola, A. 2002. *Learning with Kernels*. MIT Press.

Schölkopf, B.; Platt, J.; Shawe-Taylor, J.; Smola, A.; and Williamson, R. 2000. Estimating the support of a high-dimensional distribution. Technical report, Microsoft Research, TR87.

Veeraraghavan, H., and Veloso, M. 2008. Teaching sequential tasks with repetition through demonstration. In *Int. Conf. on Autonomous Agents and Multiagent Systems*.

Zhuo, H. H.; Hu, D. H.; Hogg, C.; Yang, Q.; and Munoz-Avila, H. 2009. Learning HTN method preconditions and action models from partial observations. In *Int. Conf. on Artificial Intelligence*.