Combining Task and Motion Planning using Policy Improvement with Path Integrals

Dominik Urbaniak[1], Alejandro Agostini[1,2] and Dongheui Lee[1,3]

Abstract— **Task and motion planning deals with complex tasks that require a robot to automatically define and execute multi-step sequences of actions in cluttered scenarios. In this context, a linear motion is often not sufficient to approach a target object since collisions of the gripper with other objects or the target object might occur. Thus, motion planners should be able to generate collision-free trajectories for every particular configuration of obstacles for grounding the symbolic actions of the task plan. Current approaches either search for feasible motions offline using computationally expensive trial-and-error processes on physically realistic simulations or learn a set of motion parameters for particular object configuration spaces with little generalization. This work proposes an appealing alternative by efficiently generating trajectories for the collision-free execution of symbolic actions in variable scenarios without the need of intensive offline simulations. Our approach combines the benefit of learning from demonstration, to quickly generate an initial set of motion parameters for each symbolic action, with policy improvement with path integrals, to diversify this initial set of parameters to cope with different obstacle configurations. We show how the improved flexibility is achieved after a few minutes of training and successfully solves tasks requiring different sequences of picking and placing actions in variable configurations of obstacles.**

I. INTRODUCTION

In the last few years, significant progresses have been made in motion planning techniques for the execution of dexterous manipulation actions using robots [1], [2], [3]. Many of the current applications of these techniques concern single tasks in controlled scenarios, where the robot should execute tasks comprising a single action or a sequence of actions carefully predefined in advanced. Task and motion planning (TAMP) frameworks seek to scale up the applicability of these techniques to multi-task applications in variable scenarios. TAMP approaches use an abstract representation of the physical changes with actions, encoded into the so called planning operators [4], to quickly define a sequence of (symbolic) actions for the robot to achieve a desired configuration of objects (or goal) from variable initial situations. In this manner, well-studied robotic actions, such as pushing, picking, or placing, can be now concatenated using human-like instructions for the execution of multi-step tasks. Each of these symbolic actions is grounded by a motion plan that regards "detailed specifications" [5] of the environment. TAMP frameworks that rely on search methods to plan tasks and motions have the limitation of requiring high computational effort each time before acting. This makes a real-time capable robot harder to achieve. On the other hand, frameworks that use learning-based motion planners generate motions without deliberation, shifting the computational effort into the training. This reduces the computations at execution time, however, requires high generalization qualities to adapt properly to new objects and object configurations. This work focuses on improving the flexibility of motions while maintaining a high level of precision and predictability. Learning from demonstration (LfD) is utilized to efficiently encode motions from a single demonstration into a set of dynamic movement primitive (DMP) parameters [6]. These DMP parameters are further refined using a reinforcement learning (RL) approach based on policy improvement with path integrals (PI^2) [7], which permits shaping the trajectories for different configuration of obstacles. These different configurations, and the corresponding set of DMP parameters, are used to encode a policy into a neural network that permits a fast inference of the DMP parameters (shape of the trajectory) for a given initial and goal position of the robotic hand and obstacles on the way.

In the next section, we describe the related work. Section III introduces the most prominent methods of the proposed TAMP framework. In section IV, the generation of trajectory samples is explained. Section V applies the methodology to experiments and assesses the performance. Limitations and ideas for future work are addressed in the concluding section.

II. RELATED WORKS

One approach to solve TAMP problems couples the task and the motion planning search [8], [9]. Bidot et al. [9] propose to backtrack generated plans on two levels. *Action backtracking* reconsiders *what* action to perform. *Geometric backtracking* reconsiders *how* the action is performed. Due to the large space of possible geometric configurations their work focuses on strategies to guide the search with heuristics and prune the search tree with constraints. Dantam et al. [8] propose to incrementally increase the plan length during search and dynamically add and remove motion constraints. First, a constraint-solver provides a task plan. Then, for each action, a motion planner searches for a feasible solution. When the search fails for one action, new constraints are added and an alternate task plan is created. When no alternate task plan is found, all motion constraints are discarded and the plan length and time horizon for the motion planner are increased. Both works allow to find solutions to complex

[1] D. Urbaniak, A. Agostini, and D. Lee are with the Chair of Human-centered Assistive Robotics, Technical University of Munich, Munich 80333, Germany. {d.urbaniak,alejandro.agostini,dhlee} @tum.de.

[2] A. Agostini is also with the Department of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria.

[3] D. Lee is also with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), 82234 Wessling, Germany.

TAMP problems, however, also result in many unsuccessful calls to the motion planner, leading to computationally expensive operations that increase exponentially with the plan length. We solve a TAMP problem performed in [8] with a planning time reduction of more than one order of magnitude on average by training the motion planner for a few minutes and disregarding probabilistic completeness. Wells et al. [10] extend on [8] by training a SVM offline to provide a set of motion constraints through inference. This reduces the computation times significantly. However, the generation of 10000 training samples take the authors "about two days". Our approach generates the same amount of training data in about ten minutes. In contrast to [10], we use neural networks to represent the training data to improve precision. The learned policy is directly used in the TAMP problem and cannot be refined to achieve probabilistic completeness.

Another approach learns the motion execution instead of searching for a solution at test time [11], [12], [13]. In [11], a RL agent performs actions in the environment and receives rewards by comparing the actual changes of predicates with the expected ones given by the task plan. At the same time, reported errors can improve the accuracy of preconditions defined for the task planner. Their work is evaluated in a low dimensional task only and applying their RL algorithm in more complex environments is not practical. In our approach we apply policy-based RL to generalize the motion given by one demonstration. This leads to a more efficient learning process. Agostini et al. [12] utilize LfD to efficiently generate motion in a high dimensional space. A deeper connection of task and motion planner is established by *Action Contexts (ACs)* which represent three consecutive symbolic actions. In their learning process, unknown *ACs* trigger a request for a demonstration to learn motion parameters associated to the new action contexts and store them in a database. This enables learning a diverse task set. However, learning one task robustly requires several demonstrations in varying situations. In contrast, we rely on only one demonstration and use RL to successfully act in varying situations.

Toussaint [14] proposes an optimization-based approach applying logic geometric programming (LGP). It solves problems where the goal is represented by an objective function instead of a symbolic description utilizing a model of the robot. Instead, our approach is model-free which allows direct transfer of the results to other systems but does not consider singular configurations. Based on the LGP framework, Driess et al. [15] train a neural network to generate feasible action sequences faster at execution time compared to running a LGP tree search. Similarly, the policy-based RL method in the proposed framework optimizes an objective function and a neural network is trained offline to reduce computation times. The approaches based on LGP need a model of the robot dynamics to find optimal solutions after intensive computations. On the contrary, our approach is able to generate plans with long action sequences at low computational costs using off-the-shelf linear planners. It also permits generating collision-free motions quickly for variable object configurations.

III. METHODS

This section introduces the fundamental methods of our task and motion planning framework that is illustrated in Fig. 1. The task planner decomposes a complex task into symbolic actions that are transformed into DMP parameters generated by a policy improvement with path integrals (PI^2) method. The PI^2 approach iteratively adjusts the parameters of the forcing term θ_i of the DMPs according to a cost function that considers obstacles between the initial and goal positions of the robot hand. The set of parameters generated by the PI^2 for different obstacle configurations are used to encode a policy into a multilayer perceptron that permits inferring the specific parameters for the forcing term according to a given configuration of obstacles.

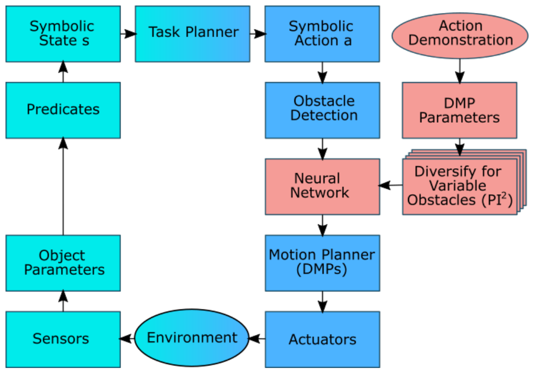

Fig. 1. The architecture of the proposed TAMP framework. High level symbolic actions are processed to robot controls that act physically on the environment (blue). The effects are observed and contrasted with the expected outcome of the executed action in the plan (cyan). This loop is executed online to solve a TAMP problem. The shape of the trajectory is decided by a neural network that is trained offline (red).

A. *Task Planning*

We adopt the traditional planning domain definition [4] and define a set of objects (e.g. `cube`, `cell`) and a set of predicates, coding object relations and properties (e.g. `on cell cube`), which are logical functions that takes value `true` or `false`. The set of predicates describing a particular scenario defines a *symbolic state* s. We define a set of planning operators (POs) represented in the traditional precondition-action-effect notation. The precondition part and effect parts describe the changes in the symbolic state with the PO execution. The action is the name of the PO and consists of a declarative description of an action and may contain parameters to ground the predicates in the precondition and effect parts. The task planner receives the description of the *initial state* s_{ini} and a *goal state* s_{goal} consisting of 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 s_{goal}. The search is performed using the fast downward planning system [16].

B. Dynamic Movement Primitives

Dynamic movement primitives (DMPs) approximate a demonstration D with a trajectory that is generated by a spring-damper system [6],

$$\hat{\tau}\ddot{\mathbf{y}} = \alpha(\beta(g - \mathbf{y}) - \dot{\mathbf{y}}) + f(t), \tag{1}$$

where $\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}$ define the trajectory, $\hat{\tau}$ the duration of the trajectory and g the goal. The system is critically damped with $\beta = \alpha/4$ and $f(t)$ represents the learnable forcing term,

$$f(t) = \frac{\sum_{i=1}^{N} \Psi_i(t)\theta_i}{\sum_{i=1}^{N} \Psi_i(t)}, \tag{2}$$

where Ψ represents N Gaussian basis functions with fixed centers and widths, scaled with adjustable parameters θ_i. The demonstration provides $\mathbf{y}_D, \dot{\mathbf{y}}_D, \ddot{\mathbf{y}}_D$. Solving (1) for $f(t)$ returns the approximation parameterized in the basis function weights θ_i. Defining any initial position y_0 and goal position g, the DMP creates a new trajectory using the learned forcing term to reproduce the demonstration in a new situation.

a) Roto-Dilatation Invariance: When the difference in length and direction of $\overrightarrow{y_0 g}$ compared to $\overrightarrow{y_{0,D} g_D}$ becomes larger, the forcing term distorts the trajectory instead of preserving the shape of the demonstration. Those distortions vary with the selection of the spring-damper parameters α, β. Ginesi et al. [17] address this problem by transforming the forcing term parameters $\theta_{i,D}$ of a DMP the same way $\overrightarrow{y_0 g}$ is transformed from $\overrightarrow{y_{0,D} g_D}$ using normalization and a rotation matrix to achieve roto-dilatation invariance. We apply the rotation invariance in the two dimensional XY plane and dilatation invariance to the three dimensions XYZ.

C. Policy Improvement with Path Integrals

Policy improvement with path integrals (PI^2) is a model-free, policy-based RL algorithm that is derived from stochastic optimal control (SOC) [7]. It is applied to parameterized policies and its performance in high dimensional problems is numerically robust. A DMP provides the initial policy, e.g. the learned parameters $\theta_{i,D}$. The exploration variance σ^2 is the only tuning parameter. The exploration noise $\epsilon_{i,t}$ is sampled at each time step from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ and is added to θ_i. Stulp et al. [18] simplified this approach by sampling ϵ_i only once at the first time step and updating the policy only once after the last time step with the aggregated costs. Those two modifications improve the performance, reduce the convergence time and "do not violate any of the assumptions required for the derivation of PI^2 from SOC" [18]. Each iteration creates K random samples,

$$\theta_{i,k}^{j+1} = \theta_i^j + \epsilon_{i,k}^j, \tag{3}$$

where $k = 1, ..., K$ and $j = 1, ..., J$ with J the maximum number of iterations. Each policy is evaluated with a cost function $S(\theta_{i,k}^j)$ and weighted according to its performance compared to the other $K - 1$ samples in one iteration:

$$W_\theta(\theta_{i,k}^j) = \exp\left(-\gamma \frac{S(\theta_{i,k}^j) - \min S(\boldsymbol{\theta}_i^j)}{\max S(\boldsymbol{\theta}_i^j) - \min S(\boldsymbol{\theta}_i^j)}\right), \tag{4}$$

where $\gamma = 10$ is a constant. Finally, the parameters of the new policy θ^{j+1} are calculated as

$$\theta_i^{j+1} = \frac{\sum_{k=1}^{K} W_\theta(\theta_{i,k}^j)\theta_{i,k}^j}{\sum_{k=1}^{K} W_\theta(\theta_{i,k}^j)}. \tag{5}$$

D. Neural Network

PI^2 provides a large amount of varying and precise trajectories represented by the forcing term parameters θ_i. A neural network is trained on the data via supervised learning to enable the decision making using the Levenberg-Marquardt algorithm [19]. The generated data is not independent. Within one optimization, each iteration j depends on all previous iterations $1, ..., j-1$. Hence, the neural network is overfitted to the training data and is not expected to generalize to unseen inputs. However, it serves as a compact representation of the data and precisely reproduces the training data. The input data is collected from the PI^2 cost function and the outputs are the forcing term parameters θ_i.

IV. TRAJECTORY GENERATION FOR OBSTACLE AVOIDANCE

Tasks that involve multiple objects demand flexible motion. In the trivial case, the target object can be approached by linear motion. In non-trivial cases, however, the goal position is located on the other side of the object or other objects must be avoided. Here, the objects might have different sizes or are placed in different configurations. Hence, dependent on the situation, a specific shape of the trajectory is suitable. We apply PI^2 on a trivial demonstration to generate curved variations that are suitable for diverse object sizes and configurations. To this end, we propose a cost definition for PI^2 that tunes the forcing term of the DMPs to create these curved variations that allow the robot to avoid obstacles of variable size. One cost term decreases continuously while the other cost terms constrain the decrease to ensure that the result of each iteration comply with the defined requirements of a TAMP problem w.r.t. to the scope of the work space and the goal precision.

The continuous term of the cost function depends on the trajectory height $H = \min(z_p)$, which is defined by a vector of weighted heights $z_p = h_p/W_p$ for observation points p along the linear trajectory. At the observation points (e.g. boundaries of an obstacle) the heights of the trajectory h_p are measured. The weights W_p normalize the heights and allow the generation of trajectories with varying heights at the observation points, when, e.g., an obstacle has different local maxima. Hence, the choice of the observation points p and weights W_p depends on the type of obstacles. The second cost term represents a *hinge loss* function,

$$S_{scope} = \sum_{t=1}^{T} \max(0, m + y_0 - \mathbf{y}_t), \tag{6}$$

which counts the time steps t where the trajectory exceeds the border with a margin m. This term depends on the task environment and summing variations of this term allows to contain the trajectories within a specific scope. The third term of the cost function ensures the goal precision of the trajectory, $S_{prec} = ||g - y_{end}||$, which increases when the actual end of the trajectory y_{end} deviates from the expected goal g of the DMP. The total cost S is the weighted sum,

$$S = -H + c_1 \cdot S_{prec} + c_2 \cdot S_{scope}, \tag{7}$$

where $c_1 = 10$, $c_2 = 1$ are chosen to regulate the influence of the corresponding terms. The height H adds negative cost, as larger heights are desired. The optimization converges towards $H = \infty$, hence, a maximum height must be set to terminate the optimization process.

During the optimization process, the distance between y_0 and g is constant and taken from the demonstration: $l_D = ||g_D - y_{0,D}||$. However, DMPs permit setting the initial position y_0 and the goal g arbitrarily. The dilatation invariance then preserves the shape of the demonstration. Hence, we define a dilatation-invariant length-to-height ratio $r_c = H/l_D$ representing the degree of curvature, which is constant for a specific set of DMP parameters θ_i.

Fig. 2 illustrates an example of this process for two points $p1, p2$ with equal weights $W_p = [1, 1]$. It reaches the termination condition $r_c = 1$ at iteration $J = 889$. Here, the trajectory heights at the observation points are $h_{p1} \geq l_D$ and $h_{p2} \geq l_D$ by definition. This means that the robot hand can avoid an obstacle of height $\hat{H} = 0.15m$ when $||y_0 - g|| = 0.15m$. When $||y_0 - g|| = 0.2m$ the same parameters reach the height of $H = 0.2m$. To avoid the obstacle with less effort, the parameters from previous iterations, e.g. $j = 700$, are more suitable. Each iteration j generates DMP parameters that avoid a virtual object most efficiently compared to any other iteration up to the defined length-to-height ratio $r_c = 1$.

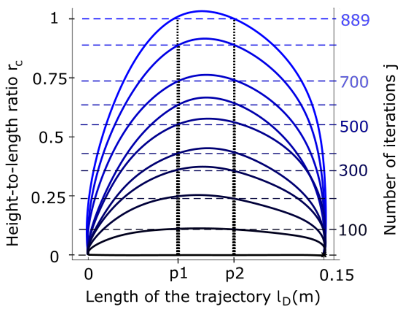

Fig. 2. Evolution of an optimization process with PI^2.

A. *Influence of the Optimization on the DMP Parameters*

Each iteration $j = 1, ..., J$ represents the forcing term parameters θ_i^j of the DMPs that generate the trajectory. In this 2D optimization, the DMPs in X and Z direction are tuned. Fig. 3 illustrates the corresponding parameters for the demonstration D, at $j = 500$ and at $r_c = 1$. We can observe a clear correlation between the trajectory height and the magnitude of the activations. A trajectory with a stronger curvature shows a steep ascend in the beginning, the first half of the parameters of θ_i^{889} in Z reflects this slope by larger values, representing higher forces in positive Z direction. Also the parameters in X direction show an increase in magnitude. This is explained by the constant duration $\hat{\tau}$. The trajectory with higher curvature requires higher velocities to arrive at the same goal g after the same amount of time. Despite the visible correspondence, the evolution provides no indication for a trivial solution such as, for example, multiplying all parameters with some scaling factor.

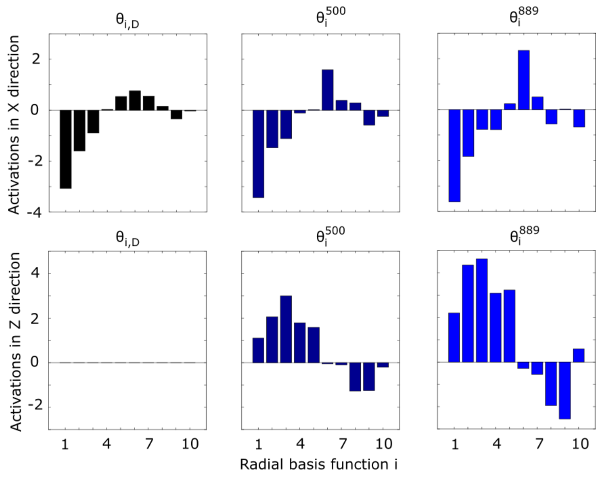

Fig. 3. Evolution of the DMP parameters with PI^2.

V. EXPERIMENTS

To assess the validity of our approach with respect to the state of the art, we use the same benchmark scenario as in [8], where colored cubes are placed in a grid configuration. The task consists of picking and placing cubes in an ordered manner until the goal configuration is reached (see Fig.4). This requires reaching the locations for grasping and placing with high precision. All initial and goal positions lie on the same XY plane. Collisions are avoided by moving in positive Z direction. As in [8], we use a simulated environment for the experiments, implemented using the physically realistic simulator V-REP [20]. Fig. 4 shows an image of the simulation during a *placing* operation of `cube4` avoiding `cube6` and `cube8`. The task planning domain is defined using the predicate `on ?cell ?cube`. The notation `on ?cell air` is used to indicate that a cell is unoccupied. A symbolic action is defined as `pickplace ?from ?to ?cube`, where `?from` and `?to` are grounded with the initial and target cells, respectively. Table I shows an example planning operator in the planning domain definition language (PDDL) [21]. Several trajectories with different shapes should be generated for sorting out the cubes without collisions in a variable set of situations. We let the system learn an action policy using PI^2, which is able to shape the trajectory for every particular configuration of cubes and action (see Sec. IV). To learn the action policy, the following steps are applied in an offline training process after the action demonstration (in red in Fig. 1): 1) Identify observation points; 2) Define cost function; 3) Generate DMPs for different obstacles; 4) Learn action policy (neural network). Afterwards, the learned action policy is evaluated and compared to [8].

TABLE I

THE PICK-AND-PLACE TASK IS DECOMPOSED INTO SEQUENCES OF THIS *pickplace* ACTION.

```
(:action pickplace
:parameters (?from ?to ?cube)
:precondition (and (on ?to air)
(on ?from ?cube))
:effect (and (on ?from air) (on ?to ?cube)
(not (on ?to air)) (not (on ?from ?cube))))
```

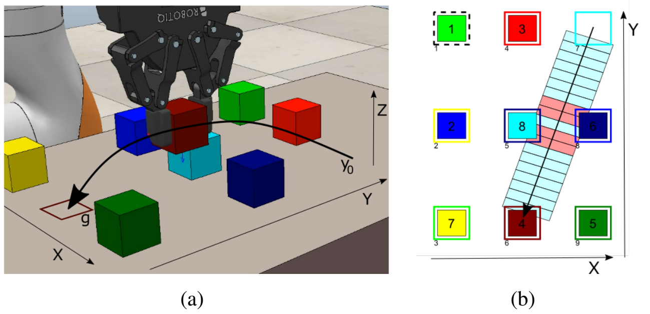

Fig. 4. Simulation of the symbolic action `pickplace cell7 cell6 cube4`.

A. Obstacle Description

The trajectories that avoid an obstacle must maintain the height H for a specific length L which depends on the object length $\hat{L}$. In Fig. 2, the points $p1, p2$ represent a length $L = |p1 - p2|$ and allow avoiding an obstacle of $\hat{L} = L$ when the distance $||y_0 - g|| = l_D$. Otherwise, $L \neq \hat{L}$ due to the dilatation invariance. To allow a representation of various situations where different trajectory lengths L are required we choose to discretize the area between the initial position y_0 and the goal g with $B = 20$ borders to create distinctive areas that detect obstacles (illustrated in Fig. 5). Each border $b = 1, ..., B$ can be chosen as observation point p. The observation width $w_A = h_{cube} + 0.03m$ is constant and can be adjusted to represent the combined width of the grasped object and the gripper. When the vertices of an object intersect with an area $A_{b,b+1}$, the borders at each side are activated. This is sufficient, since we are interested in the extreme points of convex objects. The activated border that is closest to y_0 or g is selected as the first observation point $p1$. The other point $p2$ is set symmetrically such that the trajectory length L is centered between y_0 and g. Similar to the trajectory height H, we must define a dilatation invariant description of L, which is constant for specific DMP parameters θ_i. Since the boundaries are defined relative to the distance $||y_0 - g||$, they are dilatation invariant. Hence, we define $\tilde{L}$ as a tuple of two borders, e.g. $\tilde{L} = (8, 13)$. Figure 6 shows an example how the same object parameters require different trajectory shapes (and therefore different r_c and $\tilde{L}$ representing different DMP parameters θ_i) when the distance between y_0 and g differs.

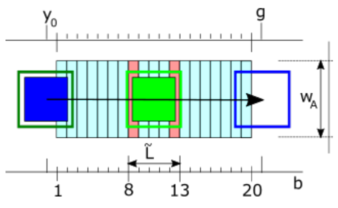

Fig. 5. Discretized observable area for obstacle description.

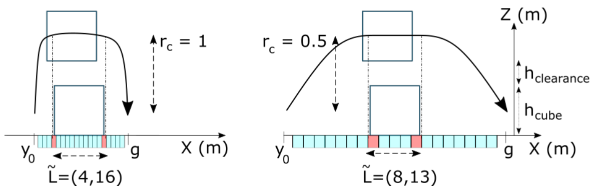

Fig. 6. The length $\hat{L}$ and height $\hat{H}$ of the obstacle are constant. The required trajectory shape, represented by r_c and $\tilde{L}$, varies depending on the distance $||y_0 - g||$.

B. PI^2 Optimization

The optimization process is applied to each of the $n_{\tilde{L}} = 10$ symmetric combinations of $\tilde{L}$. That means for each optimization we derive $p1$,

$$p1 = \frac{l_D \cdot \tilde{L}(1)}{B + 1}, \tag{8}$$

and $p2$ the same way. We constrain the generated trajectories for all optimizations to be contained along the X axis within a margin $m = 0.01$ from $y_{0,D}$ and g_D respectively,

$$\begin{aligned} S_{scope} = &-\sum_{t=1}^{T} \min(0, m + \mathbf{y}_{X,t} - y_{0,X}) \\ &-\sum_{t=1}^{T} \min(0, m + g_X - \mathbf{y}_{X,t}). \end{aligned} \tag{9}$$

Due to the stochastic property of the PI^2 optimization, we evaluate its numerical robustness running the optimization $n_P = 10$ times resulting in $n_P \times n_{\tilde{L}}$ optimizations. Figure 7 illustrates the resulting computation times and number of iteration J until the termination condition $r_c = 1$ is reached. Both metrics evolve nearly proportional. Trajectories with steep slopes, e.g. $\tilde{L} = (1, 20)$, take more computational effort to reach the same height. The deviation of the effort is also larger for more complex slopes. All 100 optimizations achieve $r_c = 1$ within one minute verifying the expected stability of the PI^2 method.

C. Learning the Action Policy

In this section we define the input parameters for training a neural network that provides the action policy for the motion planner. The height-to-length ratio r_c and the length $\tilde{L}$ serve as an input for the neural network to retrieve the forcing term parameters θ_i as output which are expected to generate the desired trajectory. The variable $r_L = (|\tilde{L}(2) - \tilde{L}(1)| + 1)/B$ represents the tuples of $\tilde{L}$ as one value, resulting in $r_L = 0.1, 0.2, ..., 1$ for our discrete selection.

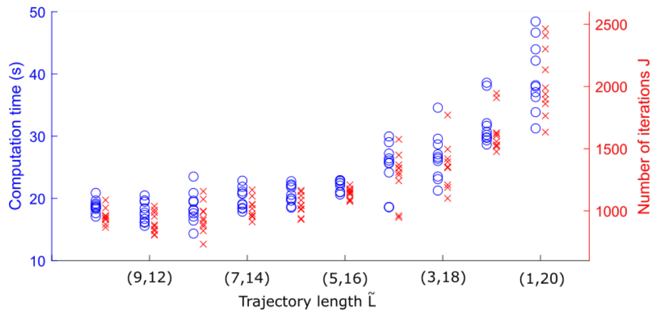

Fig. 7. Ten PI^2 optimizations are performed for each trajectory length $\tilde{L}$. For each optimization the final computation time (blue) and the number of iterations (red) are plotted. All optimizations reach the termination condition $r_c = 1$ within one minute, validating the expected numerical robustness of PI^2.

To keep a near identical distribution of the training data, we sample uniformly the same amount of samples from each optimization. The number of samples is determined by the optimization with the minimum number of iterations J. To evaluate the performance, we train n_P networks on the combined data sets with 50 hidden neurons for 40 epochs and evaluate the goal precision $d_g = S_{prec}/l_D$ and the deviations in height $d_H = r_c - H/l_D$. To this end, Fig. 8 presents for each r_L, $n_P \times 50$ uniformly distributed ratios r_c in the range $[0, 1]$. The goal locations deviate only positively with a mean of 0.027% and a maximum of 0.16%, hence, no collisions of the gripper with the surface occur here. The mean height deviation is at 0.47% with the maximum at 6.8% and the minimum at -3.4%. The largest deviations appear at the extreme points of the range at either $r_{c,0}, r_{c,50}$ and are highlighted in the figure for the two most prominent examples.

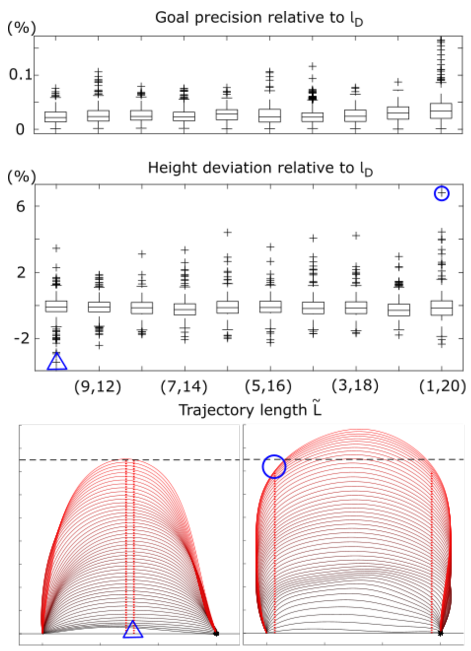

Fig. 8. Height deviations d_H and goal precision d_g of 5000 reproduced trajectories using neural networks, illustrating the two most significant outliers.

D. Experimental Evaluation

We perform 20 consecutive runs with randomly initialized goal cells and cube positions. Each *pickplace* action consists of one *pick* and one *place* operation where the DMP parameters are retrieved from the neural network. The *pick* operation must always avoid a cube at y_0 and g. Hence, the steepest trajectory shape $r_L = 1$ is selected. The *place* operation distinguishes if and where an obstacle is located and therefore selects varying r_L. The ratio r_c varies in both operations and depend on whether and where an obstacle must be avoided. A small clearance height is added on top to cope with uncertainties. The width of the goal cells w_g monitors the precision. When a vertex of a cube exceeds the area that w_g spans, the symbolic action is reported as a failure. Table II shows the parameters of the experiment that our TAMP framework performed. All 159 *pickplace* actions are successfully executed with the first attempt. Figure 9 provides the computation times per plan length for the task and the motion planner. The motion planning time is composed of the time for computing the output from the neural network and the trajectories that are derived per time step by the dynamics of each DMP. Compared to the *Iteratively Deepened Task and Motion Planning* (IDTMP) from Dantam et al. [8], our approach requires single calls to the task planner as well as to the motion planner and therefore, the computational effort is reduced significantly. Our framework performs one search for task planning at the beginning of each run. The computation times increase linearly with the plan length not surpassing $0.01s$. Due to the continuous oscillations between calls to the task and to the motion planner, the task planning time of the IDTMP increases exponentially. At a plan length of four symbolic actions it takes $0.01s$ and $10s$ for ten symbolic actions. Regarding motion planning, times of our approach increase linearly at a lower rate and is an order of magnitude faster at a plan length of five symbolic actions. In contrast to IDTMP, our approach requires to train the motion planner once. The computation time for generating the demonstration, the trajectory samples, and training the neural network is nine minutes on average (Tab. III), performed on a Intel i7-4790 CPU @ $3.60GHz$, $16GB$ RAM.

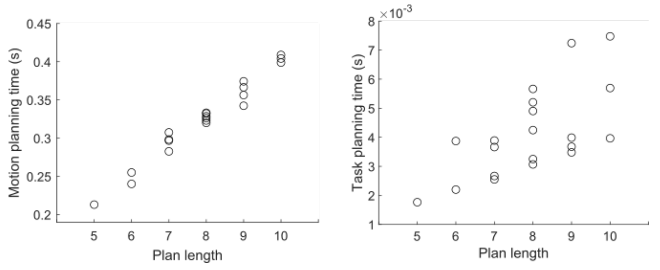

Fig. 9. Computation times of the task planner and the motion planner for 20 random runs.

VI. DISCUSSION

In the experiment, the robot rearranges a set of equally sized cubes that are located on a grid layout. In real-world

TABLE II

PARAMETERS FOR THE PICK-AND-PLACE TASK.

DMP parameters		
Number of time steps	T	200
Duration	$\hat{\tau}$	15
Number of basis functions	N	10
Damping coefficient	α	10
Length of the demonstration (m)	l_D	0.15
PI^2 parameters		
Exploration noise	ϵ	0.04
Number of samples	K	10
Experiment parameters (m)		
Width of the goal cell	w_g	0.05
Cube dimensions	h_{cube}	0.04
Width of the observed area (Fig. 5)	w_A	$h_{\text{cube}} + 0.03$
Grasping height	h_{grasp}	0.02
Clearance height	$h_{\text{clearance}}$	$10\% \cdot \|\|y_0 - g\|\|$

TABLE III

TRAINING TIMES FOR THE PICK-AND-PLACE TASK.

Training times (s)	
Generating the demonstration	0.02
PI^2 optimizations of ten trajectory shapes	241
Training of the neural network	279
Total training time	520

situations, however, objects of varying shape and size may appear in arbitrary configurations. The proposed approach promises to deal well with arbitrary configurations, as long as objects are separated enough to let the gripper manipulate them without collisions. The current version of our approach can be easily extended to handle these collisions and objects with more arbitrary shapes than cuboids. For example, we could enrich the cost function with more parameters to better define boundaries of obstacles. Another alternative would be to use the bounding boxes of objects (i.e. transforming objects into cuboids) and then use the current cost function. Compared to the probabilistic completeness of the IDTMP framework, our approach is constrained by the learned trajectories and is not guaranteed to always find feasible motions. Since our approach is model-free, the mechanical limitations of each robot must be handled additionally. In future work, we aim at improving the integration of different optimizations to also allow meaningful interpolations between different trajectory shapes and extending the DoFs to consider the gripper's orientation. Furthermore, the coupling ability of DMPs [6] might have the potential to combine our generated shapes to more complex trajectories and additionally enhance the adaption to a dynamic environment.

VII. CONCLUSIONS

The proposed TAMP framework with a motion planner based on the combination of LfD and RL reduces the computation times during an experiment by more than one order of magnitude compared to a search-based framework [8] after nine minutes of training. This makes the proposed approach suitable for task plans with more than ten actions. Our approach is able to select a trajectory shape that avoids collisions for a variable set of configuration of obstacles. In tasks with varying object sizes that require similar trajectory shapes, our approach is particularly convenient due to its efficient trajectory generation with PI^2.

ACKNOWLEDGMENT

This work has been partially supported by the Helmholtz Association and by the Austrian Science Fund (FWF) Project M2659-N38.

REFERENCES

[1] M. Deisenroth, G. Neumann, J. Peters *et al.*, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1-2, pp. 388–403, 2013.

[2] A. Billard and D. Kragic, "Trends and challenges in robot manipulation," *Science*, vol. 364, no. 6446, 2019.

[3] S. Calinon and D. Lee, "Learning control," in *Humanoid Robotics: a Reference*, P. Vadakkepat and A. Goswami, Eds. Springer, 2018.

[4] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Elsevier, 2004.

[5] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *International Conference on Robotics and Automation*. IEEE, 2011, pp. 1470–1477.

[6] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[7] E. Theodorou, J. Buchli, and S. Schaal, "Learning policy improvements with path integrals," in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 828–835.

[8] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[9] J. Bidot, L. Karlsson, F. Lagriffoul, and A. Saffiotti, "Geometric backtracking for combined task and motion planning in robotic systems," *Artificial Intelligence*, vol. 247, pp. 229–265, 2017.

[10] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE robotics and automation letters*, vol. 4, no. 2, pp. 1255–1262, 2019.

[11] B. Quack, F. Wörgötter, and A. Agostini, "Simultaneously learning at different levels of abstraction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 4600–4607.

[12] A. Agostini, M. Saveriano, D. Lee, and J. Piater, "Manipulation planning using object-centered predicates and hierarchical decomposition of contextual actions," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5629–5636, 2020.

[13] R. Caccavale, M. Saveriano, A. Finzi, and D. Lee, "Kinesthetic teaching and attentional supervision of structured tasks in human-robot interaction," *Autonomous Robots*, vol. 43(6), pp. 1291–1307, 2019.

[14] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning." in *International Joint Conference on Artificial Intelligence*, 2015, pp. 1930–1936.

[15] D. Driess, J.-S. Ha, and M. Toussaint, "Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image," *arXiv preprint arXiv:2006.05398*, 2020.

[16] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[17] M. Ginesi, N. Sansonetto, and P. Fiorini, "Dmp++: Overcoming some drawbacks of dynamic movement primitives," *arXiv preprint arXiv:1908.10608*, 2019.

[18] F. Stulp and O. Sigaud, "Policy improvement methods: Between black-box optimization and episodic reinforcement learning," 2012, hal-00738463.

[19] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*. Springer, 1978, pp. 105–116.

[20] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2013, pp. 1321–1326.

[21] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl-the planning domain definition language," 1998.