Manipulation Planning Using Object-Centered Predicates and Hierarchical Decomposition of Contextual Actions

Alejandro Agostini, Matteo Saveriano, Dongheui Lee, and Justus Piater

***Abstract*—Current approaches combining task and motion planning require intensive geometric and symbolic reasoning to find feasible motions for task execution. The poor expressiveness of task planning domains for characterizing geometric changes with actions and the difficulties faced by current approaches to efficiently identify motion dependencies for plan execution produce expensive callings to motion planning on unfeasible actions and intensive reasoning to find realizable plans. In this work we combine two recent approaches to address these problems. Task planning is carried out using an object-centered description of geometric relations that consistently characterizes changes in the object configuration space. Plan execution is implemented using a symbol to motion hierarchical decomposition that depends on consecutive actions in the plan, rather than on single actions, which permits considering motion dependencies across plan actions for a successful execution.**

***Index Terms*—AI-Based methods, cognitive control architectures, manipulation planning.**

I. Introduction

TASK planning [1] is an efficient tool for automatically defining the sequence of instructions to a robot for the execution of manipulation tasks. It permits representing structures of the environment that are relevant to describe object configurations and actions that can be performed on them in an intuitive manner, using a declarative notation compatible with human language: `on cup table`, to indicate that a cup is on the table, or `pick cup table`, to instruct the action of picking the cup from the table. The sequence of instructions to fulfill a task, the task plan, is generated using searching strategies that evaluate the changes produced by action executions encoded in the so called planning operators. For the execution of the task, it is mandatory to ground the symbolic actions to let the robot physically interact with the real world [2]–[6]. This is normally done by integrating methods of different levels of abstraction into a task and motion planning (TAMP) framework [7]–[10], where task and motion planning are brought together through geometric reasoning mechanisms that search for feasible robotic motions for the execution of task plans. However, this strategy is not cost-effective since it requires intensive computations and several calls to motion planning on unfeasible actions to search for solutions in the usually large object configuration space.

We propose a TAMP framework that generates feasible motions for task plan execution without intensive geometric reasoning or unfruitful callings to motion planning. The framework combines, on the one hand, a task planning approach that uses a novel representation of geometric constraints in the object configuration space for the generation of geometrically consistent plans and, on the other hand, a method for grounding symbolic actions based on a hierarchical decomposition of abstract tasks into specific motions. The selection of which hierarchical decomposition is most suitable for the execution of a symbolic action is determined by the geometric constraints of consecutive plan actions encoded at the task planning level.

A. Related Works

Several strategies have been proposed to alleviate the computational effort of combined task and motion planning (TAMP). A widely used approach is to hierarchically decompose a complex abstract task into several simple sub-tasks that can be easily solved and executed [11]–[14]. Along this line, Kaelbling *et al.* [12] interleave hierarchical planning with plan execution on small sub-tasks to limit the reasoning effort. The approach generates a global plan using only highly abstract tasks and without checking in detail the forward progression of the effects of actions. Given this global plan, each of the involved tasks is resolved as a smaller TAMP problem that is immediately executed. The resulting state is used to initialize the next task in the global plan, repeating the process. In this manner, the approach focuses on the execution of the task at hand but at the expense of facing frequent planning impasses. Our approach also relies on a hierarchical decomposition of plan actions. However, contrary to [12], our task planner consistently assesses

Manuscript received February 24, 2020; accepted June 22, 2020. Date of publication July 14, 2020; date of current version July 24, 2020. This letter was recommended for publication by Associate Editor J. Kober and Editor T. Asfour upon evaluation of the reviewers' comments. This work was supported by the Austrian Science Fund (FWF) Project M2659-N38 and by the Helmholtz Association. *(Corresponding author: Alejandro Agostini.)*

Alejandro Agostini is with the Department of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria, and also with the Chair of Human-centered Assistive Robotics, Technical University of Munich, Munich 80333, Germany (e-mail: alejandro.agostini@uibk.ac.at).

Matteo Saveriano and Justus Piater are with the Department of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria, and also with the Digital Science Center, University of Innsbruck, 6020 Innsbruck, Austria (e-mail: matteo.saveriano@uibk.ac.at; justus.piater@uibk.ac.at).

Dongheui Lee is with the Chair of Human-centered Assistive Robotics, Technical University of Munich, Munich 80333, Germany, and also with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), 82234 Wessling, Germany (e-mail: dhlee@tum.de).

Digital Object Identifier 10.1109/LRA.2020.3009063

the propagation of effects of actions in the plan using a rich representation of geometric changes.

The framework presented in [15] integrates a hierarchical decomposition of tasks, in the form of grammar models, and haptic predictions for complex manipulation on single objects. Grammar models are encoded in a graph representation that contains words (single actions) or sentences (sequences) at terminal nodes. These graphs are used for task plan generation according to the history of actions rather than from propagating the effect of actions in the reasoning process. The approach is specially suitable for complex manipulation of single objects involving action selection based on forces but does not address the general TAMP problem of efficiently defining motion parameters on configuration spaces comprising several objects.

Another approach to tackle the TAMP problem is to use a semantic representation of geometric constraints to better interface continuous motion parameters and symbolic task descriptions [16], [17]. Wells *et al.* [16] propose to train a support vector machine classifier that uses semantics of geometric constrains to quickly classify motions as feasible or not feasible. The classifier has a relatively low accuracy provided the coarse granularity of semantic representations but helps reducing the effort of motion exploration, which is still required to define the motion parameters to ground symbolic actions. The approach in [17] also incorporates semantics descriptions of geometrical constraints to evaluate motion feasibility of single actions. However, contrary to [16], the constraints are used within the task planning algorithm, rather than in separate methods. The task planner generates candidate plans adding and removing constraints incrementally while a sampling-based motion planner checks actions feasibility using geometric reasoning. The described approaches focus on grounding single plan actions without considering motion dependencies between actions in the plan and require the introduction of additional methods to handle the semantic representations. Our approach, instead, permits generating motions compatible with consecutive actions in the plan, rather than with single actions, and avoids the necessity of defining intermediate semantics or heuristics.

Logic programming is an appealing alternative to task planning based on state-space search. Logic programming methods search for solutions directly in the plan space, rather than in the state space, which permits better considering geometrical constraints compatible with entire plan executions [18], [19]. Lagriffoul *et al.* [18] use logic programming to find plans compatible with symbolic constraints as explanations of plan failures produced by collisions. Toussaint [19], in turn, proposes an approach specially designed for creating pile of objects with stable configurations, where symbols are tailored to describe geometric and differential constrains, e.g. (in-)equalities, for optimizing the entire plan execution. Logic programming approaches need a model of the robot dynamics to find optimal solutions after intensive computations. Our approach, instead, does not require the robot dynamics and is able to generate plans at low computational costs using off-the-shelf linear planners. Although we consider motion dependencies between consecutive actions rather than in the entire plan, as in logic programming, it provides an appealing low-complexity alternative to these methods.

In this work we adapt and combine two methods to address the efficiency problem of TAMP frameworks: 1) the generation of geometrically consistent task plans based on an object-centered representation of geometrical relations [20], and, 2) the transformation of abstract tasks into motion parameters using a symbol-signal hierarchical decomposition [21]. The main contributions of this work can be summarized as follows:

- The integration into a TAMP framework of a planning domain compatible with off-the-shelf, computationally efficient, linear planners that permits considering relevant geometrical constraints for plan execution already at the task planning level.
- The integration of task planning with a hierarchical symbol-signal decomposition of actions that combines learning from demonstration and action segmentation for plan action execution.
- A method for selecting adequate motion parameters considering motion dependencies on consecutive actions in a task plan without the need of intensive geometric reasoning or multiple callings to motion planning.

The rest of the paper is organized as follows. Section II describes the basic elements of our framework, namely the task planner and the hierarchical task representation. In Sec. III, we describe how planning and hierarchical task decomposition are effectively combined to execute manipulation tasks. Experiments on a real robot are presented in Sec. IV. Section V states the conclusion and propose further extensions.

II. Preliminaries

A. Task Planning

We use the traditional task planning domain definition comprising a set of objects (e.g. `cup`, `table`) and a set of predicates, coding object relations and properties (e.g. `on cup table`), which are logical functions that take value `true` or `false`. The set of predicates describing a particular scenario defines the *symbolic* state s. We define a set of planning operators (PO), encoded in the traditional precondition-action-effect notation [22]. A PO describes the changes on a symbolic state with an action execution. The precondition part comprises the predicates that will be changed by the execution of the PO, as well as those predicates that, even though they don't change with the execution, are necessary for these changes to occur. The effect part describes the changes in the symbolic state after the PO execution. We define a *symbolic action* as the name of the PO that consists of a declarative description of an action and may contain parameters to ground the predicates in the precondition and effect parts. In task planning, the planner receives the description of the *initial state*, s_{ini}, and a *goal* description, g, as a set of grounded predicates that should be observed after task execution. With these elements, the planner searches for a sequence of actions called *plan* that would permit producing changes in s_{ini} necessary to obtain the goal g using the set of planning operators [1]. In this work, we use the off-the-shelf linear planner Fast Downward [23].

For the generation of realizable plans, it is important to encode in the planning domain task-relevant geometrical descriptions

TABLE I
EXAMPLE PLANNING OPERATORS

```
(:action pick-top                 (:action place-top
:parameters                       :parameters
(?obj1 ?obj2)                     (?obj1 ?obj2)
:precondition (and                :precondition (and
(on ?obj1 air)                    (on ?obj1 hand)
(under ?obj1 ?obj2)               (under ?obj1 air)
(on ?obj2 ?obj1)                  (on ?obj2 air)
(in hand air)                     (in hand ?obj1)
:effect (and                      :effect (and
(on ?obj1 hand)                   (on ?obj1 air)
(under ?obj1 air)                 (under ?obj1 ?obj2)
(on ?obj2 air)                    (on ?obj2 ?obj1)
(in hand ?obj1)                   (in hand air)
(not (on ?obj1 air))              (not (on ?obj1 hand))
(not (under ?obj1 ?obj2))         (not (under ?obj1 air))
(not (on ?obj2 ?obj1))            (not (on ?obj2 air))
(not (in hand air))))             (not (in hand ?obj1))))
```

that permit consistently characterizing changes in the object configuration space. To this end, we define a set of predicates that can be unambiguously obtained from object parameters describing the object configuration space. Using the poses and bounding boxes of objects, we can identify six sides of the bounding boxes of each objects: top, bottom, front, back, left and right. We use the sides to describe the relation of each object with others through simple relational predicates: `on o1 o2` (object `o2` is on object `o1`), `under o1 o2` (`o2` is under `o1`), `in o1 o2` (`o2` is inside `o1`), and so on.

These relational predicates have been widely used in the TAMP community. However, the newly introduced concept here is that these relations are described from an *object perspective*, rather than from an observer perspective. Using an object perspective permits describing the object configuration space only with simple relational predicates, without the need of introducing additional arbitrary names. For example, the predicate `on cup table` would describe that the top of the cup is touching the table. If, instead, the bottom of the cup is touching the table, the predicate `under cup table` would be used. Instead, the interpretation of an external observer of the predicate `on cup table` is that the cup is on the table, no matter if it is lying upright, horizontally, or upsidedown. To fully describe the cup-table configuration, it would be necessary to introduce additional arbitrary symbols such as `isOriented`, `upright`, `upsidedown` [9]. This difference becomes important when the task requires to distinguish object relations with different orientations in configuration spaces involving several objects. An exhaustive analysis of the benefits of this representation is presented in [20].

Table I presents two example planning operators for picking and placing an object encoded using the object-centered predicates. These predicates permits characterizing important geometric conditions for the execution of these operators. For instance, for a picking from top action, the conditions that the hand is empty and that the top of the object is clear for grasping are encoded as `in hand air` and `on ?obj1 air`, respectively, where `air` is an abstract object indicating that no object is in contact with the corresponding side. For the placing action, in turn, the conditions `under ?obj1 air` and `on ?obj2 air` permit checking that the surfaces of the two objects that will get in contact are not obstructed.

After a plan is generated, we need to define the mechanisms for plan execution. This could be done using different strategies, ranging from predefined behaviours (e.g. control trajectories) to more elaborate methods involving geometric reasoning and motion planning. In this work, we ground plan actions using a hierarchical symbol-signal decomposition.

B. Hierarchical Task Decomposition

In [21], we decompose symbolic tasks into rooted trees, where each node corresponds to a certain robot behavior activated when a set of pre-conditions are met. In this representation, here referred as *schema*, the task planning reduces to simple three transversal and logical condition checking. The correct execution of a behavior modifies the value of the assigned post-conditions, regulating the execution. This is computationally effective but rather rigid, i.e. a schema cannot handle significant variations in the execution context.

We exploit an augmented version of a typical schema where each node is associated to a particular robot behavior and it is defined by the 5-tuple $\mathcal{B} = (l_b, r_b, p_b, c_b, e_b)$, where l_b is a unique label, r_b are the pre-conditions or releasers, p_b are the post-conditions, c_b are the child nodes, and e_b is a continuous *emphasis* parameter. The emphasis conveys in the tree information about the task execution coming from the sensors, allowing for a rapid adaptation of the task execution and helping to solve possible conflicts generated by multiple behaviors active at the same time. In this work, we continuously monitor the object poses to check the presence of the target object and plan motion trajectories in object frame. All the existing schemata are stored into a knowledge base in the form

```
schema(node_name(Obj),
    ⟨(child_node_1(Obj), pre_conditions_1),
    ( ··· ),
     (child_node_j(Obj), pre_conditions_j)⟩,
     node_post_conditions)
  schema(child_node_1(Obj),⟨ ⟩,
    post_conditions_1)
```

At run-time, we use the unique label of the root node to query for a specific schema. The task tree is dynamically instantiated and periodically traversed to monitor the task execution. Abstract schemata are grounded into concrete robot motions via kinesthetic teaching. Human demonstrations are automatically segmented using two simple rules, namely a new segment is generated if *i)* the robot enters/leaves the surveillance area (a sphere of radius 0.2 m) of an object or if *ii)* the user commands to open/close the gripper. In this phase, the emphasis is used to assign the generated segments to the most emphasize node, corresponding to the active behavior with `Obj = closest_obj`. In this way, knowledge is incrementally added by providing new demonstrations.

The outlined approach has the following limitations:

L1 The segmentation strategy generates unnecessary segments if the robot accidentally enters the proximity area of non-target objects.

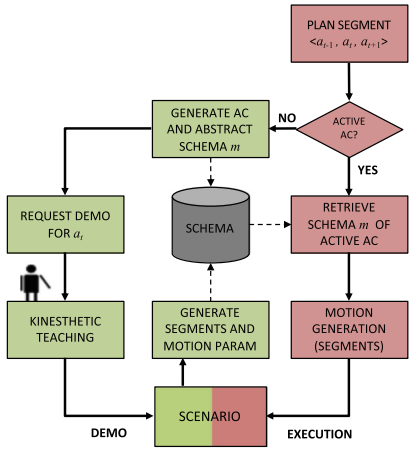


Fig. 1. General diagram representing the learning (green) and execution (red) mechanisms to ground plan actions using schemata. For the grounding of the plan action a_t, the system looks for an active action context in the set $\mathcal{AC}$ for the plan segment (a_{t-1}, a_t, a_{t+1}) (Sec. III-A). If no active action context is found, the system generates a new AC for that plan segment and an abstract schema (without motion parameters) associated to m. Afterwards, the system requests for a demonstration of the action a_t, which is performed using kinesthetic teaching. The demonstrated motion is segmented and the motion parameters for each segment are generated (Sec. III-B). If, on the contrary, an action context gets activated with (a_{t-1}, a_t, a_{t+1}) the associated schema m is retrieved from the Schema database and executed (Sec. III-C). Queries to the schema database are indicated with dashed lines.

L2 Only the bottom level of the tree is learned from demonstration, while higher levels are defined by a domain expert.

L3 Although the emphasis introduces some flexibility, a tree remains a relatively rigid structure that cannot consider significant variations in the executive context.

In this work, such limitations are overcome by combining the schema with a task planner as detailed in Sec. III.

III. TAMP Using Hierarchical Decomposition of Contextual Actions

In this section, we present the strategy to bring together the task planning approach based on object-centered geometrical descriptions (Sec. II-A) with the hierarchical decomposition of tasks using schemata (Sec. II-B). These two approaches are articulated through a new representation that we denote *action context* (AC). An AC is a tuple that represents consecutive actions in a plan and will play a fundamental role in learning schemata encoding feasible motions for task plan execution. This section presents the insights of such mechanisms, which are summarized in Fig. 1.

A. Action Context

We define an *action context* (AC) as a 4-tuple $ac = \{a_{\text{pre}}, a_{\text{now}}, a_{\text{post}}, m\}$, where a_{pre}, a_{now}, and a_{post} represent symbolic actions with grounded arguments (see Sec. II-A), and m represents an action grounding mechanism. We use the notation $\mathcal{AC}$ to refer to the set of action contexts. Given a task plan $p = \{a_0, a_1, \ldots, a_{t-1}, a_t, a_{t+1}, \ldots, a_n\}$, we say that an action context is *active* at time step t, if $a_t = a_{\text{now}}$, and the previous and posterior actions in the plan fulfill $a_{t-1} = a_{\text{pre}}$ and $a_{t+1} = a_{\text{post}}$, respectively. We refer to this active action context as ac_t^p. Active action contexts are used to execute action a_t in a plan p through the action grounding mechanisms $m \in ac_t^p$. Note that, in the general case, action contexts can be associated to different mechanisms for grounding symbolic actions: e.g. a plain set of dynamic movement primitive parameters [24] or a hierarchical task decomposition. In this work, m identifies the schema used to execute the action context.

Action context is a newly introduced concept that plays an important role for the grounding of symbolic actions. For example, if the action of picking a bottle (a_t) is followed by the action pouring (a_{t+1}), the motion performed should permit a posterior stable pouring. These specific movements for picking the bottle might be different if the next action is just to place the bottle somewhere else. In the same manner, previous actions also matter for defining adequate motions. The picking for pouring described before may involve different motions depending on if the robot picks the bottle after placing an object to the right or to the left of the bottle. An example of this situation is shown in Fig. 8. These different motions will be encoded in different schemata considering the adequate motion parameters. In general, associating symbolic actions to schemata through action contexts permits defining geometric parameters for motion planning depending on the action intentions and on the ongoing task. This allows for the generation of feasible trajectories between consecutive symbolic actions.

B. Learning From Human Demonstrations

To make our framework suitable for online planning and execution in variable scenarios, we define learning mechanisms that automatically generate action contexts and the associated schemata every time a new plan segment is observed. These mechanisms correspond to the green modules in Fig. 1. Given a task plan p, the context of the action at the current time t, a_{t-1}, a_t, a_{t+1}, is used to retrieve the active action context ac_t^p from the action context set $\mathcal{AC}$. If no action context is found, which indicates that the given plan segment was never observed before, a new AC is generated. The newly generated AC is stored in $\mathcal{AC}$ and triggers an instance of schema learning, where a human demonstration is requested to execute the action plan according to its context (e.g. pick the bottle from the table to pour water in the cup). The demonstration is used to generate a new schema that is stored in the schema database, associated to the newly generated AC through m. More in details, the new AC is used to instantiate an abstract schema (see Fig. 2 left), where the target object is specified by the planner. The schema has always a `TRUE` pre-condition, assuming the relevant preconditions for a successful execution of the schema were already checked at the task planning level. For instance, the AC activated for the action `pick side bottle table` in a plan segment `place top cup table - pick side bottle table - pour water bottle cup` is generated only if there is a bottle on the table and no object has been previously grasped. The post-condition of the schema also comes from the planner and let the system switch to the next active AC after the correct execution of the current schema. Therefore, the prior

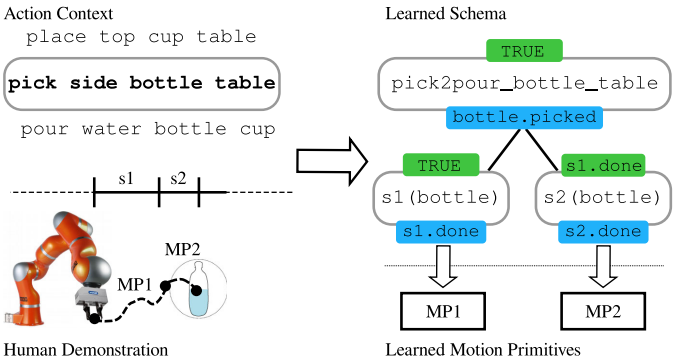


Fig. 2. Grounding of the AC into a schema with associated movement primitives. At run-time, object poses are used to adapt the motion execution.

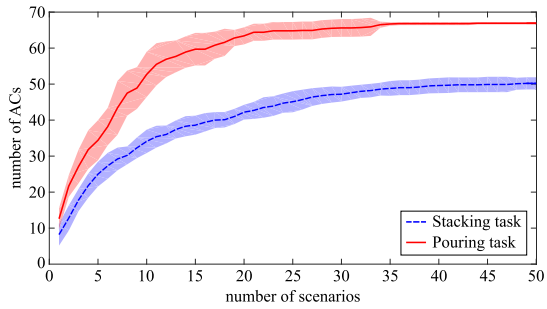


Fig. 3. Accumulated number of ACs for the stacking (blue) and pouring (red) tasks. The results present the average and standard deviation of 10 runs, each of them comprising 50 randomly generated initial states.

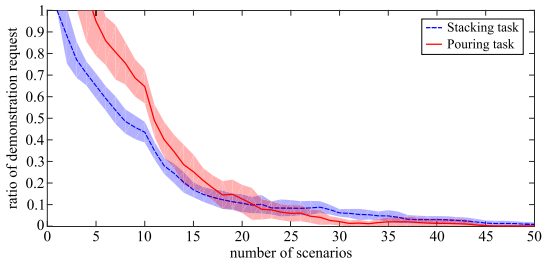


Fig. 4. Ratio of demonstration requests for plans generated in sequence for the stacking (blue) and pouring (red) tasks.

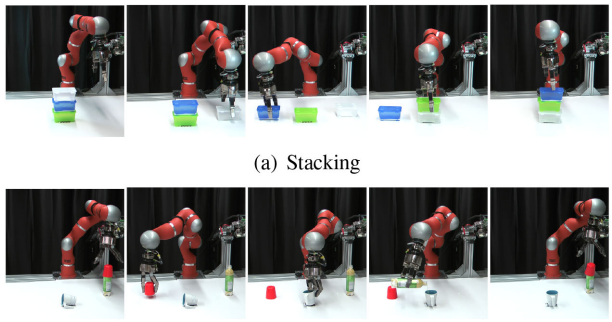


Fig. 5. Snapshots of the executions of the box stacking and the pouring tasks for the plans presented in Table II.

knowledge needed to instantiate new schemata comes from the task planner and not from an expert user as in [21] (limitation L2 in Sec. II-B).

The new abstract schema is grounded into a concrete robot behavior using human demonstrations. The user kinesthetically guides the robot to show the AC execution (e.g. pick a bottle as in Fig. 2). The demonstration is automatically segmented using robot to target object distance and gripper commands as described in Sec. II-B. The fact that a single target object is considered for the abstract schema makes the segmentation strategy more robust, resolving the limitation L1. Indeed, even if multiple objects are present, we need to monitor only the target object and trigger new segments when the robot reaches it. Considering the example in Fig. 2, the first segment is generated when the robot enters the surveillance area of the bottle independently from the other objects in the scene.

The generated segments are linked to the abstract schema. Since the segments are sequentially demonstrated, it is reasonable to assume that they are sequentially executed. This behavior is obtained by properly defining pre- and post-conditions that are automatically assigned to the segment nodes. After the demonstration the schema has `J` segments (leaves), indicated as `s1(Obj)`, $\cdots$, `sJ(Obj)`, where a segment `sj(Obj)` has been demonstrated after `sj-1(Obj)`. The post-condition of each segment `sj(Obj)` is `sj(Obj).done` $\forall$ `j=1`,\ldots, `J`. The first segment (`s1(Obj)`) has a `TRUE` pre-condition (like `s1(bottle)` in Fig. 2) and it is then executed first. The second segment (`s2(Obj)`) has `s1(Obj).done` as pre-condition and it is executed after `s1(Obj)`, and so on until the last segment `sJ(Obj)` is reached. `sJ(Obj)` sets the post-condition of the schema, successfully terminating the execution of the action context and returning to the planner that generates the next action context.

The generated segments are also uniquely associated to motion primitives used to generate motor commands for the robot. Poses collected during the demonstrations are used to generate these motion primitives as stable dynamical systems. Pick and place motions require simple point-to-point motions which are effectively represented by linear dynamical system connecting current and goal poses. Other action contexts, like pouring, require more sophisticated movements that generated using the dynamic movement primitives (DMPs) framework [24]. Initial and goal poses, as well as the robot trajectory, are automatically extracted from the demonstration and used to fit the DMP. A known problem of DMPs is the trajectory overshooting when generalizing to different initial/goal pose. To prevent the overshooting, in [21] the robot first reaches the surveillance area of the target object with a linear motion and then uses the DMP. This simple but effective strategy is used also in this work.

C. Autonomous Execution

In case an active action context is found in $\mathcal{AC}$ associated to the observed plan segment, the execution mechanisms identified by m are triggered (red modules in Fig. 1). First, the corresponding schema is retrieved from the schema database. Then, the schema

is executed segment by segment using the associated motion primitives—either DMP or linear system—and the current pose of the target object for trajectory generation. Motion trajectories are generated relative to the object pose and on-line transformed into the robot base frame using the forward kinematics. In this way, the generated motions adapt to changes between the demonstrated and the actual execution context. At run-time, the schema is periodically traversed to determine the active leaf, i.e. the next segment that the robot has to execute.

The emphasis parameter, that in this work is the inverse of the robot-object distance squashed between 0 and 1, is also periodically updated. In case the object is removed from the scene or moved to an unreachable position, the schema execution is preempted and its post-condition left unchanged. This prevents the robot to execute useless and potentially dangerous movements. The planner is informed of the failure through the unchanged post-condition, and it can generate new action contexts to recover the task. It is worth noticing that the hierarchical structure described in Sec. II-B has limited re-planning capabilities (see the limitation L3 in Sec. II-B) and that the proposed combination of task planning and schemata contributes to mitigate this limitation. If the schema ends successfully, the post-condition(s) is set and the planner proceeds with the next action context. Finally, the object pose, periodically monitored to update the emphasis, is used to adjust the motion in case of unexpected perturbations. An example of this behavior is shown in Sec. IV.

TABLE II
EXAMPLE PLANS FOR THE STACKING AND POURING TASKS

Stacking	Pouring
pk top whiteB blueB (pick2place)	pk bottom redC bottle (pick2place)
pl top whiteB tabler (placeon)	pl bottom redC tablel (placeon)
pk top blueB greenB (pick2place)	pk top whiteC tablem (pick2rotate)
pl top blueB tablel (placeon)	pl top whiteC tablem (placerotated)
pk top greenB tablem (pick2place)	pk side bottle tabler (pick2pour)
pl top greenB blueB (placeon)	pour water whiteC bottle (pour)
pk top whiteB tabler (pick2place)	pl side bottle tabler (placeon)
pl top whiteB tablem (placeon)	pk bottom redC tablel (pick2place)
pk top greenB blueB (pick2place)	pl bottom redC bottle (placeon)
pl top greenB whiteB (placeon)	
pk top blueB tablel (pick2place)	
pl top blueB greenB (placeon)	

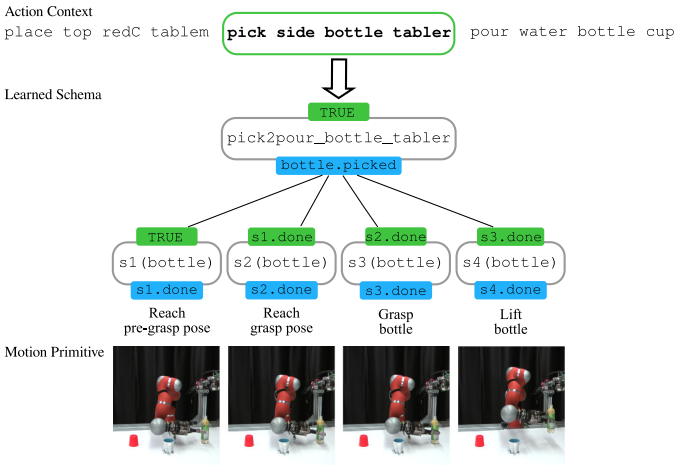


Fig. 6. Action context with associated schema for a pick2pour action.

IV. EXPERIMENTS

We evaluate the effectiveness of our approach with a set of manipulation experiments where a real 7 degrees-of-freedom robot (Kuka LWR IV) is asked to solve different pouring or stacking tasks. These tasks require several planning steps and the execution of complex manipulation actions. In all the experiments, the schemata are generated from scratch using the learning mechanisms presented in Sec. III-B. The evaluation of predicates describing the object configuration space is carried out as described in Sec. II-A.

The scenario for the **stacking task**, illustrated in Fig. 5, comprises 3 boxes, namely the white (`whiteB`), blue (`blueB`), and green (`greenB`) boxes. In addition to `whiteB`, `blueB`, and `greenB`, we define the objects `tablel`, `tablem`, and `tabler` to indicate the left, middle, and right parts of the table. The table is considered as composed of 3 parts to facilitate consistency checking for placing actions. The goal for this task is to arrange the boxes in the configuration `tablem-white-green-blue`. For the **pouring task** (see Fig. 5) we consider a different set of objects to be manipulated: a bottle `bottle`, a white cup `whiteC`, and a red cup `redC`. The goal for this task is to place the white cup on table middle and fill it with water, while the bottle should be placed on table right and covered with the red cup. For plan generation, we use the Fast Downward planner [23]. Initial states are variable and defined according to the purpose of each experiment.

Number of action contexts and demonstration requests: We carry out 10 different runs for each of the tasks (stacking and pouring). Each run consists of solving 50 different planning problems that are presented to the system in sequence, where the initial state for each of them is defined from a random initial configuration of objects. No schemata or action contexts are initially provided. For each of the experiments, we compute the accumulated number of action contexts and the ratio of actions that triggered a demonstration request, calculated as the total number of requests versus the length of the plan. For the generation of action contexts in the stacking task, we do not consider the color of the boxes to favour generalization.

Fig. 3 and 4 present the average and standard deviation of the 10 runs. We can observe that most of the action contexts are generated during the initial 20 scenarios, where the ratio of demonstration drops below 10 % after this point. The system quickly becomes fully autonomous, executing plans without the need of further demonstrations. The total average number of ACs generated for the pouring and stacking tasks were 66 and 48, respectively. Table II shows two example plans generated for stacking and pouring tasks, where we mention the intention of each action in the context of the plan to provide an intuition of motion dependencies. Snapshots of the plans execution are shown in Fig. 5.

To shed light on the specific processes that link ACs with schemata, we present in Fig. 6 a concrete example of a schema associated to the AC `place top redC tablem - pick side bottle tabler - pour water bottle cup`. If this schema is selected for execution, it is instantiated and executed segment by segment as discussed in Sec. III-C. If the query returns an empty schema (the schema has not been

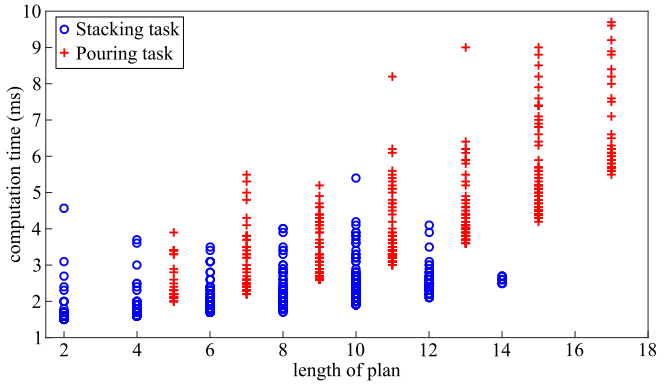


Fig. 7. Computation time for the plans generated in the experiments of Fig. 3 and 4 for the stacking (blue) and pouring (red) tasks.

generated so far), the learning from demonstration mechanism described in Sec. III-B is triggered. After demonstration, the learned schema with associated motion primitives is stored in the database and the task execution continues from the next AC. This interactive learning and execution mechanism allows us to reuse existing schemata in different tasks and to incrementally add new schemata when needed for the task execution.

Computation time and scalability: To assess the scalability of our approach to different complexity problems, we present in Fig. 7 the computation time for different plan lengths corresponding to all the plans generated in the experiments of Fig. 3 and 4. We can see that the most demanding planning problem, comprising 17 steps and corresponding to the pouring task (red crosses), does not exceed a computation time of 10 ms. It is worth noticing that we have similar computation times when planning with single actions.

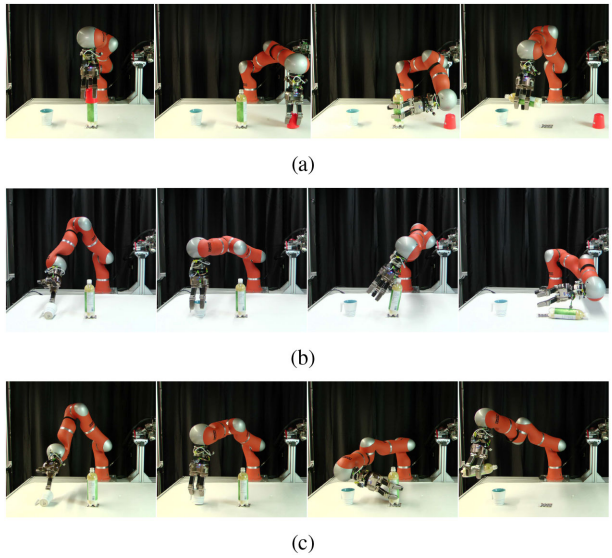
The schema is a reactive system that generates and eventually re-plan the motion trajectory at each time step. However, to provide a reference of the computation effort required for grounding a plan action, we measure the total computation time for grounding an AC into a schema. Grounding the AC into a schema requires to: query the schema from a database (≈ 5.7 ms independently of the schema), traverse the schema tree to determine the motion primitive to execute (≈ 10 ms independently of the schema)[1], load the motion primitive, determine the goal pose from the current object pose, and generate the entire robot trajectory (≈ 10.6 ms with DMP, ≈ 5.1 ms with a linear dynamical system).

Single actions vs. action contexts: To assess the validity of our framework to select feasible motions compatible with consecutive actions in the plan, we perform another set of real-robot experiments where the schema to be executed is selected only depending on the current action in the plan, without considering what action was executed before and what action comes next. We refer to these experiments as Single Action (SA) experiments. The results of the SA experiments are contrasted with those obtained using action contexts (ACs). The experiments comprises

[1]A small schema like that in Fig. 6 can be traversed in less than 1 ms. However, high loop frequencies make the communication with the robot over ROS topics unreliable.

TABLE III
RESULTS FOR THE POURING TASK USING SINGLE ACTIONS (SA) AND ACTION CONTEXTS (AC)

Planning Approach	Total Demos	Total Failures	Success Rate
SA	25	15	0.5
AC	66	0	1



(a)

(b)

(c)

Fig. 8. Snapshots of two example scenarios for the pouring task. (a) Both AC and SA generate feasible plans. (b)–(c) In the same situation, the plan generated with SA is unfeasible (b) while AC generates a feasible plan (c).

30 consecutive pouring tasks with random initial configuration of objects. The results are presented in Table III. We compute the total number of failed executions and the rate of successful plans, i.e. presenting no execution failures. The single action approach (SA) produced 15 execution failures (see Fig. 8 for a failure example), where 50 % of the total plans were completed without failure. The approach using ACs, in contrast, was able to execute all the randomly generated tasks successfully, with no execution failures, i.e. 100 % success rate. As expected, the total number of requested demonstration was higher in the AC approach (66 requests) compared to the SA approach (25 requests). The AC (with three actions) yields more possible combinations, which requires more human demonstrations than the SA with a single action. However, the difference is lower than an order of magnitude, and the number of requests in both cases represent a small percentage of the total number of actions executed or demonstrated in the 30 plans (350 actions).

To provide a failure example in the SA case, Fig. 8 shows two example scenarios for the pouring task that are presented sequentially to the robot. In scenario A (Fig. 8 a), the system starts with no schema in the database and requests demonstrations. After the demonstration, successful executions were carried out using both the SA and AC approaches. When the system is presented with scenario B (Fig. 8b-c) the SA approach used the already learned action for picking a bottle but fails in its execution, hitting the bottle before grasping it (Fig. 8 b). This is because

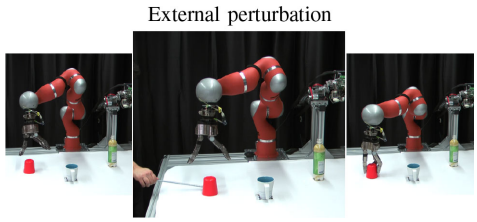


Fig. 9. Snapshots of the schema execution under an external perturbation for the AC `place side bottle tabler - pick bottom redC tablel - place bottom redC bottle`. The reaching motion is adapted on-line without calling the planner.

the SA approach is not able to identify the different motion dependencies between the picking of the bottle and the previous and posterior actions in scenario A and B. In contrast, the AC approach successfully considers these motion dependencies by defining the motion parameters for picking the bottle according to what action was executed before and what action comes next (Fig. 8c).

External perturbations: This test shows how the action execution monitoring can be exploited to cope with external perturbations. As a proof of concept, in Fig. 9 we show a local perturbation in the task execution. While the robot is approaching the red cup, this is moved away from its current position (middle of Fig. 9). The monitoring system detects this occurrence and the updated object pose is then used to adapt the execution of the next segment without the need of call of the task planner. Although preliminary, this result shows an interesting feature of our system.

V. Conclusions and Future Work

We presented a TAMP approach that efficiently generates feasible motions for the execution of manipulation tasks. Task planning is based on an object-centered description of geometric relations able to consistently represent changes with actions in the objects configuration space. This permits reasoning about feasible geometric changes already at the task planning level. For plan execution, we devise an approach that considers dependencies between consecutive actions in a plan to generate feasible motions. The approach is based on a new structure called Action Context, that associates symbolic actions to grounding mechanisms depending on the context of an action in a plan: what action comes next and what action was executed before. Motion parameters are stored in the leaves of a tree-based hierarchical decomposition of symbolic actions that is learned from demonstration. Our framework provides an appealing low-complexity alternative to existing TAMP approaches. However, it is only able to consider motion dependencies between consecutive actions, which may produce planning impasses in applications with longer horizon dependencies. On the other hand, action contexts use a symbolic representation to select motion parameters, hindering generalization over objects requiring the same manipulations but having different labels. Future work will address these limitations and extend the strategies to handle disturbances using the proposed framework.

References

[1] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning Theory and Practice*. Amsterdam, The Netherlands: Elsevier Science, 2004.
[2] S. Harnad, "The symbol grounding problem," *Physica D: Nonlinear Phenomena*, vol. 42, no. 1, pp. 335–346, 1990.
[3] D. S. Nau, M. Ghallab, and P. Traverso, "Blended planning and acting: Preliminary approach, research challenges," in *Proc. AAAI Conf. Artif. Intell.*, 2015, pp. 4047–4051.
[4] M. Ghallab, D. Nau, and P. Traverso, "The actor's view of automated planning and acting: A position paper," *Artif. Intell.*, vol. 208, pp. 1–17, 2014.
[5] C. Kemp, A. Edsinger, and E. Torres-Jara, "Challenges for robot manipulation in human environments," *IEEE Robot. Autom. Mag.*, vol. 14, no. 1, pp. 20–29, Mar. 2007.
[6] B. Quack, F. Wörgötter, and A. Agostini, "Simultaneously Learning at Different Levels of Abstraction," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015, pp. 4600–4607.
[7] F. Ingrand and M. Ghallab, "Deliberation for autonomous robots: A survey," *Artif. Intell.*, vol. 247, pp. 10–44, 2017.
[8] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *Int. J. Robot. Res.*, vol. 33, no. 14, pp. 1726–1747, 2014.
[9] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artif. Intell.*, vol. 247, pp. 229–265, 2017.
[10] G. Havur, K. Haspalamutgil, C. Palaz, E. Erdem, and V. Patoglu, "A case study on the tower of hanoi challenge: Representation, reasoning and execution," in *Proc. Int. Conf. Robot. Autom.*, 2013, pp. 4552–4559.
[11] J. Choi and E. Amir, "Combining planning and motion planning," in *Proc. Int. Conf. Robot. Autom.*, 2009, pp. 238–244.
[12] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 1470–1477.
[13] R. Lallement, L. De Silva, and R. Alami, "Hatp: An htn planner for robotics," 2014, *arXiv:1405.5345*.
[14] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 372–389, Apr. 2017.
[15] M. Edmonds *et al.*, "A tale of two explanations: Enhancing human trust by explaining robot behavior," *Sci. Robot.*, vol. 4, no. 37, 2019, doi: 10.1126/scirobotics.aay4663.
[16] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1255–1262, 2019.
[17] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *Int. J. Robot. Res.*, vol. 37, no. 10, pp. 1134–1151, 2018.
[18] F. Lagriffoul and B. Andres, "Combining task and motion planning: A culprit detection problem," *Int. J. Robot. Res.*, vol. 35, no. 8, pp. 890–927, 2016.
[19] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning." in *Proc. Int. Joint Conf. Artif. Intell.*, 2015, pp. 1930–1936.
[20] A. Agostini and D. Lee, "Efficient state abstraction using object-centered predicates for manipulation planning," 2020, *arXiv:2007.08251*.
[21] R. Caccavale, M. Saveriano, A. Finzi, and D. Lee, "Kinesthetic teaching and attentional supervision of structured tasks in human–robot interaction," *Auton. Robots*, vol. 43, no. 6, pp. 1291–1307, 2019.
[22] D. McDermott *et al.*, "Pddl-the planning domain definition language," CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Tech. Rep., 1998.
[23] M. Helmert, "The fast downward planning system," *J. Artif. Intell. Res.*, vol. 26, pp. 191–246, 2006.
[24] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural Comput.*, vol. 25, no. 2, pp. 328–373, 2013.