Blending Task and Motion Planning using Learning from Demonstration and Reinforcement Learning

handed in
MASTER'S THESIS

Bachelor of Science Dominik Urbaniak

born on the 20.10.1994
living in:
Marienburger Str. 22
85221 Dachau
Tel.: 01573-5220202

Human-centered Assistive Robotics
Technical University of Munich

Univ.-Prof. Dr.-Ing. Dongheui Lee

Supervisor:	Dr. Alejandro Agostini
Start:	15.07.2020
Intermediate Report:	09.12.2020
Delivery:	02.02.2021

TECHNISCHE UNIVERSITÄT MÜNCHEN
Human-centered Assistive Robotics
UNIV.-PROF. DR.-ING. DONGHEUI LEE

January 29, 2021

M A S T E R ' S T H E S I S
for
Dominik Urbaniak
Student ID 3653363, Degree RCI

Blending Task and Motion Planning using Learning from Demonstration and Reinforcement Learning

Problem description:

Task and motion planning (TAMP) architectures [1] combine the efficiency of symbolic planning methods to quickly generate task plans [3] with motion planning mechanisms to ground symbolic actions [4]. Most of the TAMP frameworks bring together these two planning paradigms through geometric reasoning methods that search for suitable acting parameters to execute symbolic actions [1, 2]. However, this strategy requires intensive computation and several calls to motion planning on unfeasible actions to search for solutions in the large object configuration space. This work tackles these limitations by associating to symbolic actions action policies that permit defining acting parameters reactively for each particular object configuration, without the need of deliberation. These policies are learned in a combined strategy that, on the one hand, generates a set of dynamic movement primitive (DMP) parameters from demonstration of symbolic actions [4]. On the other hand, it generates an action policy using a reinforcement learning (RL) approach based on policy improvement with path integrals [5] to generalize these parameters for action execution in unforeseen situations.

Tasks:

- Literature overview on TAMP, learning from demonstration (LfD), and policy-based RL.
- Implement an RL approach for the execution of symbolic actions using LfD as guided exploration.
- Integrate the RL approach into a TAMP framework.
- Experimental evaluation of scalability and transferability of the TAMP framework using RL.

Bibliography:

[1] J. Bidot et al. Geometric backtracking for combined task and motion planning in robotic systems, in *Artificial Intelligence Journal* 2017.
[2] N. Dantam et al. An incremental constraint-based framework for task and motion planning, in *The International Journal of Robotics Research*. 2018.
[3] M. Ghallab et al. Automated Planning: theory and practice, *Elsevier*. 2004.
[4] A. Ijspeert et al. Dynamical movement primitives: learning attractor models for motor behaviors, in *Neural computation*. 2013.
[5] M. Deisenroth et al. A survey on policy search for robotics, in *Found. and Trends in Rob.*. 2013.

Supervisor:	Dr. Alejandro Agostini
Start:	15.07.2020
Intermediate Report:	09.12.2020
Delivery:	02.02.2021

(D. Lee)
Univ.-Professor

Abstract

Task and motion planning deals with complex tasks that require the execution of multiple actions in a chronological order and the ability to generalize to variable object configurations. Symbolic planning efficiently generates task plans of multiple symbolic actions. Grounding these symbolic actions such that feasible motion is executed in varying scenarios presents a major challenge that this thesis focuses on by developing a TAMP framework that grounds symbolic actions with efficient and divers motion generation using learning from demonstration and reinforcement learning. 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 must be able to generate collision-free trajectories for every particular configuration of obstacles. Current approaches either use computationally expensive search to find feasible motion or learn a set of motion parameters for particular object configurations with little generalization. Our approach combines the benefit of learning from demonstration, to quickly generate an initial set of motion parameters for each symbolic action, and policy improvement with path integrals, to diversify this initial set of parameters to cope with different configuration of obstacles. We show how the improved flexibility is achieved with a few minutes of training and successfully solve tasks requiring different sequences of picking and placing actions with variable configuration of obstacles.

Contents

Chapter 1

Introduction

Robots are often presented in fictive novels and movies as technologies that surpass humans in their intelligence and motor skills. In reality, however, there are mostly industrial robots that are restricted to "well-known and well-engineered environments" [GNT04] or robotic vacuum cleaners that perform "single simple tasks" [GNT04]. Despite the sixteen years that have passed since Ghallab et al. published these statements, significantly more capable robots are not broadly available. In 2016, Stone et al. commented on this to be "[d]isappointingly slow growth" [SBB+16]. One major challenge of performing multi-step tasks with varying object configurations is addressed in this work. To this end, an important role plays the "reasoning side of acting": planning [GNT04].

1.1 Problem Statement

Motion planning reasons about "detailed specifications" [KLP11] of the environment such as the geometry of objects. When a robot manipulates objects, high precision is critical. A deviation of one centimeter may result in colliding with the object instead of grasping it possibly resulting in irreversible consequences. Recent advances in motion planning enable robots to reliably perform dexterous tasks under variable situations [ABC+20, VSB+19]. Motion planners excel at performing a single task, however, often in daily life, tasks consist of multiple steps which are linked logically. For example, when pouring a drink into a glass, the bottle must be grasped first. These logical requirements are difficult to implement into a motion planner. On the other hand, task planners handle long sequences of actions efficiently by defining high-level symbolic descriptions of the environment. Task and motion planning (TAMP) combines both approaches to deal with dexterous and multi-step tasks. However, the combination requires careful considerations about the degree of task decomposition and the communication of geometric constraints from symbolic to numerical description. Figure 1.1 illustrates the general problem of Task and Motion planning. TAMP frameworks approach a solution in different ways. Frameworks that rely on search methods to plan tasks and motions have

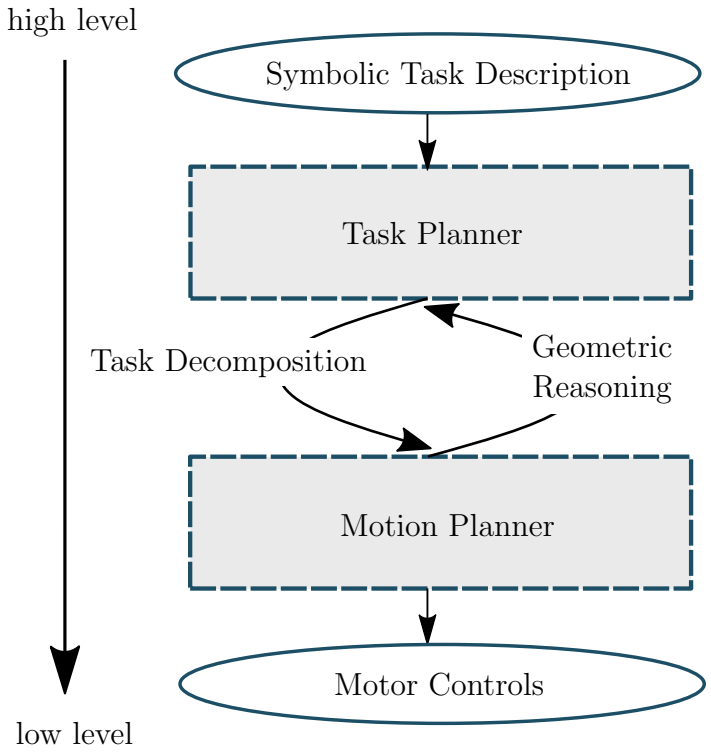


Figure 1.1: General approach to task and motion planning.

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 motion 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 [INH+13]. These DMP parameters are further refined using a reinforcement learning (RL) approach based on policy improvement with path integrals (PI^2) [TBS10], 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.

1.2 Related Work

One approach to solve TAMP problems couples the task and the motion planning search [DKCK18, BKLS17]. Bidot et al. [BKLS17] 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. [DKCK18] 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 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 [DKCK18] 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. [WDSK19] extend on [DKCK18] 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 [WDSK19], 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 [QWA15, ASLP20]. In [QWA15], 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. However, their work is evaluated in a low dimensional task only and requires large training efforts to generate feasible motion from scratch. 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. [ASLP20] 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 [Tou15] 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 do not consider singular configurations. Based on the LGP framework, Driess et al. [DHT20] 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 require intensive computations to find optimal solutions. On the contrary, our approach is able to generate plans with long action sequences at low computational cost using off-the-shelf linear planners. It also permits generating collision-free motions quickly for variable object configurations.

Chapter 2

Technical Approach

This chapter introduces the fundamental methods of our task and motion planning framework that is illustrated in Fig. 2.1. The task planner decomposes a complex task into symbolic actions that are transformed into parameters generated by the action policy. The action policy is learned in an offline process. One action is demonstrated and encoded into dynamic movement primitives (DMPs). The policy improvement with path integrals (PI^2) method iteratively adjusts the parameters of the forcing term θ_i of the DMPs according to a cost function that considers obstacles between the initial position and the goal position 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.

2.1 Task Planning

Tasks are defined using the traditional planning domain definition [GNT04]. A *domain* file describes the relevant predicates, constants and actions of a task environment in a general way using variables (`?obj`). It is encoded in the *Planning Domain Definition Language* (PDDL) [MGH+98] and presented in Tab. 2.1. Predicates describe the object relations and properties (e.g. `on cell cube`). A planning operator describes how the task planner can change the environment. It is represented by an action that must comply with specified preconditions and generates effects after the execution. The *problem* definition describes a specific situation with an initial and a goal configuration of objects using the predicates (Tab. 2.2). A linear planner (e.g. fast-downward planner [Hel06]) then searches for a solution to reach the goal configuration using the available actions. This decomposes a complex task into a set of symbolic actions. In this work, we then map object identities to coordinates that satisfy the numerical input requirements of the DMPs.

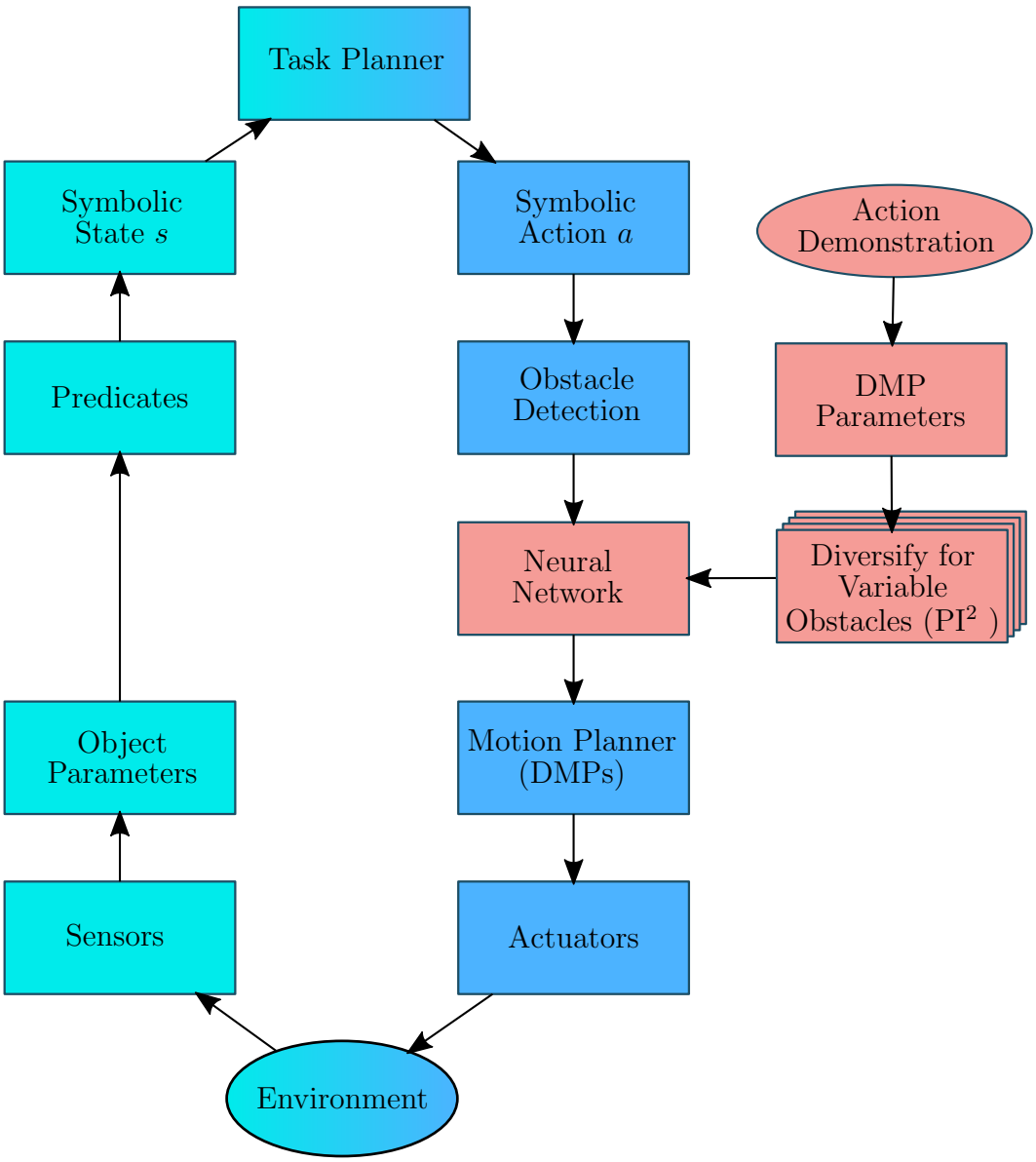


Figure 2.1: The architecture of the proposed TAMP framework. High level symbolic actions are processed to robot controls that act physically on the environment (blue). The effects are observed and contrasted with the expected outcome of the executed action in the plan (cyan). This loop is executed online to solve a TAMP problem. The shape of the trajectory is decided by a neural network that is trained offline (red).

```
(define (domain <domain-name>)
(:requirements :strips :typing)
(:constants air <...>)
(:predicates (on ?obj2 ?obj1) (left ?obj2 ?obj1) <...>)
(:action pop
:parameters (?obj1 ?obj2)
:precondition (on ?obj2 ?obj1))
:effect (and (on ?obj2 air) (not(on ?obj2 ?obj1))))
(action <next-action>
<...>))
```

Table 2.1: Composition of a *domain* in PDDL.

```
(define (problem <problem-name>)
(:domain <domain-name>)
(:objects object1 object2 object3 <...>)
(:init (on object1 object3) (left object1 object2) <...>)
(:goal (and (left object1 object2) <...>)))
```

Table 2.2: Composition of a *problem* in PDDL.

2.2 Motion Planning

2.2.1 Dynamic Movement Primitives (DMP)

Dynamic movement primitives (DMPs) approximate a demonstration with a trajectory that is generated by a spring-damper system [INH+13],

$$\hat{\tau}\ddot{\mathbf{y}} = \alpha(\beta(g - \mathbf{y}) - \dot{\mathbf{y}}) + f(t), \tag{2.1}$$

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,

$$f(t) = \frac{\sum_{i=1}^{N} \Psi_i(t)\theta_i}{\sum_{i=1}^{N} \Psi_i(t)}, \tag{2.2}$$

where Ψ represents N basis functions with fixed centers and widths, scaled with adjustable parameters θ_i.
The demonstration D provides $\mathbf{y}_D, \dot{\mathbf{y}}_D, \ddot{\mathbf{y}}_D$. Solving (2.1) 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.

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. [GSF19] 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.

2.2.2 Policy Improvement with Path Integrals (PI^2)

Policy improvement with path integrals (PI^2) is a model-free, policy-based RL algorithm that is derived from stochastic optimal control (SOC) [TBS10]. It is applied to parameterized policies and performs numerically robust in high dimensional problems. 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. [SS12] 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" [SS12]. Each iteration creates K random samples,

$$\theta_{i,k}^{j+1} = \theta_i^j + \epsilon_{i,k}^j, \tag{2.3}$$

where $k = 1, ..., K$ and $j = 1, ..., J$ with J the maximum number of iterations. Each policy is evaluated with a cost function $S(\theta_{i,k}^j)$,

$$S(\theta_{i,k}^j) = \phi_{T,k}^j + \sum_{t=1}^{T-1} q_{t,k}^j, \tag{2.4}$$

where $\phi_{T,k}^j$ is the *terminal cost* of a sample k at an iteration j and $q_{t,k}^j$ is the *immediate cost* at each time step t. Each trajectory sample $\theta_{i,k}^j$ is then weighted according to its performance compared to the other $K - 1$ samples in one iteration:

$$W_\theta(\theta_{i,k}^j) = \exp\left(-\gamma \frac{S(\theta_{i,k}^j) - \min S(\boldsymbol{\theta}_i^j)}{\max S(\boldsymbol{\theta}_i^j) - \min S(\boldsymbol{\theta}_i^j)}\right), \tag{2.5}$$

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

$$\theta_i^{j+1} = \frac{\sum_{k=1}^K W_\theta(\theta_{i,k}^j)\theta_{i,k}^j}{\sum_{k=1}^K W_\theta(\theta_{i,k}^j)}. \tag{2.6}$$

The PI^2 optimization-loop is illustrated in Fig. 2.2 (adapted from [STS12]). Hence, given a parameterized trajectory, PI^2 has the ability improve it step-by-step towards a certain goal that is described by the cost function S.

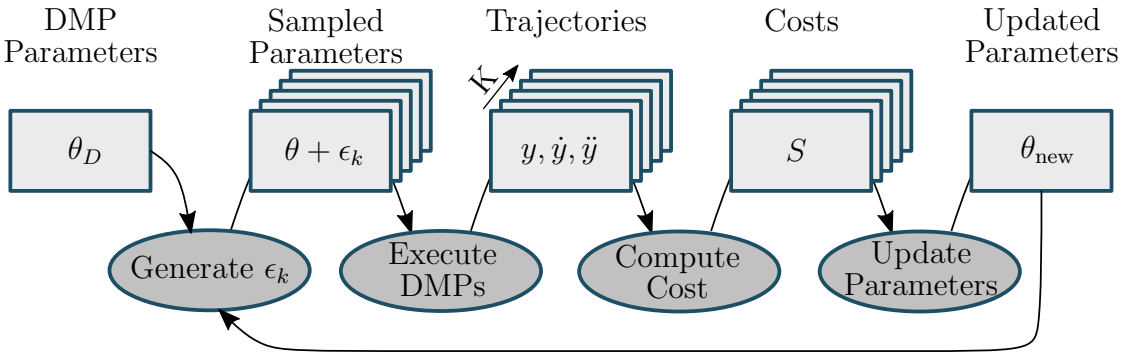


Figure 2.2: Optimization loop of PI^2 [STS12].

2.2.3 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 [Mor78]. 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.

2.3 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 divers 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 is a *terminal cost* and depends on the trajectory height H,

$$H = \min(z_p), \tag{2.7}$$

which is defined by a vector of weighted heights z_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,

$$z_p = \frac{h_p}{W_p}. \tag{2.8}$$

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 or when there are several obstacles of different heights. The choice of the observation points p depends on the relative position of obstacles between y_0 and g. Each observation point can be interpreted as a thin wall that the trajectory learns to avoid. The second cost term features an *immediate cost* represented by a *hinge loss* function,

$$S_{scope} = \sum_{t=1}^{T} \max(0, m + y_0 - \mathbf{y}_t), \tag{2.9}$$

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 is an *terminal cost* and ensures the goal precision of the trajectory,

$$S_{prec} = ||g - y_{end}||, \tag{2.10}$$

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,

$$S = -H + c_1 \cdot S_{prec} + c_2 \cdot S_{scope}, \tag{2.11}$$

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 representing the degree of curvature,

$$r_c = \frac{H}{l_D}, \tag{2.12}$$

which is constant for a specific set of DMP parameters θ_i.
Figure 2.3(a) illustrates an example of this process for two points $p1, p2$ with equal weights $W_p = [1, 1]$. Every one hundred iterations the current policy is plotted, starting from the linear trajectory (black). The dashed line marks the current r_c^j.

Figure 2.3(b) shows the evolution of the cost function. It decreases steadily, however has small peaks that illustrate that not every iteration j has lower cost than the previous iteration $j-1$. The cost for the goal precision S_{prec} is constantly smaller than 0.02 indicating that the goal precision of each generated trajectory is smaller than $0.02/c_1 = 0.002m$. The optimization 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 maximum ratio r_c^*. However, the relative length L where the trajectory maintains the height H is constant during one optimization. To allow more variations, we combine ten pairs of $p1$ and $p2$ in the experiment (see Sec. 3) creating trajectories that vary in steepness. When a point $p1$ is placed close to y_0 the trajectory must quickly increase to reach H, when $p1$ is placed in the center between y_0 and g the trajectory does not require that steep ascend. In the example (Fig. 2.3(a)), the points $p1, p2$ represent a length $L = |p1 - p2|$ and allow avoiding an obstacle of $\hat{L} = L$ requiring a moderate steepness of the trajectory slope.
Also complex shapes can be created reliably. Figure 2.4 illustrates an example where we reshape a linear trajectory in 3D with observation points $p_Z = [0.03, 0.06, 0.12]$ that measure the height in Z direction with corresponding weights $W_{p_Z} = [-0.5, 1, -0.3]$. Additionally, at points $p_Y = [0.04, 0.13]$ the height in Y direction is optimized with weights $W_{p_Y} = [1, 1]$. Ten consecutive runs achieve the termination condition $r_c^* = 1$ after $J \in \{9900, 16900\}$ iterations. The optimization is stable but requires more iterations and time compared to simpler shapes. A more detailed assessment of the computation effort for simple shapes is given in Section 3.2. Generally, it can be observed that the complexity of the trajectory shape depends mainly on the number of observation points and the required steepness.

2.3.1 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. Figure 2.5 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.

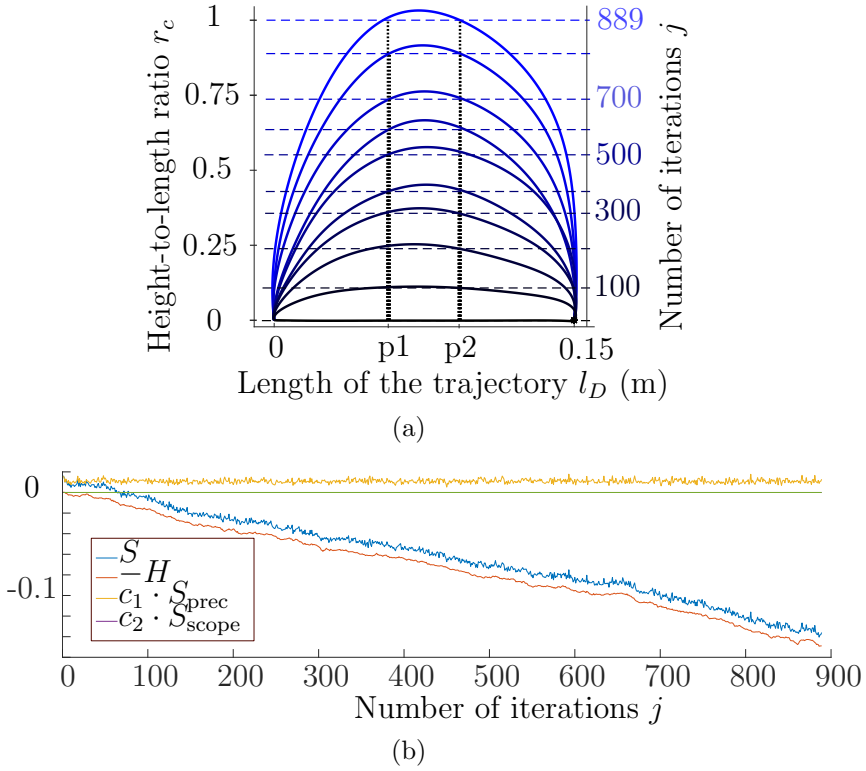


Figure 2.3: Evolution of the trajectory (a) and the cost (b) during one PI^2 optimization.

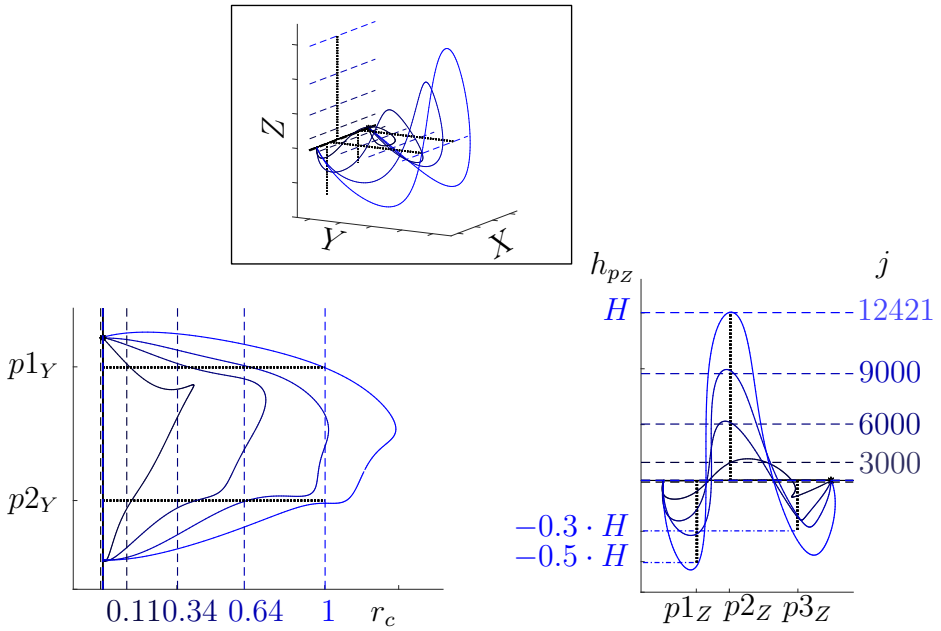


Figure 2.4: PI^2 optimizations for a complex trajectory. At each plotted iteration, the heights increase and keep similar proportions as specified by the weights W_Y and W_Z.

Despite the visible correspondence, the evolution provides no indication for a trivial solution such as, for example, multiplying all parameters with some scaling factor.

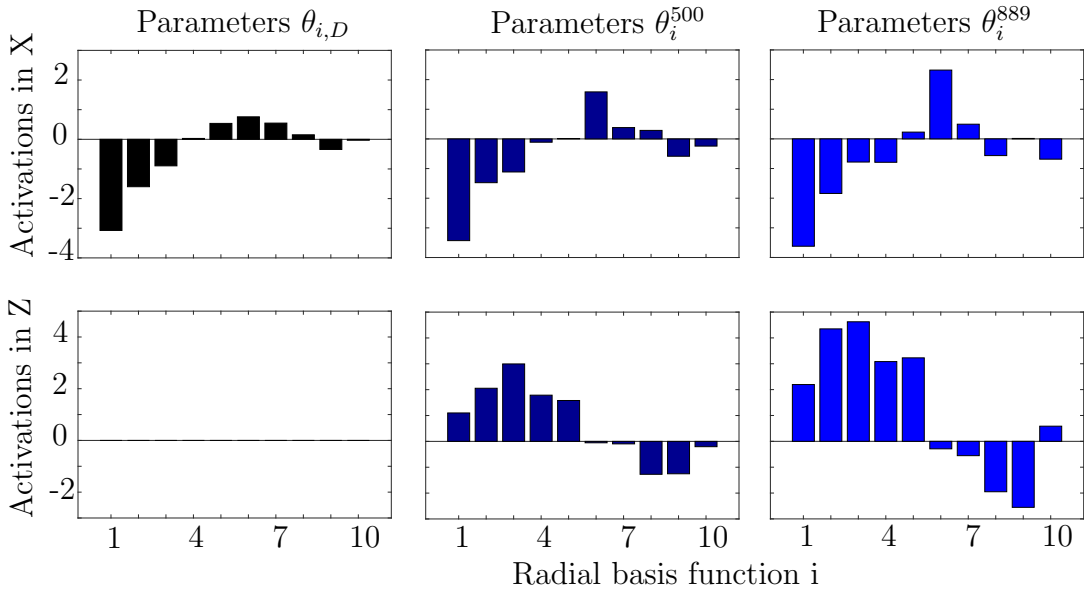


Figure 2.5: Evolution of the DMP parameters with PI^2.

Chapter 3

Evaluation

To assess the validity of our approach with respect to the state of the art, we use the same benchmark scenario as in [DKCK18], where colored cubes are placed in a grid configuration where the cells have a distance of $v_X = 0.1m$ in X direction and $v_Y = 0.15m$ in Y direction. The task consists of picking and placing cubes in an ordered manner until the goal configuration is reached (see Fig.3.1). This task 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 [DKCK18], we use a simulated environment for the experiments, implemented using the physically realistic simulator V-REP [RSF13]. Figure 3.1 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 3.1 shows an example planning operator in the PDDL.

Several trajectories with different shapes must 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 that is able to shape the trajectory for every particular configuration of cubes and action, using the approach described in Sec. 2.3. To learn the action

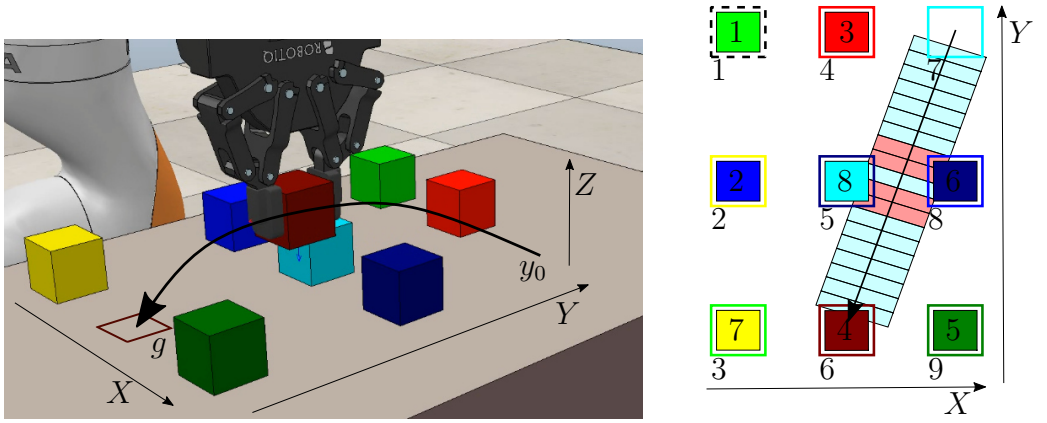
```
(:action pickplace
:parameters (?from ?to ?cube)
:precondition (and (on ?to air)
(on ?from ?cube))
:effect (and (on ?from air) (on ?to ?cube)
(not (on ?to air)) (not (on ?from ?cube))))
```

Table 3.1: The pick-and-place task is decomposed into sequences of this *pickplace* action.

policy, the following steps are applied in an offline training process after the action demonstration (in red in Fig. 2.1):

- Identify observation points.
- Define cost function.
- Generate DMPs for different obstacles.
- Learn action policy (neural network).

Afterwards, the learned action policy is evaluated and compared to [DKCK18].



(a) Simulation of a *place* operation

(b) Observation of the cube poses in MATLAB

Figure 3.1: Simulation of the symbolic action `pickplace cell7 cell6 cube4`.

3.1 Obstacle Description

To allow a representation of various situations where different lengths L at height H 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. 3.2). Each border $b = 1, ..., B$ can be chosen as observation point p. The observation width $w_A = h_{cube} + 0.03m$ is chosen empirically 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 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 3.3 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.

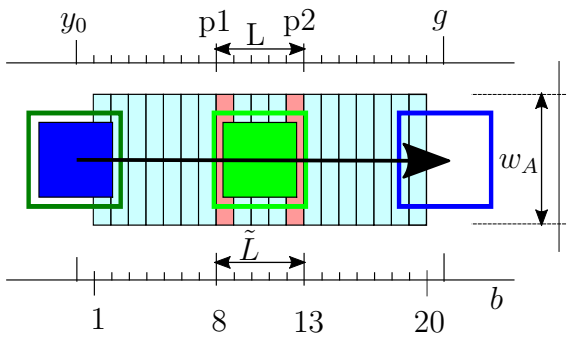


Figure 3.2: Discretized observable area for describing an obstacle. In this example $\tilde{L} = (8, 13)$. A trajectory that avoids this obstacle is generated with a minimum height H between $p1$ and $p2$.

3.2 PI^2 Optimizations

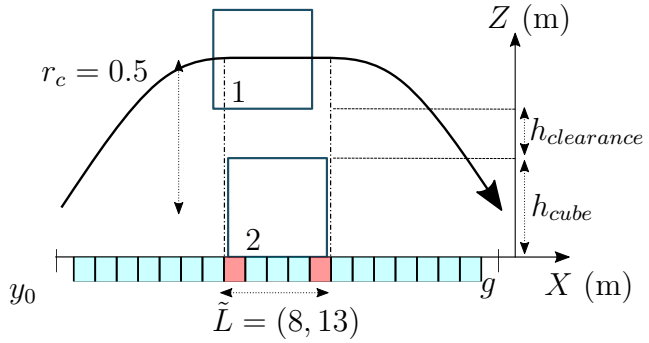
The optimization process is applied to each of the $n_{\tilde{L}} = 10$ symmetric combinations of $\tilde{L}$. To this end, we derive $p1$ and $p2$ for each of the $n_{\tilde{L}}$ optimizations, where

$$p1 = \frac{l_D \cdot \tilde{L}(1)}{B + 1}, \tag{3.1}$$

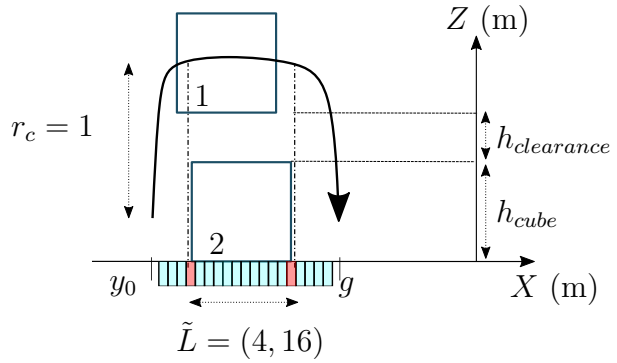
and similarly, $p2$ is derived with $\tilde{L}(2)$. We constrain the generated trajectories for all optimizations to be contained along the X axis within a margin $m = 0.01m$ from $y_{0,D}$ and g_D respectively,

$$\begin{aligned} S_{scope} = &- \sum_{t=1}^{T} \min(0, m + \mathbf{y}_{X,t} - y_{0,X}) \\ &- \sum_{t=1}^{T} \min(0, m + g_X - \mathbf{y}_{X,t}). \end{aligned} \tag{3.2}$$

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 3.4 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



(a) Placing when y_0 and g are further away from the obstacle



(b) Placing when y_0 and g are close to the obstacle

Figure 3.3: 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 $||y_0 - g||$.

effort to reach the same height. The deviation of the effort is also larger for more complex slopes. All 100 optimizations achieve r_c^* within one minute verifying the expected stability of the PI^2 method.

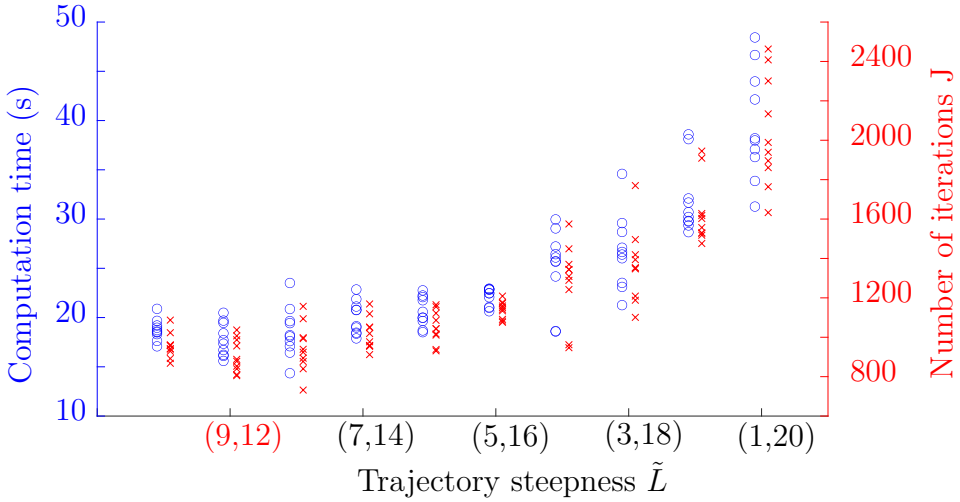


Figure 3.4: Ten PI^2 optimizations are performed for each trajectory steepness $\tilde{L}$. For each optimization the final computation time (blue) and the number of iterations (red) are plotted. All optimizations reach the termination condition r_c^* within one minute, validating the expected numerical robustness of PI^2.

3.3 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 (2.12) and the steepness $\tilde{L}$ 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 represents the tuples of $\tilde{L}$ as one value,

$$r_L = (|\tilde{L}(2) - \tilde{L}(1)| + 1)/B, \tag{3.3}$$

resulting in $r_L = 0.1, 0.2, ..., 1$ for our discrete selection. 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,

$$d_g = \frac{S_{prec}}{l_D}, \tag{3.4}$$

where S_{prec} is defined in (2.10) and the deviations in height d_H,

$$d_H = r_c - \frac{H}{l_D}, \tag{3.5}$$

where H is defined in (2.7). To this end, Fig. 3.5 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 0.27% and a maximum of 1.6%, hence, no collisions of the gripper with the surface occur here. The mean height deviation is 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.

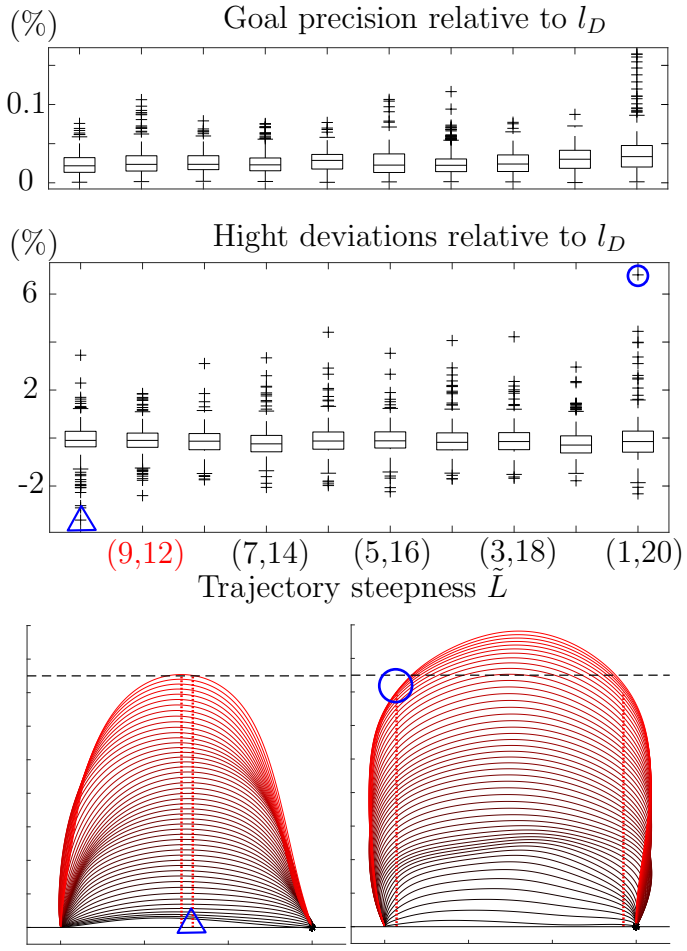


Figure 3.5: Height deviations d_H and goal precision d_g of 5000 reproduced trajectories using neural networks, illustrating the two most significant outliers.

3.4 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 $r_L = 1$

($\tilde{L} = (1, 20)$) 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 depends on whether and where an obstacle must be avoided. Figure 3.6 illustrates the range of different trajectory shapes that the action policy can select. A 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 3.2 shows the parameters of the experiment. All 159 *pickplace* actions are successfully executed with the first attempt.

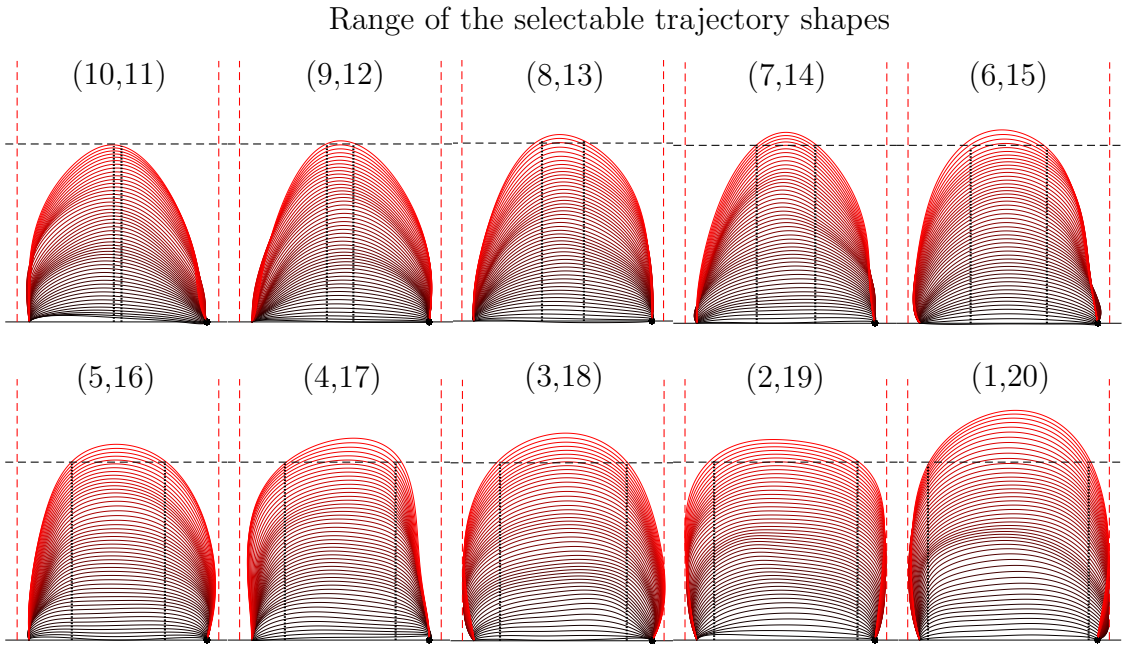


Figure 3.6: Depicted are the trajectory shapes for the ten combinations of $\tilde{L}$. For each $\tilde{L}$, 50 uniformly distributed r_c in the range $[0, 1]$ are plotted.

3.4.1 Computation Times

Figure 3.7 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. [DKCK18], 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

DMP parameters		
Number of time steps	T	200
Duration (s)	$\hat{\tau}$	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. 3.2)	w_A	$h_{\text{cube}} + 0.03$
Grasping height	h_{grasp}	0.02
Clearance height	$h_{\text{clearance}}$	$10\% \cdot \|\|y_0 - g\|\|$

Table 3.2: Parameters for the pick-and-place task.

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 $0.01s$, for ten symbolic actions $10s$. 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 times for generating the demonstration, generating the trajectory samples and training the neural network accumulate to nine minutes on average (Tab. 3.3), performed on a Intel i7-4790 CPU @ $3.60GHz$, $16GB$ RAM.

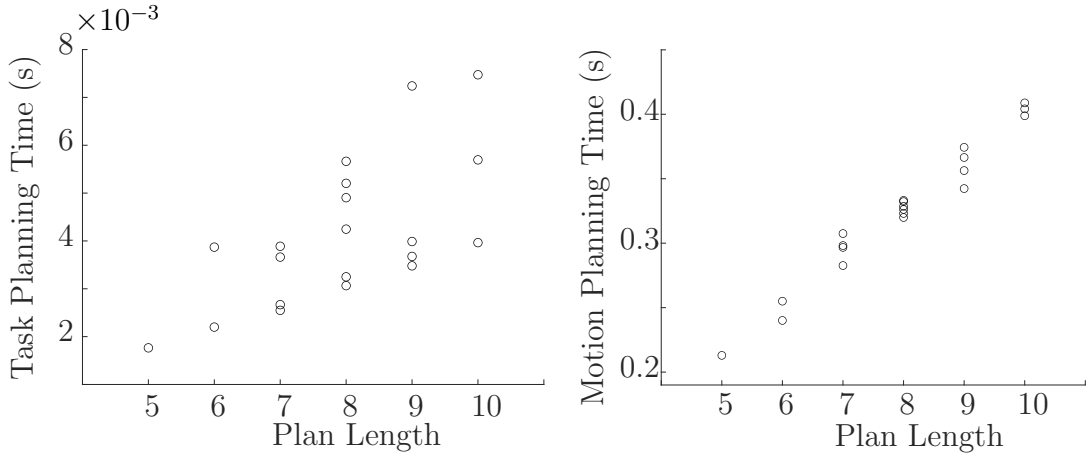


Figure 3.7: Computation times of the task planner and the motion planner for 20 random runs.

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

Table 3.3: Training times for the pick-and-place task.

3.4.2 Analysis of the Performed Actions

In the experiment, the obstacle height $\hat{H}$ is constant and hence, not many trajectory variations are expected. However, considering the height relative to the trajectory length l_D creates more divers situations. Figure 3.8 shows all pick-and-place operations during the 159 *pickplace* actions. There are 25 different combinations of inputs for the neural network (r_c, r_L). The performed trajectories can be assigned to three different scenarios: placing with or without obstacles and picking. When placing a cube without an obstacle the steepness of a trajectory does not matter and a random value for r_L is chosen. The grid is relatively large compared to the cube size which explains the large number of *place* operations that do not require obstacle avoidance. Also noticeable is the small range of $r_c \in [0.1, 0.3]$. The clearance height marks the lower bound. The upper bound is significantly smaller than the maximum learned $r_c^* = 1$.

3.5 Generalization Ability to Varying Object Sizes

Performing the task with the same cubes requires only five different *place* operation with obstacles. In an additional step, we evaluate the same action policy on the pick-and-place task randomly varying the obstacle length $\hat{L}$ and the obstacle height $\hat{H}$ (same for all cubes) and inducing random noise e_v to the locations of the cells (each individually). Table 3.4 shows the range of $\hat{H}$, $\hat{L}$ that are arbitrarily chosen and e_v, which depends on $\hat{L}$ to avoid overlapping. We perform the same 20 tasks initializing the block size $\hat{L} \times \hat{L} \times \hat{H}$ and the cell positions at the beginning of each task. Again, all 159 performed actions are successful at the first attempt. Figure 3.9 shows the analysis of the performed actions. This time, the action policy selects 76 distinct trajectory shapes. Especially the required *place* operations with obstacles are significantly more divers with 48 distinct trajectories compared to the analysis with equally sized cubes and constant cell positions throughout the 20 runs. The maximum r_c is also significantly closer to the learned r_c^*. There are seven more *place* operations without obstacles than in the previous analysis indicating that the varying size or cell position creates enough space to pass the object instead of having to avoid it. The *pick* operations are similar to the previous analysis, since the grasp height is the same. However, due to the random cell places, the distance between two cells varies and thus, also the trajectory shape. The blue

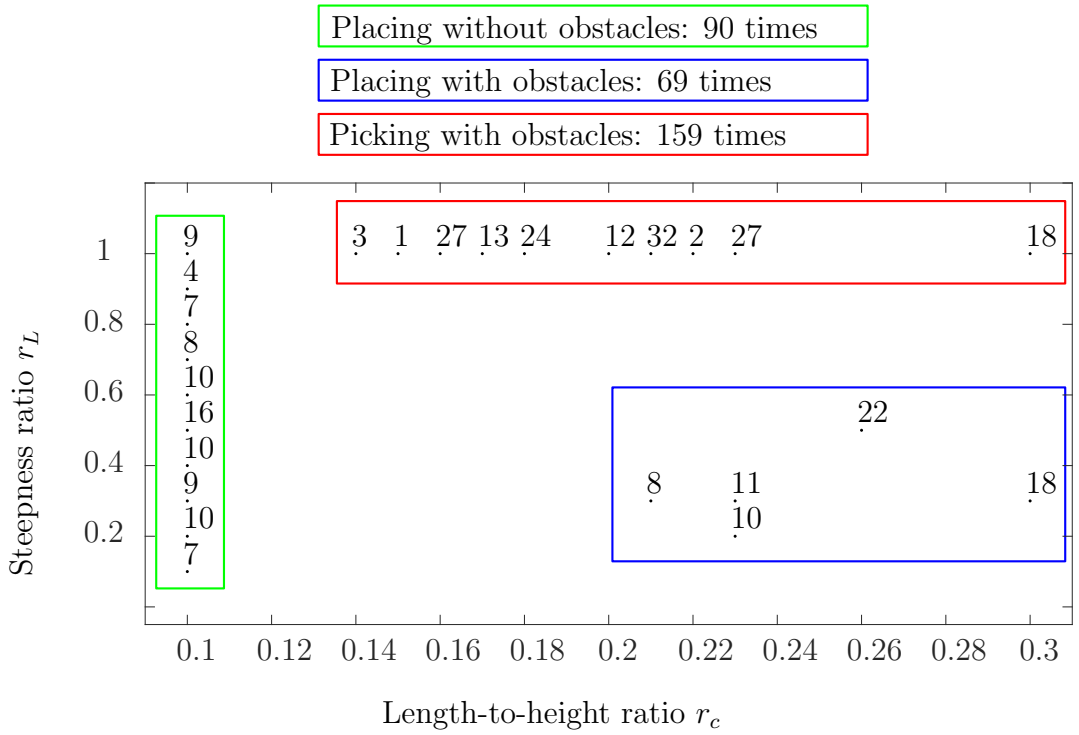


Figure 3.8: The analysis of the performed pick-and-place operations shows a point for each trajectory shape that is generated in the experiment. There are 159 *place* operations and 159 *pick* operations. In total, there are 25 distinct input combinations r_c, r_L representing 25 different trajectory shapes. At each combination the frequency of the appearance is placed on top of the point.

Variation range of the parameters	
Obstacle height $\hat{H}$ (m)	$[0.8, 2.8] \cdot h_{\text{cube}}$
Obstacle length $\hat{L}$ (m)	$[0.6, 1.2] \cdot h_{\text{cube}}$
Cell position noise e_v	$\pm(v_X/2 - \hat{L} - 0.001)$

Table 3.4: Range of variations to the object parameters.

asterisks mark the performed operations of one run that features ten distinct *place* operations and seven distinct *pick* operations. The randomly initialized dimensions of the blocks are $\hat{L} = 0.95 \cdot h_{\text{cube}}$, $\hat{H} = 2.56 \cdot h_{\text{cube}}$, the positions of the cells deviate by up to $e_v = 0.011$. Screenshots of this task from the simulation in V-REP and the corresponding MATLAB figures are presented in Appendix A.

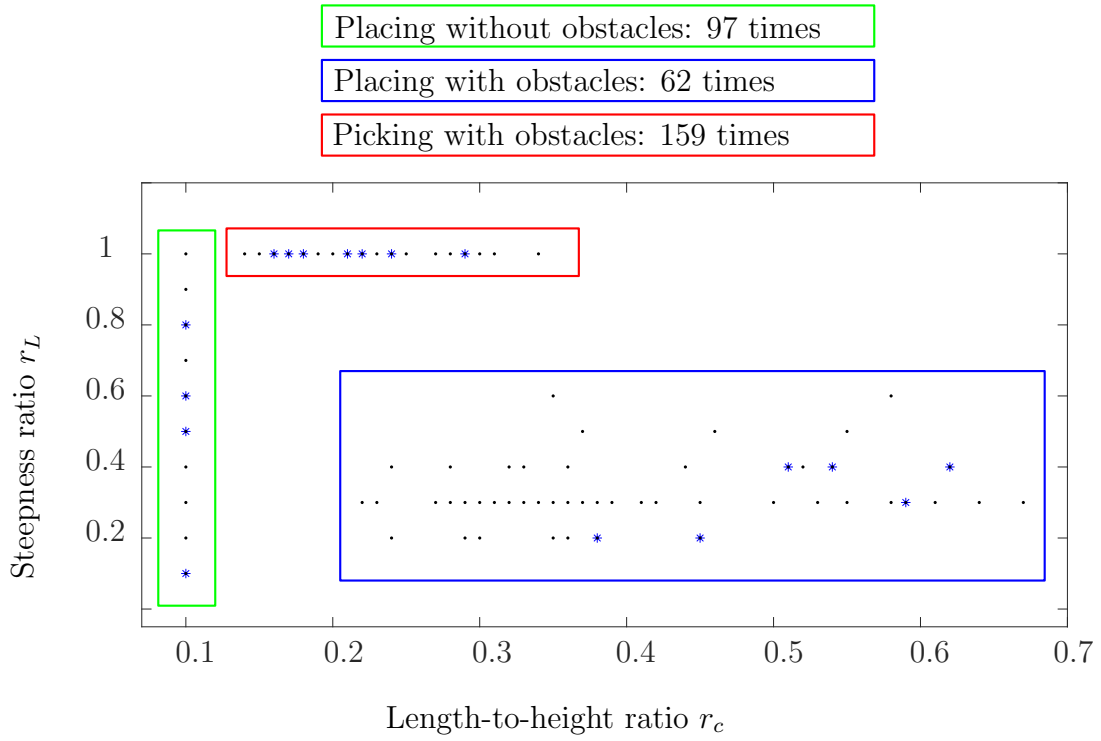


Figure 3.9: The same tasks are performed with varying object sizes and positions. Again, a point in the figure represents one trajectory shape that is generated in the experiment. This time, there are 76 distinct input combinations r_c, r_L representing 76 different trajectory shapes. The blue asterisks mark the operations that are performed in one task that is illustrated in Appendix A.

Chapter 4

Discussion

Comparison to IDTMP The proposed TAMP framework solves the evaluation task more efficiently than the compared IDTMP approach by Dantam et al. [DKCK18]. Especially for increasing plan lengths the computation times for the combined planning decreases by orders of magnitudes due to the IDTMP's exponentially growing task planning times. Hence, our approach is suitable for TAMP problems with large plan lengths. This efficiency comes at a cost of limited connectivity of the task planner and the motion planner. Problems, where the motion execution of the first subtask depends on another subtask, e.g. placing two objects on the same area with limited space. Those constraints must be handled by the task planner providing the appropriate goal position already at the beginning. The proposed approach is then able to select a trajectory shape that avoids collisions.

Limitation of a Constant Gripper Orientation In the experiment the orientation of the gripper is constant, consistently opening and closing in Y direction. This constrains the configuration of the cubes to positions that keep certain space to the next cube along the Y axis. Introducing a second grasping pose rotated by 90 degree improves the flexibility and can be solved by introducing another predicate into the task planner (e.g. `left cube1 cube2`) or by observing the direct surroundings of each cube (similar to the observation of obstacles between y_0 and g) and reacting with initializing the DMPs with one of the two orientations.

Complexity of Generated Trajectory Shape and Adaption to Dynamic Changes We apply the optimization with PI^2 to smoothly reshape the trajectory towards an engineered goal. The more complex and precise a desired trajectory shape defined, the more effort must be placed in creating the reward functions for the optimization. Balancing the costs becomes harder. The $\min(z_p)$ ensures that each value is regarded in the optimization process. Otherwise, when $-\infty$ marks the minimum cost, the value that achieve the highest rate of decrease is favored by the algorithm. For future work, another way of increasing the complexity of the shape could be achieved by coupling DMPs during the execution adding several different

primitive shapes from our approach. Additionally, this has potential to improve the ability to adapt to dynamic changes. The DMPs used in this work do not use the proposed *canonical system* [INH+13] that allows coupling of DMPs. Therefore, our trajectories keep approaching the goal pose without reacting to any dynamic inputs from the environment. The safety of humans can be guaranteed by sensors that stop the robot when a collision occurs, however, for human-robot collaborations, dynamic motion adaption is important to permit practicability.

Supervised Learning on Dependent Data The proposed approach learns the decision making in a supervised way on the data that is continuously generated during the PI^2 optimization. Each optimization loop builds upon the previous result (Fig. 2.2). During the training of the neural network, there is no independent data for a validation or test set that creates the generalization ability, hence, the neural network overfits to the training data. Each of the ten optimizations for $\tilde{L}$ are independent. The trained network shows no exploration ability and also no useful interpolation results for $r_L \in [0.1, 1]$. Interpolation regarding the ratio r_c is possible. A data set of independent data can be generated by taking only a few samples per optimization run. However, instead of generating a data set of 1000 samples within one minute, 1000 independent data samples may take about eight hours. For future work, it might be interesting to evaluate the performance when trained on independent data. Certainly, interpolation between several continuous variables could enable the action policy to generalize to more divers scenarios.

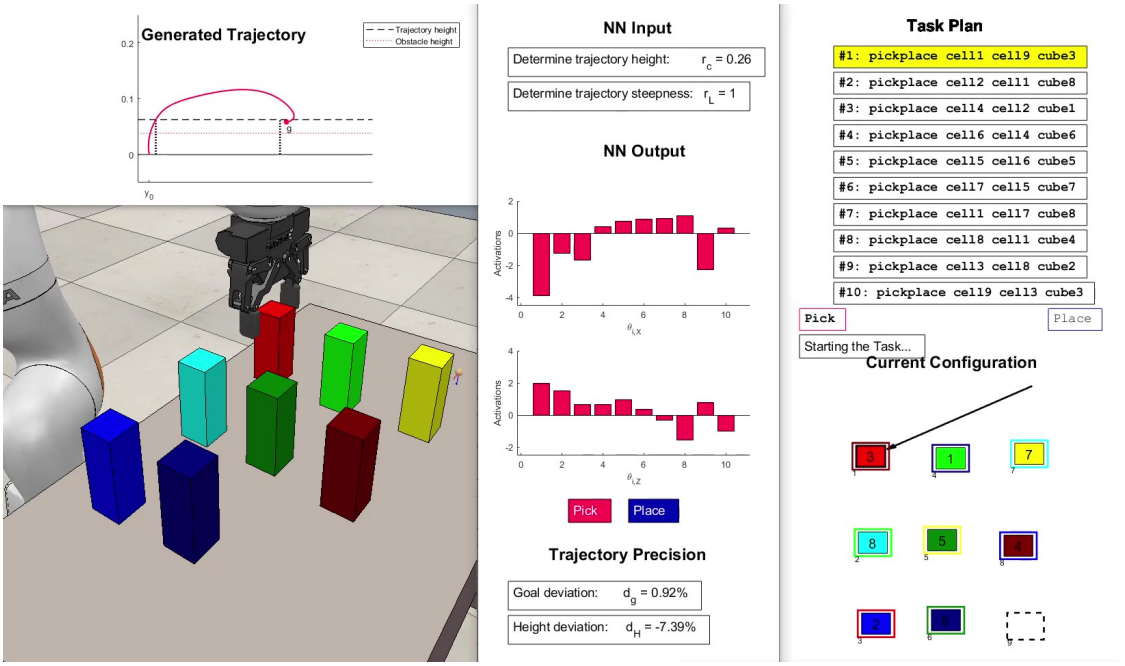
Chapter 5

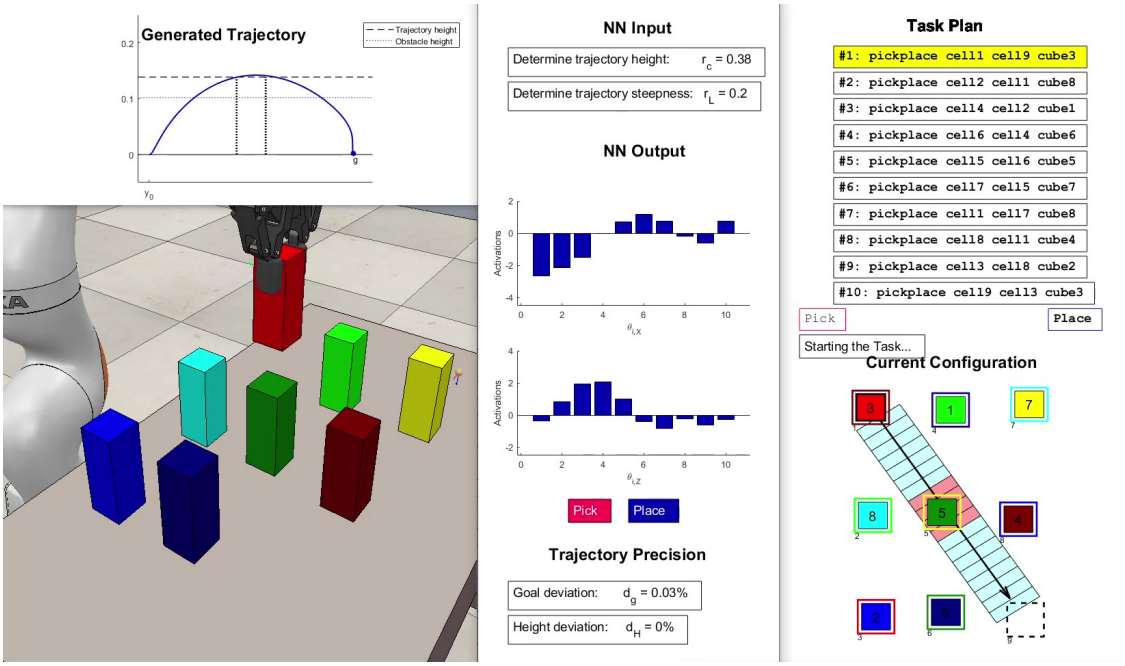
Conclusion

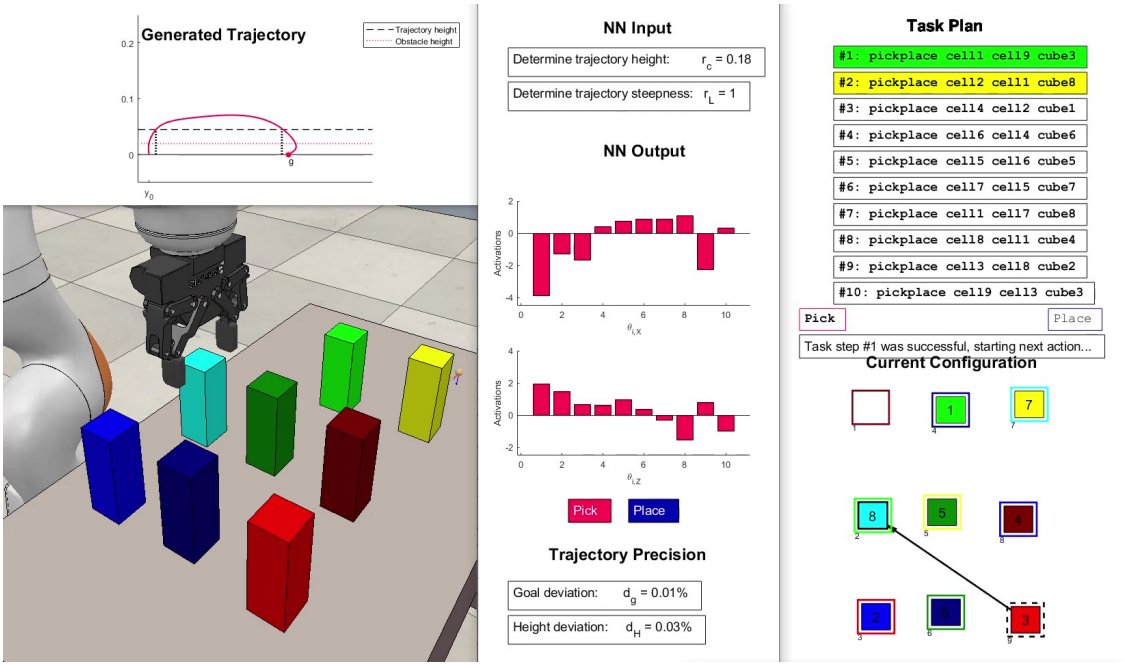
The proposed TAMP framework with a motion planner based on the combination of LfD and RL reduces the computation times during an experiment by more than one order of magnitude compared to a search-based framework [DKCK18] after nine minutes of training. This makes the proposed approach also suitable 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. In future work, we aim at improving the integration of different optimizations to also allow meaningful interpolations between different trajectory shapes. Furthermore, the coupling ability of DMPs [INH+13] might have the potential to combine our generated shapes to more complex trajectories and additionally enhance the adaption to a dynamic environment.
Task and motion planning will attempt problems that humans face every day. From creating shopping lists based on meals that one would like to prepare in the next week to carefully breaking eggs to scramble them in a pan. The variation of tasks that humans perform routinely is vast and challenging to implement the required functionality into a machine. However, considering all the tasks that humans *must* do rather than *want* to do, there is enormous potential to improve human life with technology that is capable of planning.

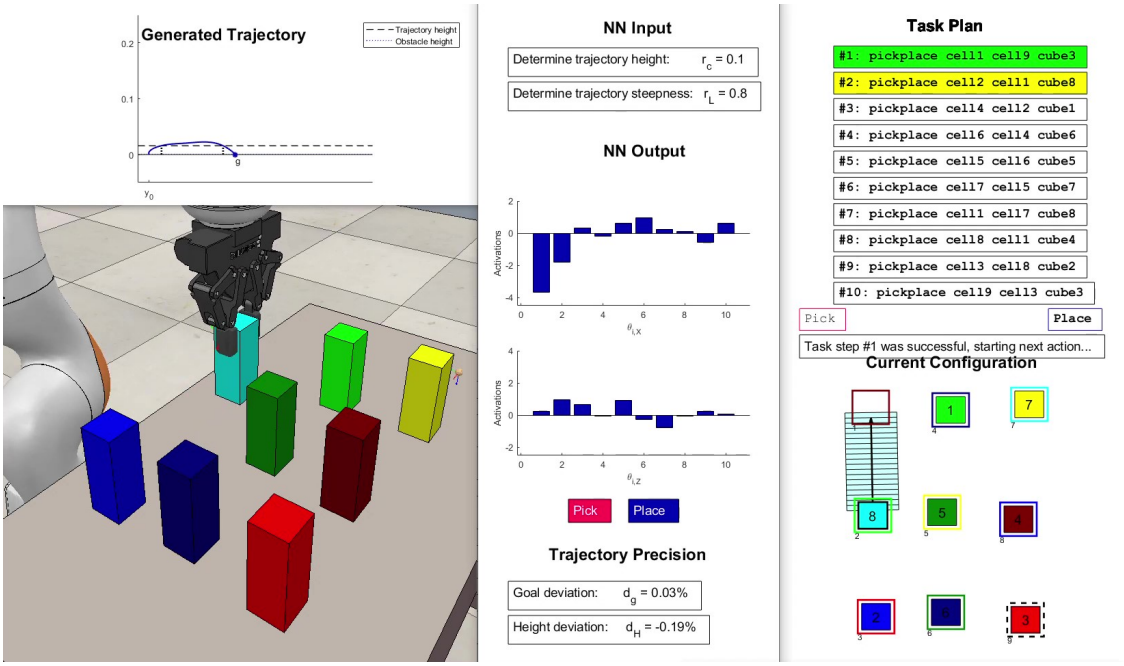
Appendix A

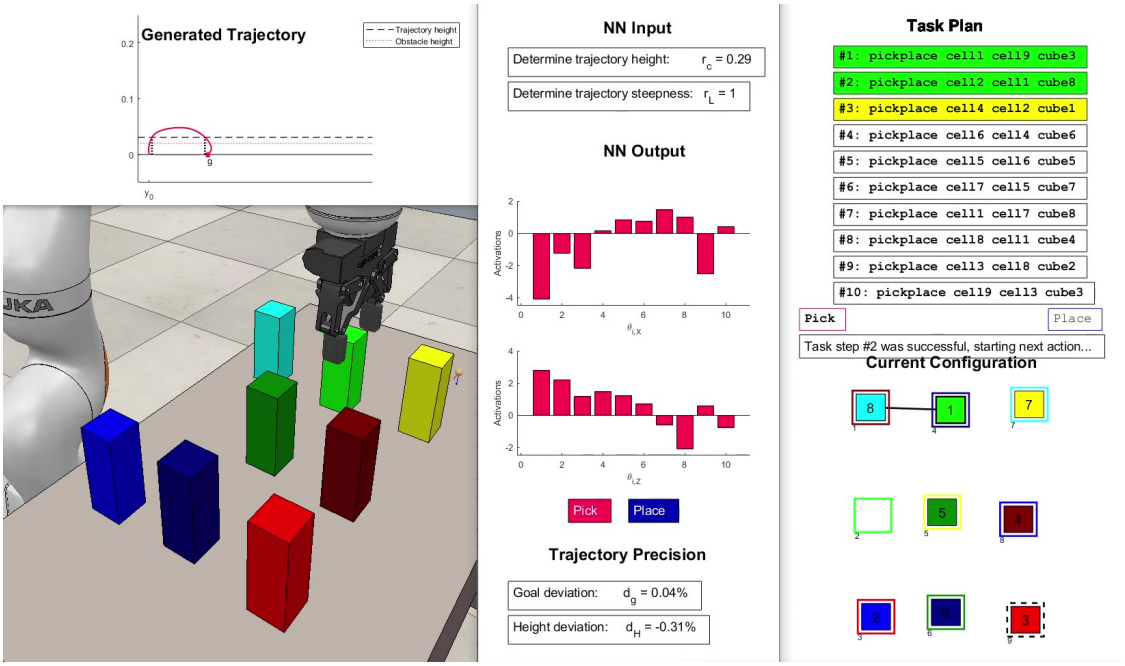
Complete Execution of a Task Example

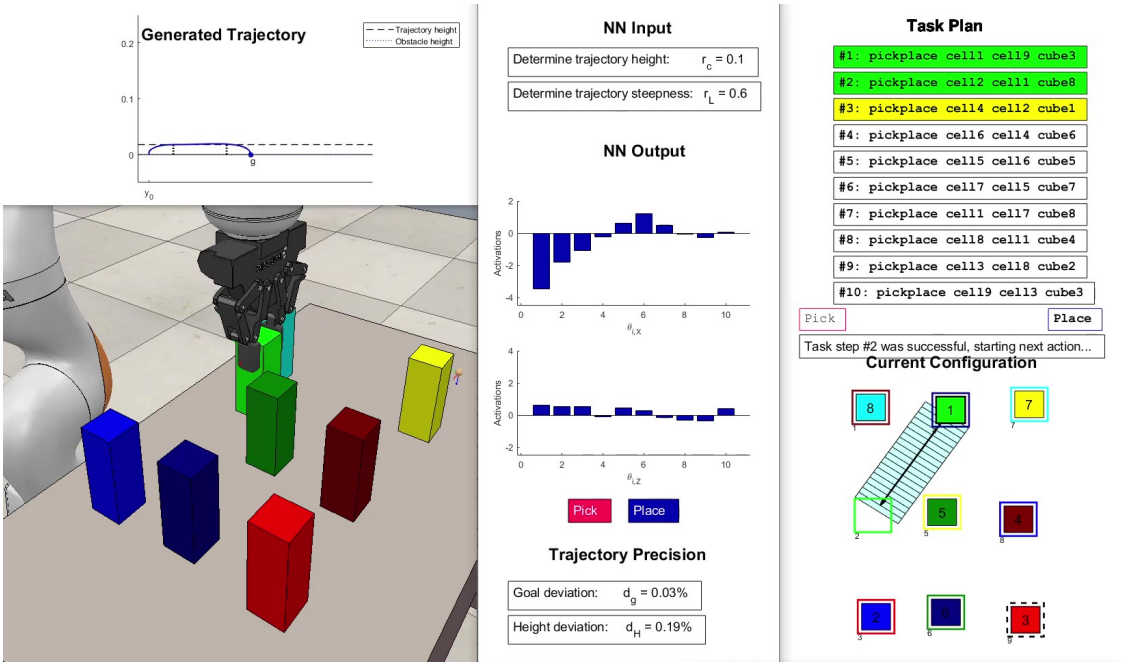

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.26
Determine trajectory steepness: r_L = 1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.92%
Height deviation: d_H = -7.39%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell2 cell1 cube8
#3: pickplace cell4 cell2 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Starting the Task...
Current Configuration

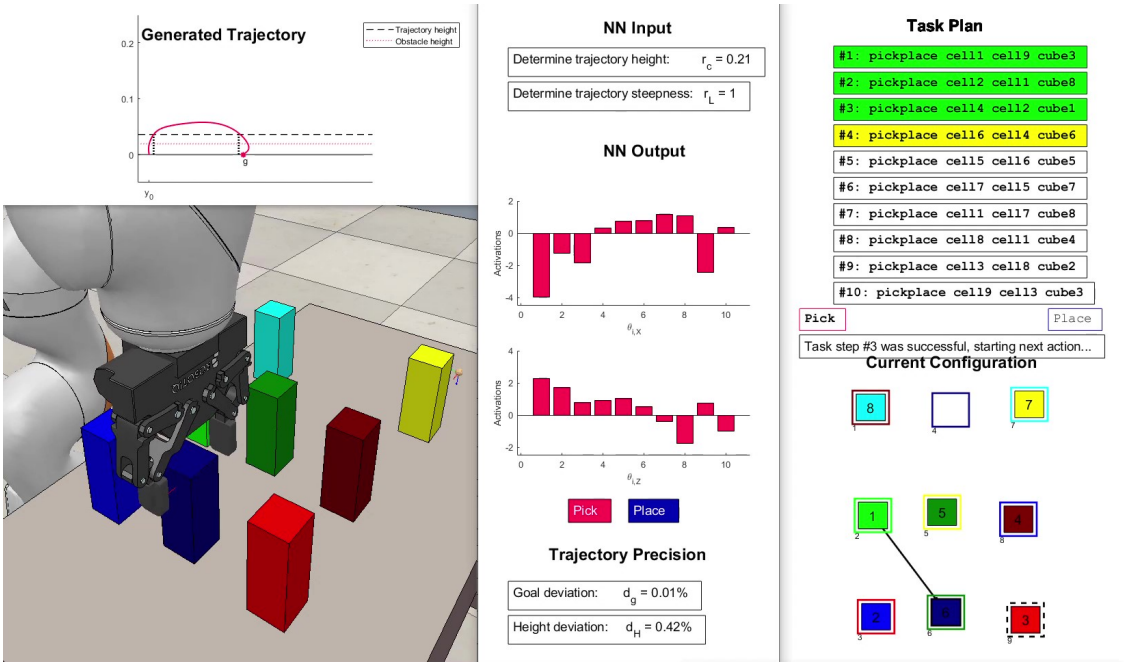

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.38
Determine trajectory steepness: r_L = 0.2
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.03%
Height deviation: d_H = 0%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell2 cell1 cube8
#3: pickplace cell4 cell2 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Starting the Task...
Current Configuration

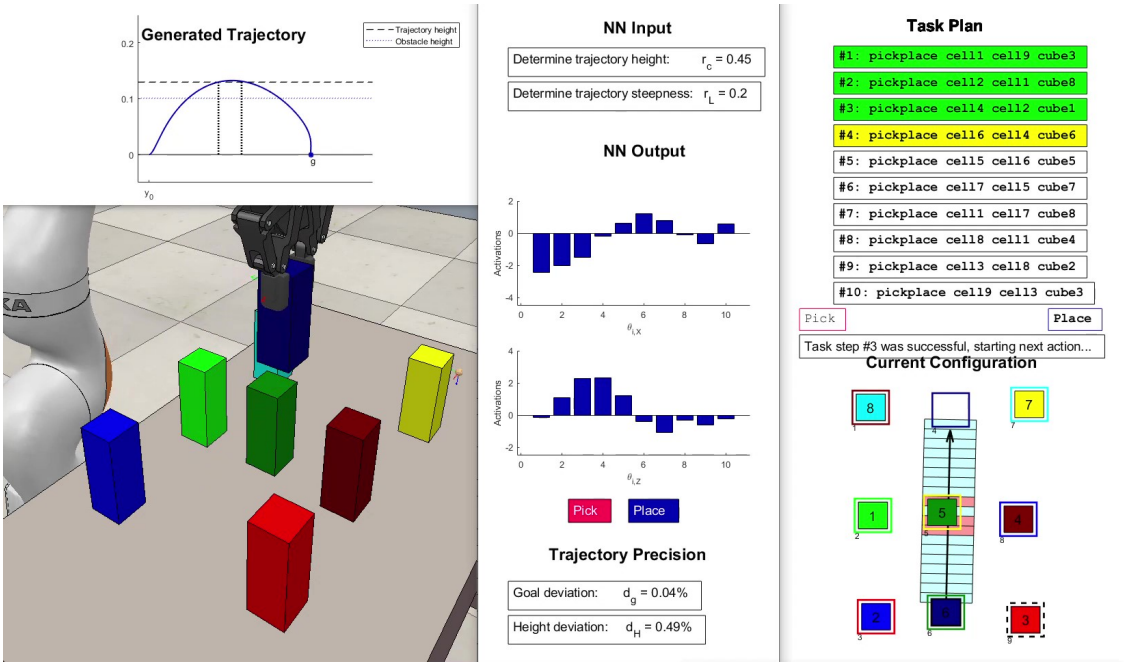

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.18
Determine trajectory steepness: r_L = 1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.01%
Height deviation: d_H = 0.03%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell2 cell1 cube8
#3: pickplace cell4 cell2 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #1 was successful, starting next action...
Current Configuration

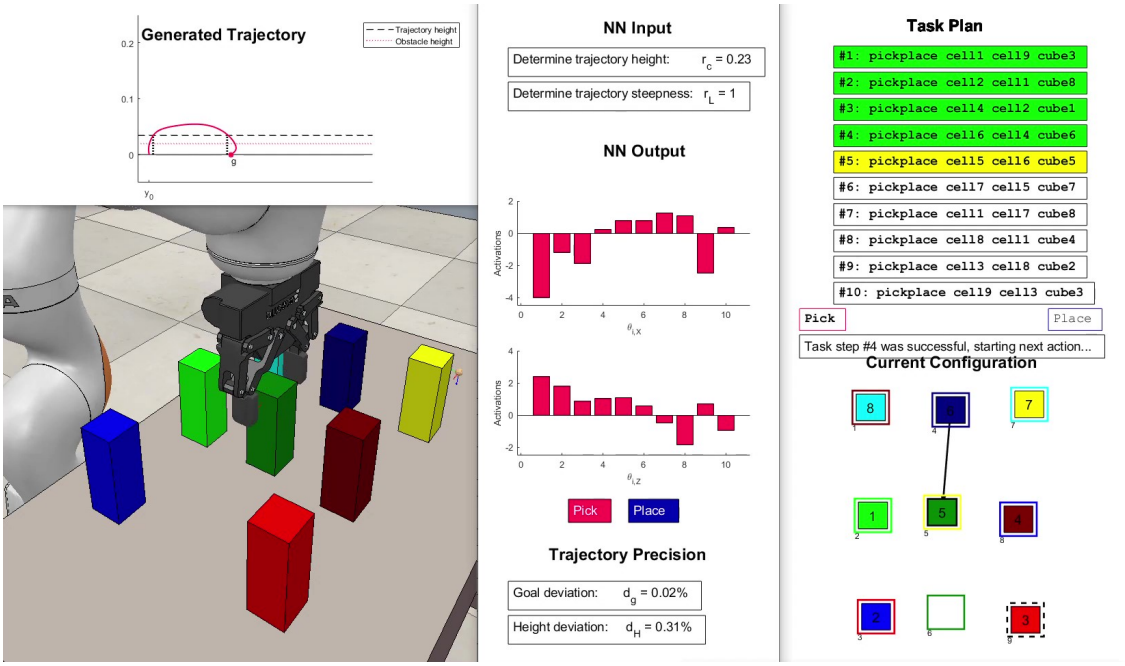

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.1
Determine trajectory steepness: r_L = 0.8
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.03%
Height deviation: d_H = -0.19%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell2 cell1 cube8
#3: pickplace cell4 cell2 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #1 was successful, starting next action...
Current Configuration

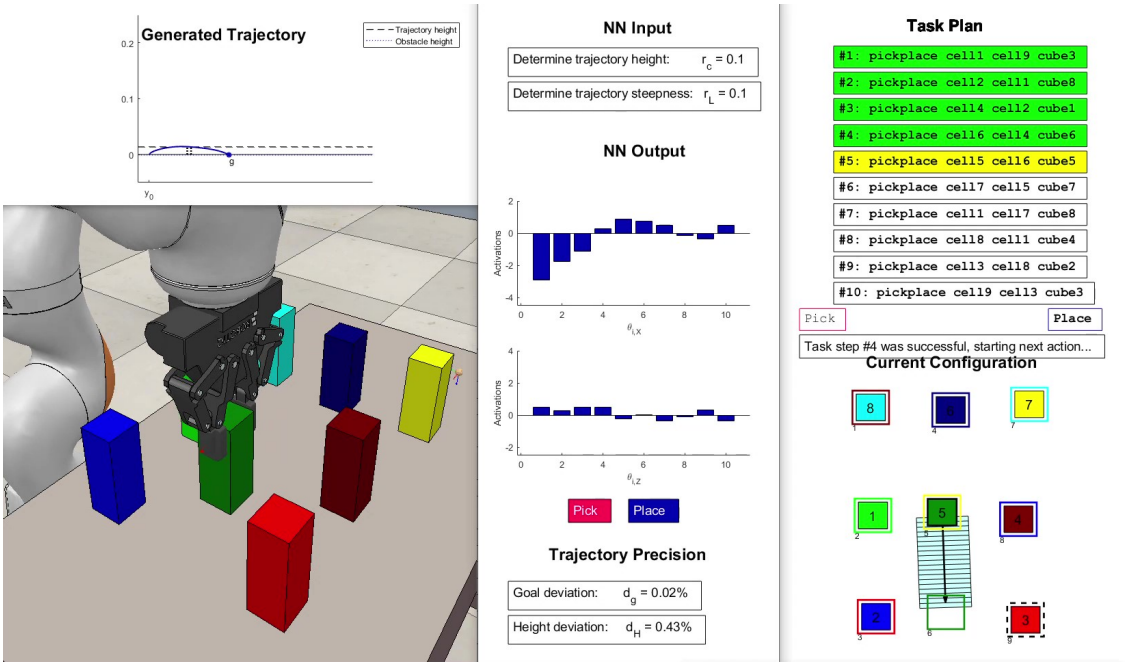

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.29
Determine trajectory steepness: r_L = 1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.04%
Height deviation: d_H = -0.31%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell4 cell12 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #2 was successful, starting next action...
Current Configuration

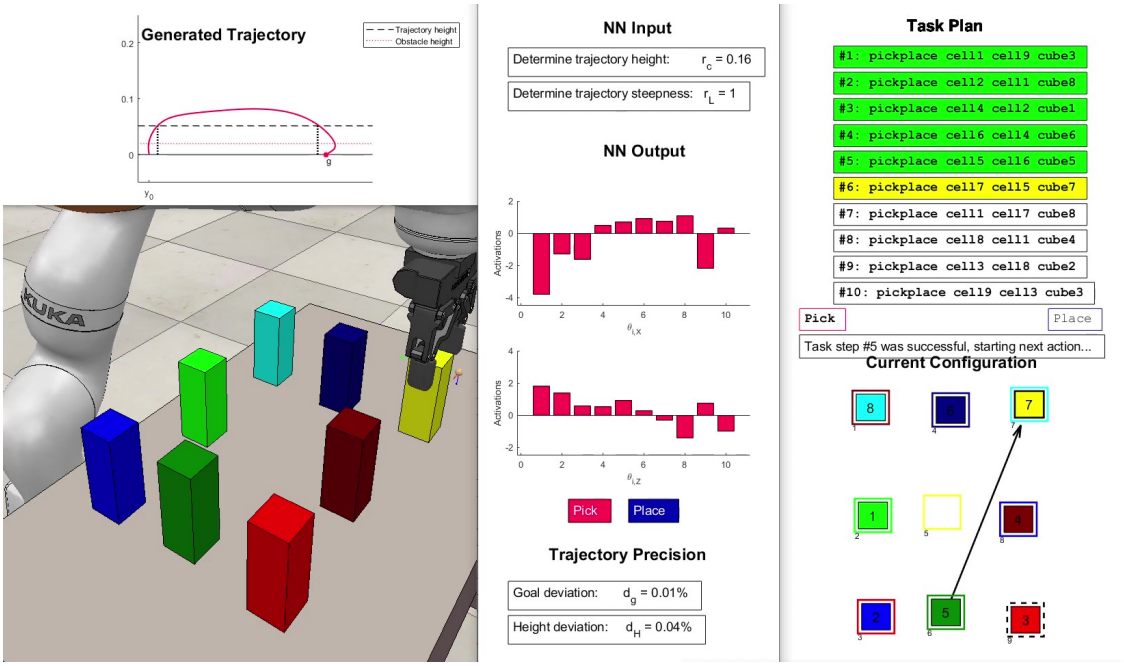

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.1
Determine trajectory steepness: r_L = 0.6
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.03%
Height deviation: d_H = 0.19%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell4 cell12 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #2 was successful, starting next action...
Current Configuration

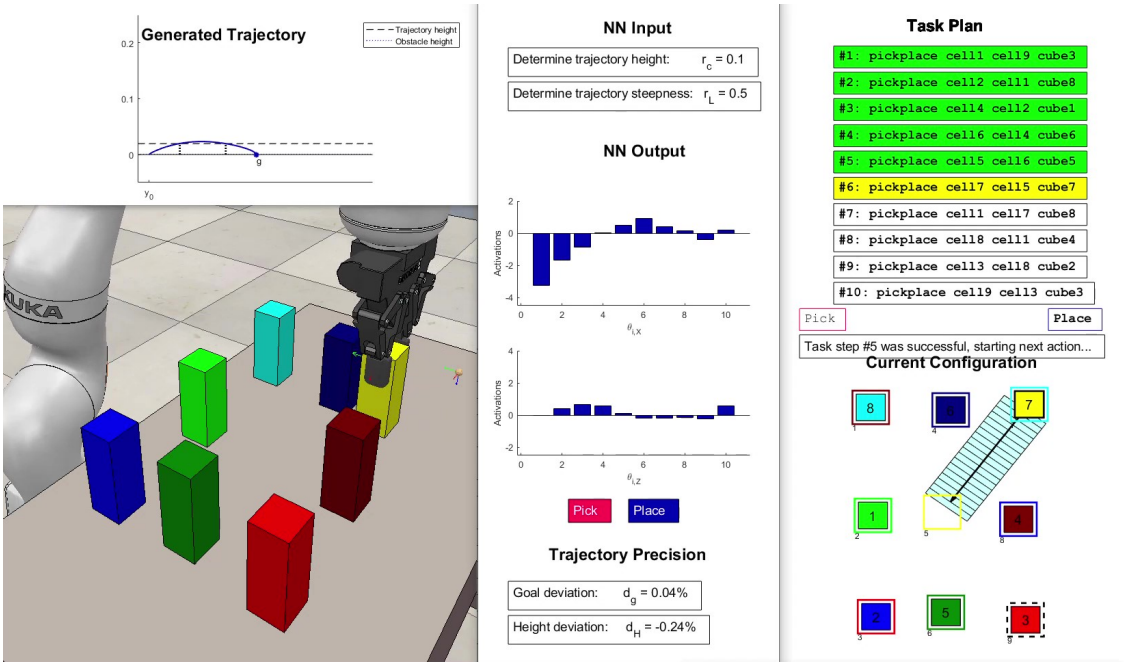

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.21
Determine trajectory steepness: r_L = 1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.01%
Height deviation: d_H = 0.42%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell14 cell12 cube1
#4: pickplace cell16 cell14 cube6
#5: pickplace cell15 cell16 cube5
#6: pickplace cell7 cell15 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell13 cell8 cube2
#10: pickplace cell9 cell13 cube3
Pick
Place
Task step #3 was successful, starting next action...
Current Configuration

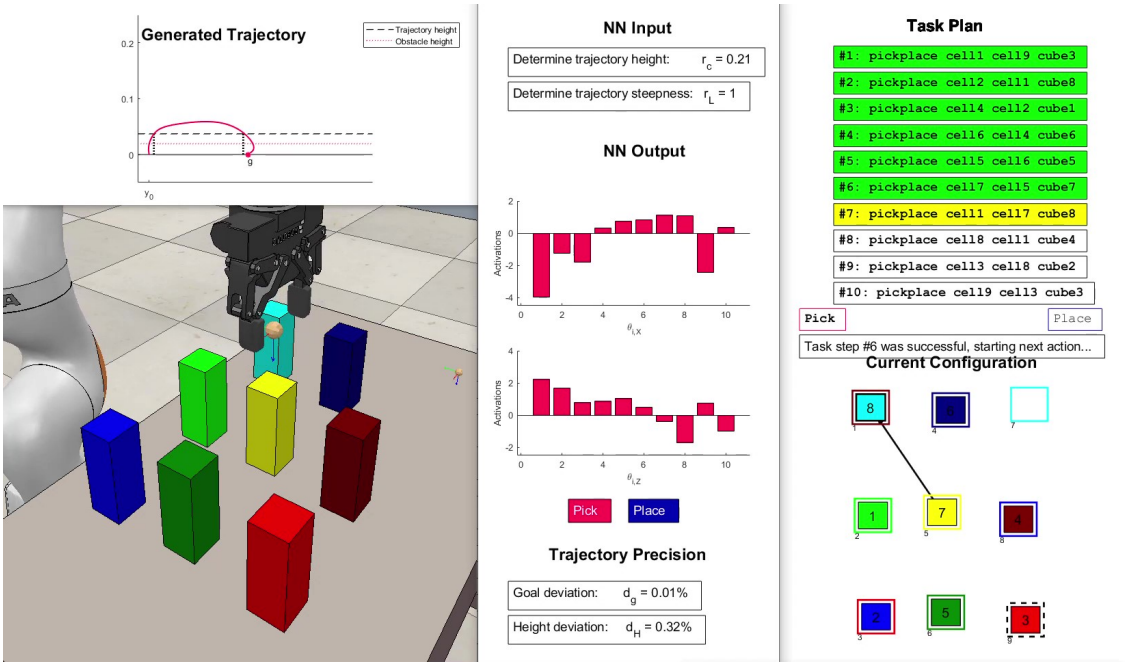

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.45
Determine trajectory steepness: r_L = 0.2
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.04%
Height deviation: d_H = 0.49%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell14 cell12 cube1
#4: pickplace cell16 cell14 cube6
#5: pickplace cell15 cell16 cube5
#6: pickplace cell7 cell15 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell13 cell8 cube2
#10: pickplace cell9 cell13 cube3
Pick
Place
Task step #3 was successful, starting next action...
Current Configuration

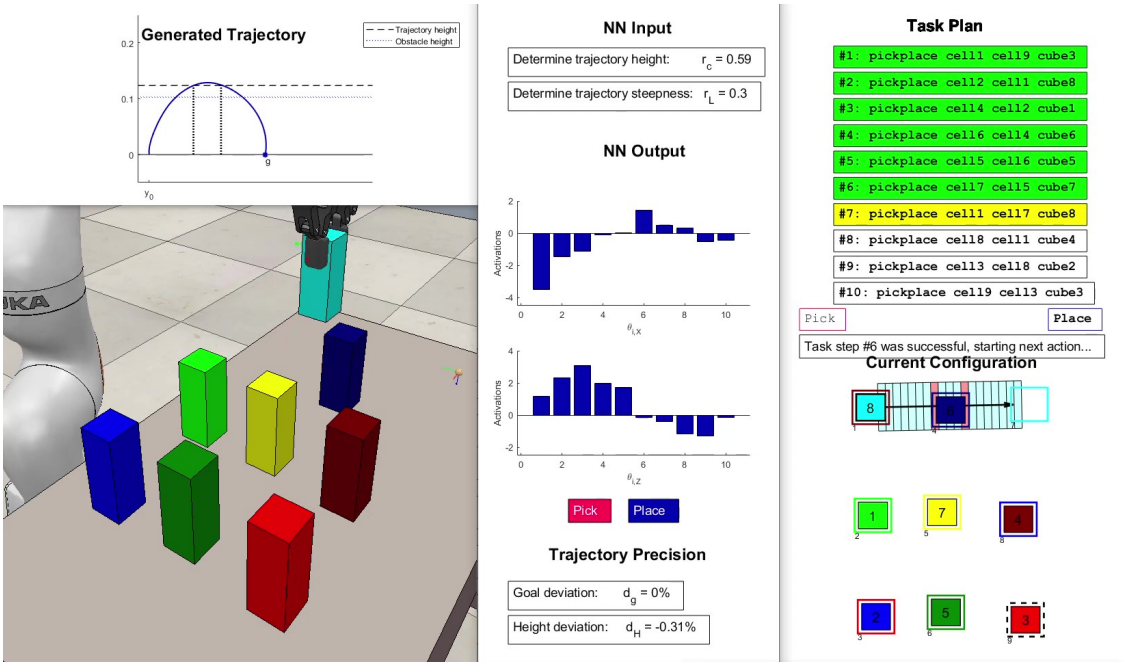

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.23
Determine trajectory steepness: r_L = 1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.02%
Height deviation: d_H = 0.31%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell14 cell12 cube1
#4: pickplace cell16 cell14 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell13 cell8 cube2
#10: pickplace cell9 cell13 cube3
Pick
Place
Task step #4 was successful, starting next action...
Current Configuration

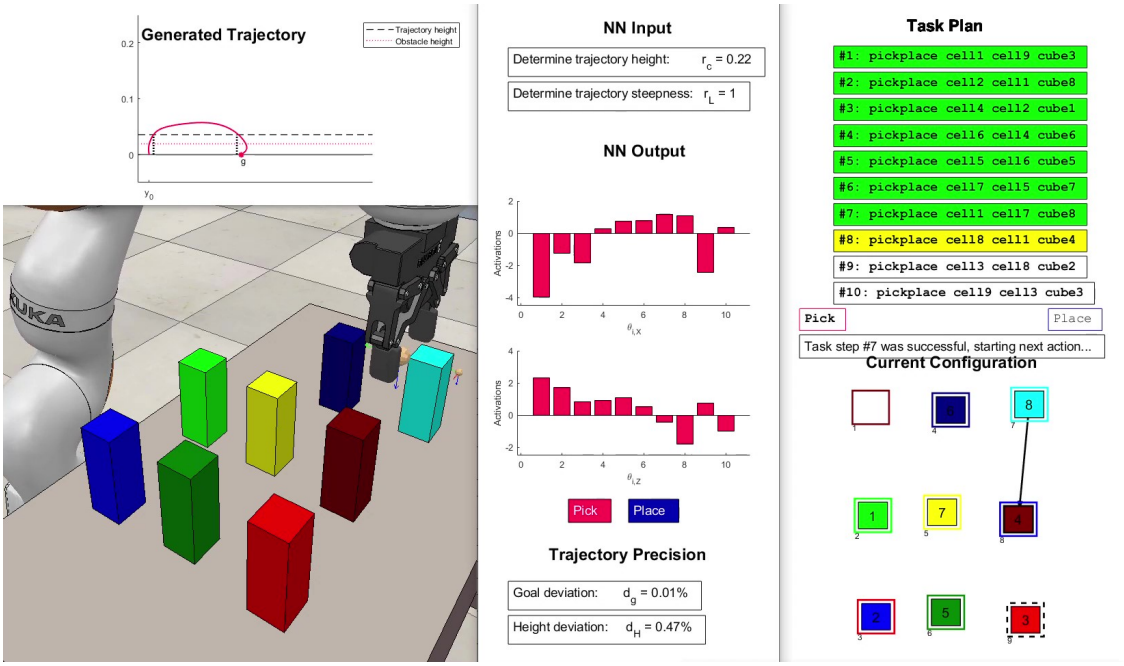

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.1
Determine trajectory steepness: r_L = 0.1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.02%
Height deviation: d_H = 0.43%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell14 cell12 cube1
#4: pickplace cell16 cell14 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell13 cell8 cube2
#10: pickplace cell9 cell13 cube3
Pick
Place
Task step #4 was successful, starting next action...
Current Configuration

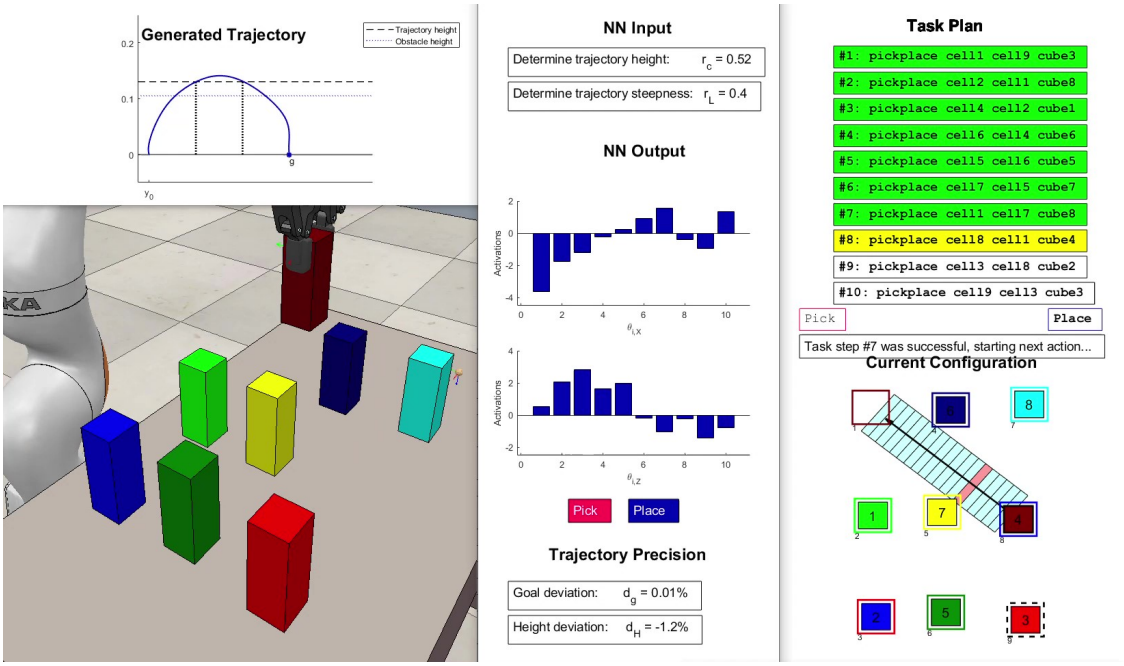

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.16
Determine trajectory steepness: r_L = 1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.01%
Height deviation: d_H = 0.04%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell4 cell12 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #5 was successful, starting next action...
Current Configuration

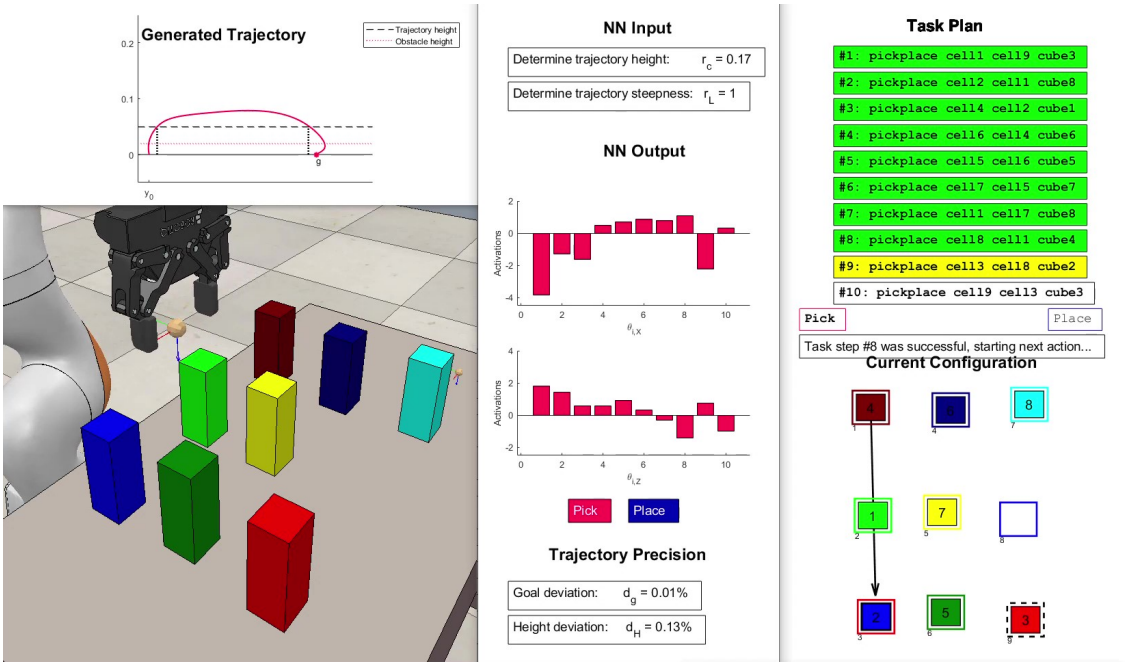

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.1
Determine trajectory steepness: r_L = 0.5
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.04%
Height deviation: d_H = -0.24%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell4 cell12 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #5 was successful, starting next action...
Current Configuration

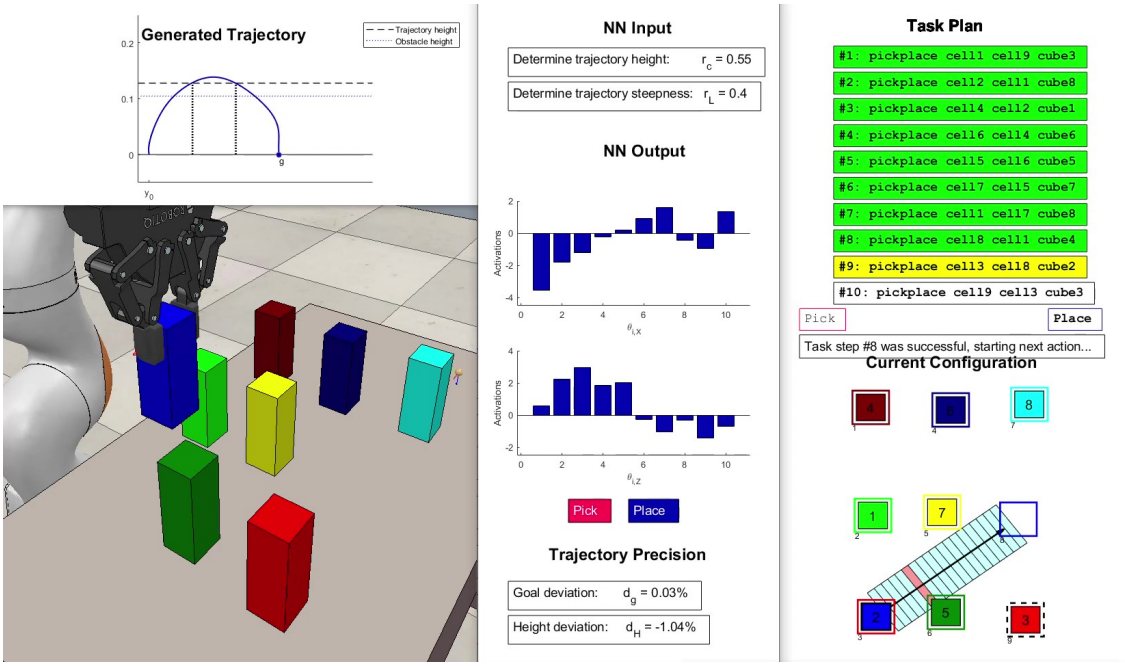

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.21
Determine trajectory steepness: r_L = 1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.01%
Height deviation: d_H = 0.32%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell4 cell12 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #6 was successful, starting next action...
Current Configuration

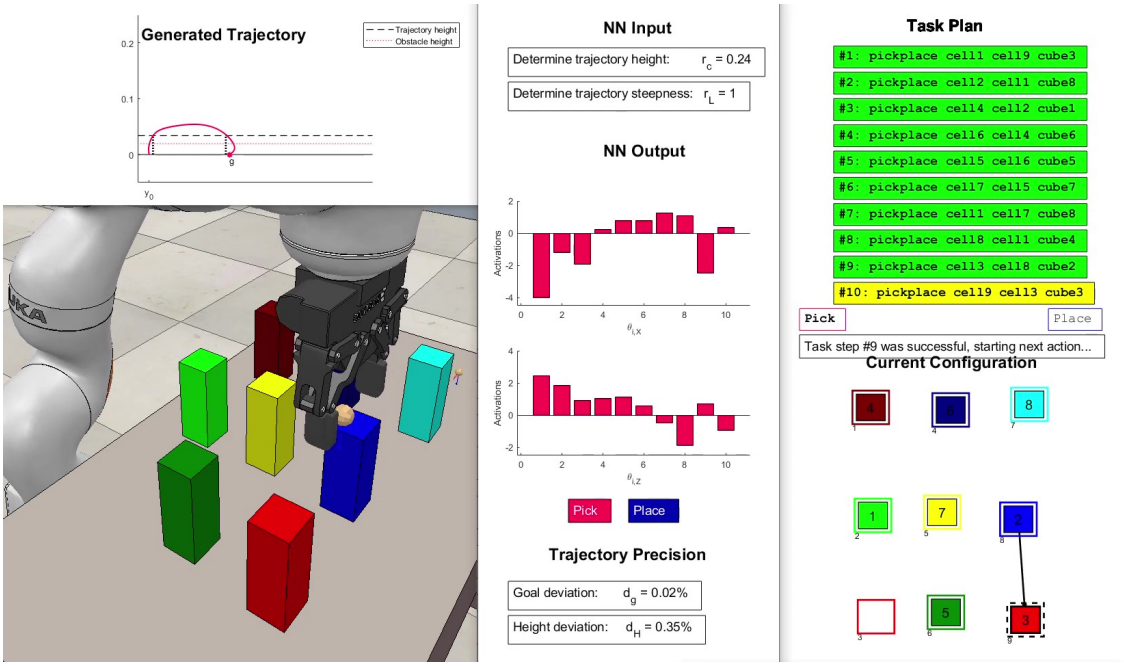

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.59
Determine trajectory steepness: r_L = 0.3
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0%
Height deviation: d_H = -0.31%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell4 cell12 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #6 was successful, starting next action...
Current Configuration

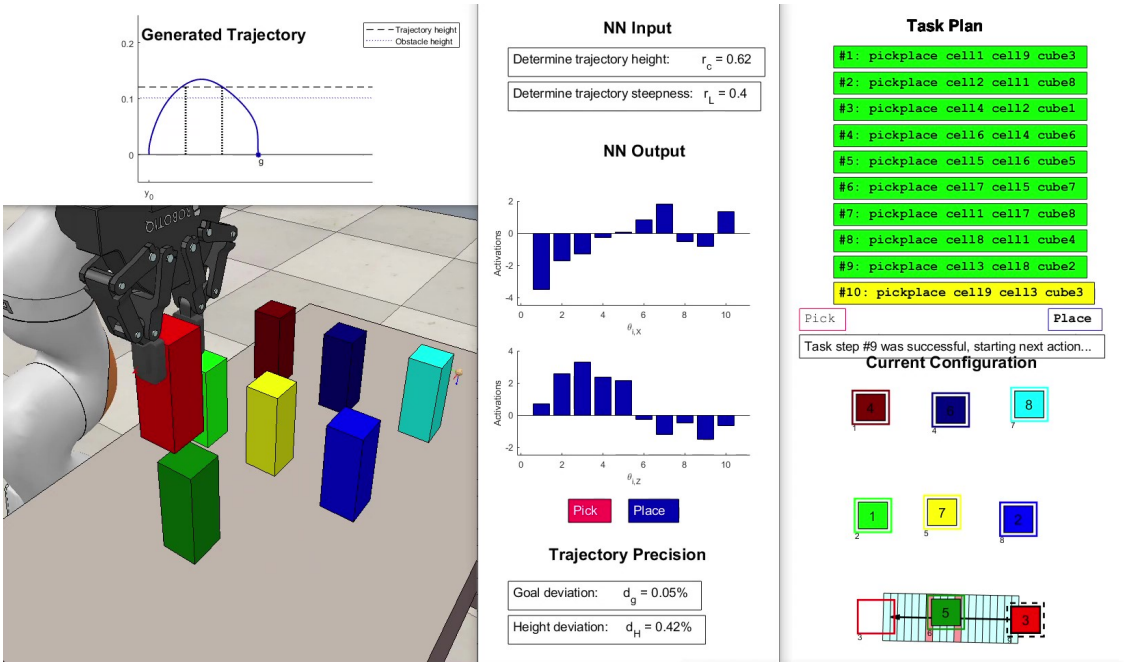

Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.22
Determine trajectory steepness: r_L = 1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.01%
Height deviation: d_H = 0.47%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell14 cell12 cube1
#4: pickplace cell16 cell14 cube6
#5: pickplace cell15 cell16 cube5
#6: pickplace cell17 cell15 cube7
#7: pickplace cell1 cell17 cube8
#8: pickplace cell18 cell1 cube4
#9: pickplace cell13 cell18 cube2
#10: pickplace cell9 cell13 cube3
Pick
Place
Task step #7 was successful, starting next action...
Current Configuration


Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.52
Determine trajectory steepness: r_L = 0.4
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.01%
Height deviation: d_H = -1.2%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell14 cell12 cube1
#4: pickplace cell16 cell14 cube6
#5: pickplace cell15 cell16 cube5
#6: pickplace cell17 cell15 cube7
#7: pickplace cell1 cell17 cube8
#8: pickplace cell18 cell1 cube4
#9: pickplace cell13 cell18 cube2
#10: pickplace cell9 cell13 cube3
Pick
Place
Task step #7 was successful, starting next action...
Current Configuration


Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.17
Determine trajectory steepness: r_L = 1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.01%
Height deviation: d_H = 0.13%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell4 cell12 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #8 was successful, starting next action...
Current Configuration


Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.55
Determine trajectory steepness: r_L = 0.4
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.03%
Height deviation: d_H = -1.04%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell4 cell12 cube1
#4: pickplace cell6 cell4 cube6
#5: pickplace cell5 cell6 cube5
#6: pickplace cell7 cell5 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #8 was successful, starting next action...
Current Configuration


Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.24
Determine trajectory steepness: r_L = 1
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.02%
Height deviation: d_H = 0.35%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell14 cell12 cube1
#4: pickplace cell16 cell14 cube6
#5: pickplace cell15 cell16 cube5
#6: pickplace cell7 cell15 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #9 was successful, starting next action...
Current Configuration


Generated Trajectory
Trajectory height
Obstacle height
NN Input
Determine trajectory height: r_c = 0.62
Determine trajectory steepness: r_L = 0.4
NN Output
Activations
Pick
Place
Trajectory Precision
Goal deviation: d_g = 0.05%
Height deviation: d_H = 0.42%
Task Plan
#1: pickplace cell1 cell9 cube3
#2: pickplace cell12 cell1 cube8
#3: pickplace cell14 cell12 cube1
#4: pickplace cell16 cell14 cube6
#5: pickplace cell15 cell16 cube5
#6: pickplace cell7 cell15 cube7
#7: pickplace cell1 cell7 cube8
#8: pickplace cell8 cell1 cube4
#9: pickplace cell3 cell8 cube2
#10: pickplace cell9 cell3 cube3
Pick
Place
Task step #9 was successful, starting next action...
Current Configuration

List of Figures

Symbols

s Symbolic state

a Symbolic action

$\hat{\tau}$ Time constant encoding the duration of the demonstrated trajectory

$\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}$ Positions, velocities and accelerations representing the trajectory

α, β Constants of a spring-damper system

y_0 Initial position of the DMPs

g Goal position of the DMPs

$i = 1, ..., N$ Number of basis functions for the DMPs

Ψ Basis function

f Forcing term of a DMP

θ Forcing term parameter

D Demonstration

l Length of the trajectory (distance between y_0 and g)

σ^2 Exploration variance of PI^2

ϵ Exploration noise of PI^2

$k = 1, ..., K$ Number of explored samples within one PI^2 iteration

$j = 1, ..., J$ Number of iterations within one PI^2 optimization

S Cost term of PI^2

ϕ Terminal cost of PI^2

q Immediate cost of PI^2

$t = 1, ..., T$ Time steps of a trajectory

m Margin of S_{scope}

W_θ Weights of the K samples depending on S

p Observation point along the linear demonstration (X coordinate)

h_p Trajectory coordinate in Y or Z direction at p

W_p Weight that regulates h_{p1} relative to another h_{p2}

z_p Normalized h_p by W_p

H Height of the trajectory

$\hat{H}$ Height of the obstacle

v_X, v_Y Distance of the grid cells in X and Y direction

$b = 1, ..., B$ Number of borders that define the obstacle location

L Length of the trajectory at height H indicating the trajectory steepness

$\hat{L}$ Length of the obstacle

$\tilde{L}$ Dilatation invariant L represented as a tuple of observation points $(p1, p2)$

A Area that is observed for obstacles

w_A Width of the observable area A

r_c Degree of curvature (dilatation invariant description of H)

r_L Trajectory steepness (dilatation invariant description of L as scalar)

$n_{\tilde{L}}$ Number of distinct r_L

n_P Number of distinct PI^2 optimizations

d_g Goal deviation relative to l_D

d_H Height deviation relative to l_D

e_v Maximum noise applied to the cell positions

Acronyms

TAMP Task and Motion Planning

LfD Learning from Demonstration

RL Reinforcement Learning

DMP Dynamic Movement Primitive

PI^2 Policy Improvement with Path Integrals

PDDL Planning Domain Definition Language

Bibliography

[ABC+20] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[ASLP20] Alejandro Agostini, Matteo Saveriano, Dongheui Lee, and Justus Piater. Manipulation planning using object-centered predicates and hierarchical decomposition of contextual actions. *IEEE Robotics and Automation Letters*, 5(4):5629–5636, 2020.

[BKLS17] Julien Bidot, Lars Karlsson, Fabien Lagriffoul, and Alessandro Saffiotti. Geometric backtracking for combined task and motion planning in robotic systems. *Artificial Intelligence*, 247:229–265, 2017.

[DHT20] Danny Driess, Jung-Su Ha, and Marc 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.

[DKCK18] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research*, 37(10):1134–1151, 2018.

[GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[GSF19] Michele Ginesi, Nicola Sansonetto, and Paolo Fiorini. Dmp++: Overcoming some drawbacks of dynamic movement primitives. *arXiv preprint arXiv:1908.10608*, 2019.

[Hel06] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[INH+13] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[KLP11] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477. IEEE, 2011.

[MGH+98] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.

[Mor78] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[QWA15] Benjamin Quack, Florentin Wörgötter, and Alejandro Agostini. Simultaneously learning at different levels of abstraction. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4600–4607. IEEE, 2015.

[RSF13] Eric Rohmer, Surya PN Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE, 2013.

[SBB+16] Peter Stone, Rodney Brooks, Erik Brynjolfsson, Ryan Calo, Oren Etzioni, Greg Hager, Julia Hirschberg, Shivaram Kalyanakrishnan, Ece Kamar, Sarit Kraus, Kevin Leyton-Brown, David Parkes, William Press, AnnaLee Saxenian, Julie Shah, Milind Tambe, and Astro Teller. Artificial intelligence and life in 2030, 2016.

[SS12] Freek Stulp and Olivier Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. 2012.

[STS12] Freek Stulp, Evangelos A Theodorou, and Stefan Schaal. Reinforcement learning with sequences of motion primitives for robust manipulation. *IEEE Transactions on robotics*, 28(6):1360–1370, 2012.

[TBS10] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Learning policy improvements with path integrals. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 828–835, 2010.

[Tou15] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence*, pages 1930–1936, 2015.

[VSB+19] Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothörl, Todd Hester, and Jon Scholz. A practical approach to insertion with variable socket position using deep reinforcement learning. In *2019 International*

Conference on Robotics and Automation (ICRA), pages 754–760. IEEE, 2019.

[WDSK19] Andrew M Wells, Neil T Dantam, Anshumali Shrivastava, and Lydia E Kavraki. Learning feasibility for task and motion planning in tabletop environments. *IEEE robotics and automation letters*, 4(2):1255–1262, 2019.

License