Article

The International Journal of Robotics Research
2021, Vol. 40(12-14) 1435–1466

DOI: 10.1177/02783649211056967
journals.sagepub.com/home/ijr

Learning to solve sequential physical reasoning problems from a scene image

Danny Driess[1,2], Jung-Su Ha[1,2] and Marc Toussaint[1,2]

Abstract

In this article, we propose deep visual reasoning, which is a convolutional recurrent neural network that predicts discrete action sequences from an initial scene image for sequential manipulation problems that arise, for example, in task and motion planning (TAMP). Typical TAMP problems are formalized by combining reasoning on a symbolic, discrete level (e.g., first-order logic) with continuous motion planning such as nonlinear trajectory optimization. The action sequences represent the discrete decisions on a symbolic level, which, in turn, parameterize a nonlinear trajectory optimization problem. Owing to the great combinatorial complexity of possible discrete action sequences, a large number of optimization/motion planning problems have to be solved to find a solution, which limits the scalability of these approaches. To circumvent this combinatorial complexity, we introduce deep visual reasoning: based on a segmented initial image of the scene, a neural network directly predicts promising discrete action sequences such that ideally only one motion planning problem has to be solved to find a solution to the overall TAMP problem. Our method generalizes to scenes with many and varying numbers of objects, although being trained on only two objects at a time. This is possible by encoding the objects of the scene and the goal in (segmented) images as input to the neural network, instead of a fixed feature vector. We show that the framework can not only handle kinematic problems such as pick-and-place (as typical in TAMP), but also tool-use scenarios for planar pushing under quasi-static dynamic models. Here, the image-based representation enables generalization to other shapes than during training. Results show runtime improvements of several orders of magnitudes by, in many cases, removing the need to search over the discrete action sequences.

Keywords

Task and motion planning, deep learning, sequential manipulation, physical reasoning, planar pushing, logic geometric programming, offline reinforcement learning, deep Q-learning

1. Introduction

A major challenge in sequential manipulation problems is that they inherently involve discrete and continuous aspects. To account for this hybrid nature of manipulation, task and motion planning (TAMP) problems are usually formalized by combining reasoning on a symbolic, discrete level with continuous motion planning. The symbolic level, e.g., defined in terms of first-order logic, proposes high-level discrete action sequences for which the motion planner, e.g., nonlinear trajectory optimization or a sampling-based method, tries to find configurations and motions that fulfill the requirements induced by the high-level action sequence or return that the action sequence is infeasible. Although most TAMP approaches focus on kinematic tasks such as pick and place, the hybrid nature of manipulation more broadly appears in manipulation planning through contacts, where it is typically addressed in terms of mixed integer optimization formulations.

Owing to the high combinatorial complexity of possible discrete action sequences or integer assignments, a large number of (potentially hard) motion planning problems have to be solved to find a solution to the overall TAMP/manipulation problem. This is mainly caused by the fact that many TAMP problems are difficult, because, mostly due to kinematic limits and geometric constraints, the majority of action sequences are actually infeasible. Moreover, it typically takes more computation time for a motion planner to reliably detect infeasibility of a high-level action sequence than to find a feasible motion when

[1]Learning and Intelligent Systems, TU Berlin, Germany
[2]Max-Planck Institute for Intelligent Systems, Stuttgart, Germany

Corresponding author:
Danny Driess, Learning and Intelligent Systems Lab, TU Berlin, Marchstraße 23, 10587 Berlin, Germany.
Email: danny.driess@gmail.com

it exists. Proving infeasibility is, in many cases, not even possible. In addition, for a feasible action sequence, the resulting motion planning problem in itself is often non-trivial and hard to solve. Consequently, sequential manipulation problems, which intuitively seem simple, can take a very long time to solve.

To overcome this combinatorial complexity, we aim to learn to predict promising action sequences from the scene as input. Using this prediction as a search heuristic on the symbolic level, we can drastically reduce the number of motion planning problems that need to be evaluated. Ideally, we seek to directly predict a *feasible* and cost-efficient action sequence, requiring only a *single* motion planning problem to be solved.

However, learning to predict such action sequences imposes multiple challenges. First, the objects in the scene and the goal have to be encoded as input to the predictor in a way that enables similar generalization capabilities to classical TAMP approaches with respect to scenes with many and changing numbers of objects and goals. Second, the large variety of such scenes and goals, especially if multiple objects are involved, makes it difficult to generate a sufficient dataset.

Recently, Wells et al. (2019) and Driess et al. (2020b) proposed a classifier that predicts the feasibility of a motion planning problem resulting from a discrete decision in the task domain. However, a major limitation of their approaches is that the feasibility for only a *single* action is predicted, whereas the combinatorial complexity of TAMP especially arises from action *sequences* and it is not straightforward to utilize such a classifier for action sequence prediction within TAMP.

To address these issues, we train a neural network to predict action *sequences* from the initial scene *and* the goal as input. An important question is how the objects in the scene and the goal can be encoded as input to the predictor to ensure strong generalization. By encoding the objects (and the goal) in the image space, we show that the network is able to generalize to scenes with many and changing numbers of objects with only little runtime increase, although it has been trained on only a fixed number of objects.

Compared with a purely discriminative model, because the predictions of our network are goal-conditioned, we do not rely on the network to search over many sequences, but can directly generate promising sequences with the network efficiently.

The predicted action sequences parameterize a nonlinear trajectory optimization problem that optimizes a globally consistent path fulfilling the requirements induced by the actions. This not only allows us to solve typical kinematic problems such as pick and place, but also problems that involve dynamic models. In our case, we build upon the contact formulation from Toussaint et al. (2020) to solve manipulation problems where the robot has to utilize a hook-shaped tool to push and/or pull an object to a desired target location. In order to make this possible, we extend the formulation from Toussaint et al. (2020) by introducing additional discrete decisions that model which part of the tool should establish contact with which side of the object. This robustifies the convergence of the optimizer, but also introduces additional combinatorics, which we can tackle with our proposed network. For this scenario, we demonstrate another advantage of the image-based representation, namely that the network can also generalize to other shapes than during training, because the side to push on the object is also encoded in the image space.

To summarize, our main contributions are as follows.

- A convolutional, recurrent neural network (RNN) that predicts from an initial segmented scene image and a task goal promising action sequences, which parameterize a nonlinear trajectory optimization problem, to solve the TAMP problem.
- A way to integrate this network into the tree search algorithm of the underlying TAMP framework.
- We demonstrate that the network generalizes to situations with many and varying numbers of objects in the scene, although it has been trained on only two objects at a time.
- We demonstrate the method not only on typical kinematic pick-and-place scenarios, but also in a scenario where a hook-shaped tool has to be used to push/pull an object to a desired target location.
- The image representation enables the network to generalize to other shapes than during training for the pushing scenario.

From a methodological point of view, this work combines nonlinear trajectory optimization, first-order logic reasoning and deep convolutional RNNs.

The present work is an extended version of Driess et al. (2020a). Apart from more in-depth explanations, we include a new set of experiments on a tool-use pushing scenario (Section 5.3), where we also show generalization capabilities to other shapes than during training. Moreover, we provide more evaluations of the existing experiments, e.g., with an investigation of the data efficiency of the approach. Furthermore, we extend the framework to not only being able to find feasible solutions as in the original work, but also taking the trajectory costs into account.

The rest of the article is organized as follows. We describe logic geometric programming (LGP) as the TAMP framework of this work in Section 3. Then, in Section 4, the deep visual reasoning neural network methodology, its architecture, and how it can be integrated into the LGP tree search algorithm is proposed. In Section 4.5, we discuss the relation of our proposed network to offline reinforcement learning (RL). Sections 5.2 and 5.3 present the results of the pick and place and the pushing experiments, respectively. A discussion about the strengths and limitations of the framework can be found in Section 6.

2. Related work

2.1. Learning to plan

There is great deal of interest in learning to mimic planning itself. The architectures in Tamar et al. (2016), Okada et al. (2017), Srinivas et al. (2018), and Amos et al. (2018) resemble value iteration, path integral control, gradient-based trajectory optimization, and iterative linear quadratic regulator (LQR) methods, respectively. For sampling-based motion planning, Ichter et al. (2018) learned an optimal sampling distribution conditioned on the scene and the goal to speed up planning. To enable planning with raw sensory input, there are several works that learn a compact representation and its dynamics in sensor space to then apply planning or RL in the learned latent space (Boots et al., 2011; Finn et al., 2016; Ha et al., 2018; Ichter and Pavone, 2019; Lange et al., 2012; Silver et al., 2017; Watter et al., 2015; Xie et al., 2019). Another line of research is to learn an action-conditioned predictive model (Dosovitskiy and Koltun, 2017; Ebert et al., 2017; Finn and Levine, 2017; Pascanu et al., 2017; Paxton et al., 2019; Racanière et al., 2017; Xie et al., 2019). With this model, the future state of the environment, e.g., in image space conditioned on the action, is predicted, which can then be utilized within model predictive control (Finn and Levine, 2017; Xie et al., 2019) or to guide tree search (Paxton et al., 2019). The underlying idea is that learning the latent representation and dynamics enables reasoning with high-dimensional sensory data. However, a disadvantage of such predictive models is that a search over actions is still necessary, which grows exponentially with sequence length. For our problem which contains handovers or other complex behaviors that are induced by an action, learning a predictive model in the image space seems difficult. Most of these approaches focus on low-level actions. Furthermore, the behavior of our trajectory optimizer is only defined for a complete action sequence, because future actions have an influence on the trajectory of the past. Therefore, state predictive models cannot be applied directly to our problem.

The proposed method in the present work learns a relevant representation of the scene from an initial scene image such that a recurrent module can reason about long-term action effects without a direct state prediction.

2.2. Learning heuristics for TAMP and MIP in robotics

A general approach to TAMP (Garrett et al., 2020) is to combine discrete logic search with a sampling-based motion planning algorithm (Dantam et al., 2018; de Silva et al., 2013; Kaelbling and Lozano-Pérez, 2011; Srivastava et al., 2014) or constraint satisfaction methods (Lagriffoul et al., 2014; Lagriffoul et al., 2012; Lozano-Pérez and Kaelbling, 2014). A major difficulty arises from the fact that the number of feasible symbolic sequences increases exponentially with the number of objects and sequence length. To reduce the large number of geometric problems that need to be solved, many heuristics have been developed, e.g., Kaelbling and Lozano-Pérez (2011), Rodriguez et al. (2019), and Driess et al. (2019a), to efficiently prune the search tree. Another approach to TAMP is LGP (Ha et al., 2020; Toussaint, 2015; Toussaint et al., 2018, 2020; Toussaint and Lopes, 2017), which combines logic search with trajectory optimization. The advantage of an optimization-based approach to TAMP is that the trajectories can be optimized with global consistency, which, e.g., allows handover motions to be generated efficiently. LGP will be the underlying framework of the present work. For large-scale problems, however, LGP also suffers from the exponentially increasing number of possible symbolic action sequences (Hartmann et al., 2020). Solving this issue is one of the main motivations for our work.

Instead of handcrafted heuristics, there are several approaches to integrate learning into TAMP to guide the discrete search in order to speed up finding a solution (Chitnis et al., 2016, 2020; Garrett et al., 2016; Kim et al., 2018, 2019; Wang et al., 2018). However, these mainly act as heuristics, meaning that one still has to search over the discrete variables and probably solve many motion planning problems. In contrast, the network in our approach generates goal-conditioned action sequences, such that in most cases there is no search necessary at all. Similarly, in optimal control for hybrid domains mixed-integer programs suffer from the same combinatorial complexity (Doshi et al., 2020; Hogan et al., 2018; Hogan and Rodriguez, 2016). LGP also can be viewed as a generalization of mixed-integer programs. In Carpentier et al. (2017) (footstep planning) and Hogan et al. (2018) (planar pushing), learning was used to predict the integer assignments, however, this was for a single task only with no generalization to different scenarios.

A crucial question in integrating learning into TAMP is how the scene and goals can be encoded as input to the learning algorithm in a way that enables similar generalization capabilities to classical TAMP. For example, in Paxton et al. (2019) the considered scene always contains the same four objects with the same colors, which allows them to have a fixed input vector of separate actions for all objects. In Wilson and Hermans (2019), convolutional neural networks (CNNs) and graph neural networks are utilized to learn a state representation for RL, similarly in Li et al. (2020). In Bejjani et al. (2019), rendered images from a simulator were used as state representation to exploit the generalization ability of CNNs. The work of Kloss et al. (2020) exploits image-based representations for pushing scenarios, similar to our encoding. In our work, the network learns a representation in image space that is able to reason over complex action sequences from an initial observation only and is able to generalize over changing numbers of objects.

The work of Wells et al. (2019) and Driess et al. (2020b) is most related to our approach. They both proposed to learn a classifier which predicts the feasibility of a motion planning problem resulting from a *single* action. The input

is a feature representation of the scene (Wells et al., 2019) or a scene image (Driess et al., 2020b). Although both show generalization capabilities to multiple objects, one major challenge of TAMP comes from action sequences and it is, however, unclear how a single step classifier as in Wells et al. (2019) and Driess et al. (2020b) could be utilized for sequence prediction.

On challenge in TAMP scenarios that contain many objects is to identify which objects are relevant for solving the task, as has been considered in Lang and Toussaint (2009) and Silver et al. (2020). Our proposed approach is also able to reason over object importance, which makes the generalization to multiple objects possible.

To the best of the authors' knowledge, our work is the first that learns to generate action sequences for an optimization-based TAMP approach from an initial scene image and the goal as input, while showing generalization capabilities to multiple objects.

3. Logic Geometric Programming for Task and Motion Planning

This work relies on LGP (Toussaint, 2015; Toussaint and Lopes, 2017) as the underlying TAMP framework. The high-level main idea behind LGP is a nonlinear trajectory optimization problem over the continuous variable x, which is the path of all robot joints and objects in the scene. The constraints and costs of this trajectory optimization problem are parameterized by a discrete variable s that represents the state of a symbolic domain. The transitions of this symbolic variable are subject to a first-order logic language that induces a decision tree. Solving an LGP involves a tree search over sequences of this discrete variable, where each leaf node represents a nonlinear trajectory optimization program (NLP). If a symbolic leaf node, i.e., a node where the state s is in a symbolic goal state, is found and its corresponding NLP is feasible, a solution to the TAMP problem has been obtained. At all nodes, simpler NLPs define lower bounds to the full trajectory optimization problem, which can guide the search over the discrete actions. The LGP formulation we present in this section is based on a recent variant from Toussaint et al. (2020) that additionally allows optimizing for physical interactions as compared with the original formulation from Toussaint (2015). However, in addition to Toussaint et al. (2020), we also consider the search over actions that define a sequence of symbolic states as part of the problem for the contact model formulations introduced by Toussaint et al. (2020), where the sequence of the symbolic state was specified manually.

3.1. LGP

Let $\mathcal{X} = \mathcal{X}(s, S) \subset \mathbb{R}^{n(S)} \times SE(3)^{m(s,S)} \times \mathbb{R}^{6 \cdot n_{\text{cp}}(s,S)}$ be the configuration space as a function of the scene S and the symbolic state variable $s \in \mathcal{S}(S)$. This configuration space contains the $n(S)$-dimensional generalized coordinate of robot joints, the poses of $m(s, S)$ multiple rigid objects and six-dimensional wrench contact interactions for each of the $n_{\text{cp}}(s, S)$ contact pairs.

The idea is to find a globally consistent path $x : [0, KT] \to \mathcal{X}(s_{k(t)}, S)$ in this configuration space which minimizes the LGP

$$P(g,S) = \min_{\substack{K \in \mathbb{N} \\ x:[0,KT] \to \mathcal{X}(s_{k(t)},S) \\ a_{1:K}, s_{1:K}}} \int_0^{KT} c(x(t), \dot{x}(t), \ddot{x}(t), s_{k(t)}, S)\, \mathrm{d}$$

$$\text{s.t} \tag{1a}$$

$$\forall_{t \in [0,KT]}: \quad h_{\text{eq}}(x(t), \dot{x}(t), s_{k(t)}, S) = 0 \tag{1b}$$

$$\forall_{t \in [0,KT]}: \quad h_{\text{ineq}}(x(t), \dot{x}(t), s_{k(t)}, S) \leqslant 0 \tag{1c}$$

$$\forall_{k=1,\ldots,K}: \quad h_{\text{sw}}(x(kT), \dot{x}(kT), a_k, S) = 0 \tag{1d}$$

$$\forall_{k=1,\ldots,K}: \quad a_k \in \mathbb{A}(s_{k-1}, S) \tag{1e}$$

$$\forall_{k=1,\ldots,K}: \quad s_k = \text{succ}(s_{k-1}, a_k) \tag{1f}$$

$$x(0) = \tilde{x}_0(S) \tag{1g}$$

$$s_0 = \tilde{s}_0(S) \tag{1h}$$

$$s_K \in \mathcal{S}_{\text{goal}}(g). \tag{1i}$$

The path x is assumed to be globally continuous ($x \in C([0, TK])$) and consists of $K \in \mathbb{N}$ phases (the number is part of the decision problem itself), each of fixed duration $T > 0$, in which we require smoothness $x \in C^2([(k-1)T, kT])$. These phases are also referred to as kinematic modes (Mason, 1985; Toussaint et al., 2018). Note that the number of degrees of freedom of the objects as well as the number of contact interactions depends on the symbolic state $s_{k(t)}$ with $k(t) = \lfloor t/T \rfloor$. Therefore, the dimension of the path may vary between the phases. For example, the symbolic state can express that a contact interaction should take place at a certain phase, which introduces a wrench interaction variable to the configuration space in that phase.

The functions $c, h_{\text{eq}}, h_{\text{ineq}}$ and, hence, the objectives in phase k of the motion are parameterized by the discrete variable (or integers in mixed-integer programming) $s_k \in \mathcal{S}(S)$, representing the state of the symbolic domain. The possible discrete transitions between s_{k-1} and s_k are determined by the successor function succ($\cdot$, $\cdot$), which is a function of the previous state s_{k-1} and the discrete action a_k at phase k. The successor function is defined through the first-order logic language.

The discrete actions a_k are of great importance for the rest of the article. In general, those actions are grounded action operators. Which actions are possible at which symbolic state is determined by the logic and expressed in the set $\mathbb{A}(s_{k-1}, S)$.

The task or goal of the TAMP problem is defined symbolically through the set $\mathcal{S}_{\text{goal}}(g)$ for the symbolic goal (a set of grounded literals) $g \in \mathbb{G}(S)$, e.g., placing an object on a table.

To complete the description, h_{sw} is a function that imposes transition constraints on the path between the kinematic modes. The quantity $\tilde{x}_0(S)$ is the scene-dependent initial continuous state. For fixed s it is assumed that c, h_{eq} and h_{ineq} are differentiable.

A sequence of actions $a_{1:K}$ uniquely determines the sequence of symbolic states $s_{0:K}$ for a given initial symbolic state $s_0 = \tilde{s}_0(S)$. Therefore, we equivalently express that obtaining a solution to the LGP means finding an action sequence $a_{1:K}$ whose corresponding symbolic state sequence reaches the symbolic goal state *and* the corresponding nonlinear trajectory optimization problem is feasible. We define the feasibility of an action sequence $a_{1:K} = (a_1, \ldots, a_K)$ via the existence of a respective path, namely,

$$F_S(a_{1:K}) = \begin{cases} 1 & \exists x : [0, KT] \to \mathcal{X} : (1\text{b}) - (1\text{h}) \\ 0 & \text{otherwise} \end{cases}. \quad (2)$$

The complete LGP formulation (1), however, not only seeks to find a feasible solution, but also an action sequence that leads to the minimum trajectory costs (1a) compared with all other goal reaching sequences. We call a feasible solution a *solution to the TAMP/LGP problem*, whereas we call a solution that minimizes the cost an *optimal solution of the LGP*. Owing to the vast number of possible discrete action sequences, obtaining optimal solutions is often intractable. Therefore, we mostly focus on obtaining a feasible solution, with the exception of Sections 4.8 and 5.3.7.

The feasibility as defined in (2) is a theoretical property of the resulting nonlinear program. In practice, we solve (1) numerically by discretizing x with a finite number of collocation points in time, which leads to a finite-dimensional optimization problem that we solve with an augmented Lagrangian method with the Gauss–Newton method in its inner loop. Therefore, $F_S(a_{1:K})$ is also determined numerically in the following way. If the accumulated constraint violations ($h_{\text{eq}} \neq 0$, $h_{\text{ineq}} > 0$) along the discretized trajectory are below a certain threshold, then the solution found by the optimizer is considered feasible, otherwise it is considered infeasible.

3.2. Multi-bound LGP tree search and lower bounds

The logic induces a decision tree (called LGP tree) through the set of possible actions $\mathbb{A}(s_{k-1}, S)$ (1e) and the successor function (1f). Solving a full path problem as a search heuristic to guide the tree search is too expensive. A key contribution of Toussaint and Lopes (2017) is therefore to introduce relaxations or lower bounds on (1) in the sense that the feasibility of a lower bound is a necessary condition on the feasibility of the complete problem (1), whereas these lower bounds should be computationally faster to compute. Each node in the LGP tree defines several lower bounds of (1).

More specifically, multi-bound LGP tree search uses two lower bounds P_1, P_2, where P_1 evaluates kinematic feasibility of transition constraints h_{sw} for a single configuration only (akin to inverse kinematics), and P_2 evaluates feasibility of a sequence of K transition configurations. Only if both are feasible along a tree path it evaluates the full trajectory LGP for a leaf node.

As we show in the experiments, even with those bounds, a large number of NLPs have to be solved to find a feasible solution for problems with a high combinatorial complexity. Therefore, using these bounds is not sufficient to achieve desirable solution times for the problems we consider in the experiments. This is especially true if many decisions are feasible in early phases of the sequence, but then later become infeasible, because then the lower bounds do not help much in pruning the search tree. These lower bounds are, however, highly important to make the data generation process tractable.

4. Deep visual reasoning

The central idea of this work is, given the scene and the task goal as input, to predict a promising discrete action sequence $a_{1:K} = (a_1, \ldots, a_K)$ which reaches a symbolic goal state and its corresponding trajectory optimization problem is feasible. An ideal algorithm would directly predict an action sequence such that only a single NLP has to be solved to find an overall feasible solution, which consequently would lead to a significant speedup in solving the LGP (1).

We first describe more precisely what should be predicted, then how the scene, i.e., the objects and actions that operate on them, and the goal can be encoded as input to a neural network that should perform the prediction. Finally, we discuss how the network is integrated into the tree search algorithm in a way that either directly predicts a feasible sequence or, in case the network is mistaken, acts as a search heuristic to further guide the search without losing completeness, i.e., the ability to find a solution if one exists. To allow for a more thorough comparison, we additionally propose an alternative way to integrate learning into LGP based on a *goal-independent* recurrent feasibility classifier.

4.1. Predicting promising action sequences

First, we define for the goal g the set of all action sequences that lead to a symbolic goal state in the scene S as

$$\mathcal{T}(g, S) = \{a_{1:K} : \forall_{i=1}^{K} a_i \in \mathbb{A}(s_{i-1}, S), s_0 = \tilde{s}_0(S) \\ s_i = \text{succ}(s_{i-1}, a_i), s_K \in \mathcal{S}_{\text{goal}}(g)\}. \quad (3)$$

In relation to the LGP tree, this is the set of all leaf nodes and, hence, candidates for an overall feasible solution. One idea is to learn a discriminative model which predicts whether a complete sequence leads to a feasible NLP and, hence, to a solution. To predict an action sequence one

Table 1. Number of action sequences that lead to the symbolic goal state over the length of the sequence for different numbers of objects in the scene. These numbers correspond to the number of *candidate* trajectory optimization problems that in the worst case have to be solved. Only a very small subset of those candidate sequences actually leads to a feasible NLP.

Number of objects	Length of the action sequence				
in the scene	2	3	4	5	6
1	8	32	192	1,024	5,632
2	8	96	704	6,400	51,200
3	8	160	1,216	15,872	145,920
4	8	224	1,728	29,440	289,792
5	8	288	2,240	47,104	482,816

would then choose the sequence from $\mathcal{T}(g,S)$ where the discriminative model has the highest prediction. However, computing $\mathcal{T}(g,S)$ (up to a maximum K) and then checking all sequences with the discriminative model is computationally inefficient or even intractable, since $|\mathcal{T}(g,S)|$ can be very large (Table 1).

Instead, we propose to learn a function π (a neural network) that, given a scene description S, the task goal g, and the past decisions $a_{1:k-1}$, predicts whether a next action a_k at the current time step k is promising in the sense of the probability that there exist future actions $a_{k+1:K}$ such that the complete sequence $a_{1:K}$ leads to a feasible NLP that solves the original TAMP problem. Formally,

$$\pi(a_k, g, a_1, \ldots, a_{k-1}, S) = p\Big(\exists_{K \geqslant k} \exists_{a_{k+1},\ldots,a_K} : a_{1:K} \in \mathcal{T}(g,S), F_S(a_{1:K}) = 1 \;\Big|\; a_k, g, a_1, \ldots, a_{k-1}, S\Big). \tag{4}$$

Note that this includes $K = k$, meaning the case where at step k there exists an action such that $s_k \in \mathcal{S}_{\text{goal}}$ and the NLP is feasible.

In this way, we can utilize π to generate an action sequence by choosing the action at each step where π has the highest prediction. The exact algorithm is presented in Section 4.6.

4.2. Training targets

The crucial question arises how π as defined in (4) can be trained. The semantics of π is related to a universal Q-function, but it evaluates actions a_k based on an implicit representation of state (see Section 4.5). Furthermore, it turns out that we can cast the problem into supervised learning by transforming the data into suitable training targets. Assume that one samples scenes S^i, goals g^i as well as goal-reaching action sequences $a^i_{1:K^i} \in \mathcal{T}(g^i, S^i)$, e.g., with multi-bound LGP tree search. For each of these sampled sequences, the feasibility of the resulting NLP is determined and saved in the set

$$\mathcal{D}_{\text{data}} = \left\{ \left(S^i, a^i_{1:K^i}, g^i, F_{S^i}\left(a^i_{1:K^i}\right) \right) \right\}_{i=1}^{n} \tag{5}$$

where the feasibility F_{S^i} in scene S^i is as defined in (2). Based on this dataset, we define the *training* dataset for π as

$$\mathcal{D}_{\text{train}} = \left\{ \left(S^i, a^i_{1:K^i}, g^i, f^i \right) \right\}_{i=1}^{n} \tag{6}$$

where $f^i \in \{0,1\}^{K^i}$ is a sequence of binary labels. Its jth component f^i_j indicates for every *sub*sequence $a^i_{1:j}$ whether it should be classified as promising as follows

$$f^i_j = \begin{cases} 1 & F_{S^i}\left(a^i_{1:K^i}\right) = 1 \\ 1 & \exists \left(S^l, a^l_{1:K^l}, g^l, F^l\right) \in \mathcal{D}_{\text{data}} : \\ & \quad F^l = F_{S^l}\left(a^l_{1:K^l}\right) = 1 \\ & \quad \wedge\, g^l = g^i \wedge a^l_{1:j} = a^i_{1:j} \\ 0 & \text{otherwise} \end{cases}. \tag{7}$$

Superscripts denote dataset indexing. If the action sequence is feasible and solves the problem specified by g^i, then $f^i_j = 1$ for all $j = 1, \ldots, K^i$ (first case). This is the case where π should predict a high probability at each step k to follow a feasible sequence. If the action sequence with index i is *not* feasible, but there exists a feasible one in $\mathcal{D}_{\text{data}}$ (index l) which has an overlap with the other sequence up to step j, i.e., $a^l_{1:j} = a^i_{1:j}$, then $f^i_j = 1$ as well (second case). In addition, in this case the network should suggest to follow this decision, because it predicts that there exist future decisions which lead to a feasible solution. Finally, in the last and third case where the sequence is infeasible and has no overlap with other feasible sequences, $f^i_j = 0$, meaning that the network should predict to not follow this decision. This data transformation is a simple pre-processing step that allows us to train π in a supervised sequence labeling setting, with the standard (weighted) binary cross-entropy loss. Another advantageous side-effect of this transformation is that it creates a more balanced dataset with respect to the training targets.

4.3. Input to the neural network: encoding a, g, and S

So far, we have formulated the predictor π in (4) in terms of the scene S, symbolic actions a, and the goal g of the LGP (1). In order to represent π as a neural network, we need to find suitable encodings of a, g, and S.

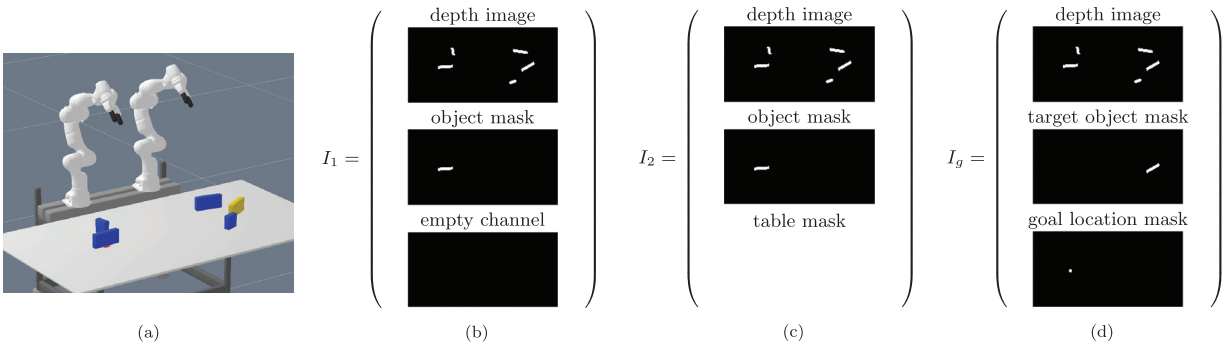


Figure 1. Visualization of the action–object image encodings $I(O,S)$ for an example test scene of the pick-and-place experiment from Section 5.2.9 (generalization to multiple objects). (a) Initial scene. (b) Action–object image for `grasp` action with the left robot arm and the blue object that occupies the red goal location. (c) Action–object image for `place` action with the left robot arm representing placing the blue object on the table. (d) Goal–object and action–object image (for `place` action) representing placing the yellow target object on the red goal location. In all images, the first channel is a depth channel of the complete scene. The images always refer to the initial scene configuration (a). The table mask in (c) covers the complete table, hence is completely white and indicates that the object should be placed somewhere on the table. Here I_1 and I_2 correspond to the first two actions chosen by the network. The network found a solution to solve the task with a sequence of six actions.

4.3.1. Splitting actions into action operator symbols and objects. An action a is a grounded action operator, i.e., it is a combination of an action operator symbol and objects in the scene it refers to. The same holds true for a goal g. Although the number of action operators is assumed to be constant, the number of objects can be completely different from scene to scene. Most neural networks, however, expect inputs of fixed dimension. In order to achieve the same generalization capabilities of TAMP approaches with respect to changing numbers of objects, we encode action and goal symbols very differently to the objects they operate on. In particular, object references are encoded in a way that includes geometric scene information.

Specifically, given an action a, we decompose it into $a=(\bar{a},O)\in\mathcal{AO}(s,S)\subset\mathcal{A}\times\mathcal{P}(\mathcal{O}(S))$, where $\bar{a}\in\mathcal{A}$ is its discrete action operator symbol and $O\in\mathcal{P}(\mathcal{O}(S))$ the *tuple* of objects the action operates on. Note that O is a tuple of objects, because an action a can operate on multiple objects. For example, placing an object on a table requires two elements from $\mathcal{O}$ to fulfill the predicate. The goal is similarly decomposed into $g=(\bar{g},O_g)$, $\bar{g}\in\mathcal{G}$, and $O_g\in\mathcal{P}(\mathcal{O}(S))$. This separation seems to be a minor technical detail, which is, however, of key importance for the generalizability of our approach to scenes with changing numbers of objects, following the generalizability of the underlying first-order logic.

The cardinality of $\mathcal{A}$ and $\mathcal{G}$ is constant and independent from the scene, no matter how many objects are in the scene. Therefore, through the separation of a into $\bar{a}$ and $\mathcal{O}$, we can input $\bar{a}$ and $\bar{g}$ directly as a one-hot encoding to the neural network.

4.3.2. Encoding the objects O and O_g in the image space. For our approach it is crucial to encode the information about the objects in the scene in a way that allows the neural network to generalize over objects, i.e., the number of objects in the scene and their properties. By using the separation of the last paragraph, we can introduce the mapping $I:(O,S)\mapsto\mathbb{R}^{(n_c+n_O)\times w\times h}$, which encodes any scene S and object tuple O to a so-called action–object image encoding, namely an n_c+n_O-channel image of width w and height h, where the first n_c channels represent an image of the *initial* scene and the last n_O channels are binary masks which indicate the subset of objects that are referenced by the action. These last mask channels not only encode object identity, but substantial geometric and relational information, which is a key for the predictor to predict feasible action sequences.

In Figure 1, we show these image encodings for a scene of the pick-and-place experiment. Figures 1(b) and (c) show action–object images, whereas Figure 1d shows the goal-image encoding.

It is important to understand that n_O is *not* the number of objects in the scene, but the maximal number of objects that *any* action refers to. In the experiments, the scene image is a depth image, i.e., $n_c=1$ and the maximum number of objects that a single action refers to is two, hence $n_O=2$. If an action takes fewer objects into account than n_O, this channel is zeroed. As the maximum number n_O depends on the set of actions operator symbols $\mathcal{A}$, which has a fixed cardinality independent from the scene, this is no limitation. The masks create an attention mechanism which is the key to generalize to multiple objects (Driess et al., 2020b). However, because each action object image $I(O,S)$ always contains a channel providing information of the complete scene, also the geometric relations to other objects can be taken into account. This is of crucial importance as illustrated with the following example. Assume that the goal location is occupied with an object as in

Figure 1. If the action–object image of a place action to place an other object on the goal location only consisted of the masks of the target object and the goal, then the network would not be able to reason that this action sequence is infeasible. Therefore, the purpose of the masks is to encode object identity of the directly involved objects in an action, whereas the channel of the whole scene is of great importance to enable the network to potentially take other objects into account.

Please note that these action–object images always correspond to the *initial* scene.

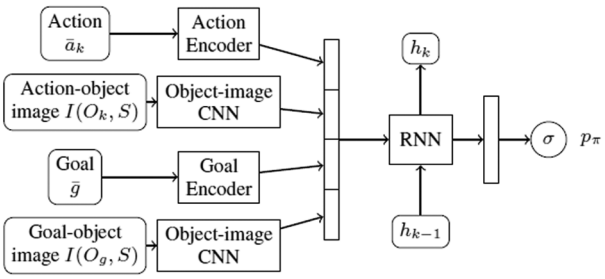


Figure 2. Proposed neural network architecture.

4.4. Network architecture

Figure 2 shows the network architecture that represents π as a convolutional RNN. Assume that in step k the probability should be predicted whether an action $a_k = (\bar{a}_k, O_k)$ for the goal $g = (\bar{g}, O_g)$ in the scene S is promising. The action object images $I(O_k, S)$ as well as the goal object images $I(O_g, S)$ are encoded by a CNN. The discrete action/goal symbols $\bar{a}, \bar{g}$ are encoded by fully connected layers with a one-hot encoding as input. As the only information the network has access to is the *initial* configuration of the scene, a RNN takes the current encoding of step k and the past encodings, which it has to learn to represent in its hidden state h_{k-1}, into account. Therefore, the network has to implicitly generate its own predictive model about the effects of the actions, without explicitly being trained to reproduce some future state. The symbolic goal $\bar{g}$ and its corresponding goal object image $I(O_g, S)$ are fed into the neural network at each step, because it is constant for the complete task. The weights of the CNN action–object image encoder can be shared with the CNN of the goal–object image encoder, because they operate on the same set of object images. To summarize,

$$\begin{aligned}(p_\pi, h_k) &= \pi_{\mathrm{NN}}(\bar{a}_k, I(O_k, S), \bar{g}, I(O_g, S), h_{k-1}) \\ &= \pi(a_k, g, a_{1:k-1}, S).\end{aligned} \tag{8}$$

4.5. Relation to Q-functions and offline RL

In principle, one can view the way we define π in (4) and how we propose to train it with the transformation (7) as learning a goal-conditioned Q-function in a partially observable Markov decision process (POMDP), where a binary reward of 1 is assigned if a complete action sequence is feasible and reaches the symbolic goal. As the learning is performed based on an offline collected dataset, one can interpret our work as offline or batch RL.

However, we want to clarify that the decision process π has to reason about cannot directly be associated with an underlying continuous state of the manipulation system. A sequence of discrete actions parameterizes the constraints of the trajectory optimization problem. These constraints do usually not fully specify the behavior of the motion within a phase. Otherwise, no trajectory optimization would be necessary. Instead, the optimization problem tries to find the remaining degrees of freedom such that all constraints implied by the action sequence can globally be fulfilled.

Therefore, the Q-function (and π) should be understood as a Q-function for the abstract POMDP of making decisions such that a feasible optimization problem is obtained. Although the symbolic state transition dynamics (1f) is known through the logic, it is not known a priori whether a certain choice of actions leads to a feasible optimization problem without attempting to solve it, i.e., executing the discrete decisions, which is in parts caused by the fact that the problems are non-convex.

To illustrate that, we show in Figure 3 the fact that future discrete actions can change the resulting optimized trajectory in earlier motion phases.

This raises the question which state/action representation is appropriate for learning such a Q-function for the abstract decision process, because it has to be conditioned on the objects in the scene and their geometry. In particular, the only state information available is the initial scene. Taking discrete actions does not lead to well-defined new observations in terms of new input images or robot configurations.

We propose the action–object image sequence as a powerful input representation for such a Q-function for multiple reasons. First, it would not be sufficient to base the representation on the symbolic state alone, because the symbolic state does not contain sufficient information, neither about the geometry of the objects, their geometric realizations, nor the effects of all past decisions on the NLP. For example, picking an object up and placing it again on the same table does not change the symbolic state, but the resulting continuous state can then be completely different. Therefore, the history of the symbolic state needs to be taken into account.

Furthermore, it is difficult to encode the symbolic state as an input, especially for multiple objects. Therefore, we implicitly encode the state via the *sequence* (or history) of actions. This is not only sufficient, because a sequence of actions uniquely determines the symbolic state sequence, but advantageous, because actions explicitly only depend on a small subset of objects, as compared with the complete symbolic state. However, as discussed previously, the

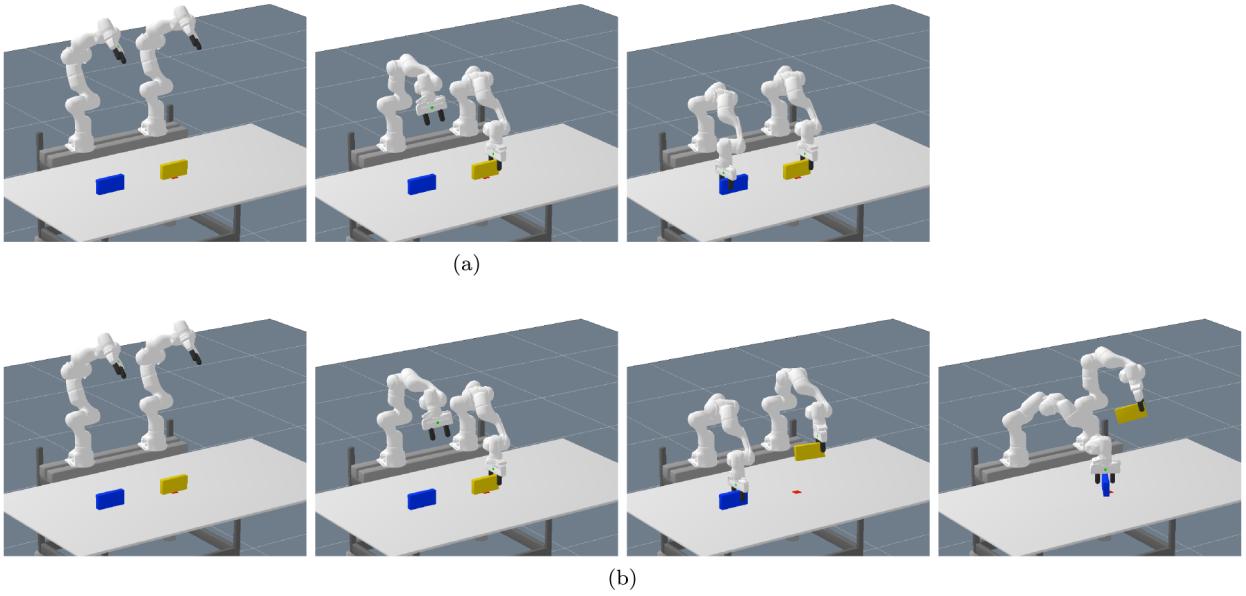


Figure 3. Illustration that the behavior of the optimized trajectory is only fully defined for a complete sequence of discrete actions. (a) Action sequence grasp(R O_{yellow}), grasp(L O_{blue}), (b) Action sequence grasp(R O_{yellow}), grasp(L O_{blue}), place(L O_{blue} O_{goal}). Both (a) and (b) have the same first two discrete actions. However, in (b) (bottom row), the fact that there is a third discrete action influences the trajectory in the past (third image from the left). This is because the actions impose constraints that only partially determine the motions. The optimizer then tries to find a motion that is globally consistent with all constraints. Hence, the yellow object is moved away from the goal location already in the second phase of the motion although no discrete action explicitly told it to do so. Further, one can also see that although in the first motion phase (second image from the left) only constraints for grasping the yellow object are present, the left arm is already moving.

actions also require the complete initial scene information, which is provided in terms of the image of the whole scene. The fact that the action–object images contain geometry information of the scene couples the abstract symbolic level to the concrete scene instance, making the reasoning process possible.

In summary, our network has to learn a state representation for the abstract decision process from the past action–object image sequence, while only observing the initial state in form of the depth image of the scene as input, conditioned on the goal.

Through the data transformation (7), we can frame learning π or the Q-function as a (stable) supervised learning problem on offline collected data.

4.6. Algorithm

Algorithm 1 presents the pseudocode of how π is integrated in the tree search algorithm.

The main idea of the algorithm is to maintain the set E of expand nodes. A node $n = \left(s, (\bar{a}, O), k, p_\pi, h, n_{\text{parent}}\right)$ in the tree consists of its symbolic state $s(n)$, action–object pair $(\bar{a}, O)(n)$, depth $k(n)$, i.e., the current sequence length, the prediction of the neural network $p_\pi(n)$, the hidden state of the neural network $h(n)$, and the parent node $n_{\text{parent}}(n)$. We write $\cdot(n)$ to denote an entry of the node tuple.

At each iteration, the algorithm chooses the node n_E^* of the expand set where the network has the highest prediction (line 5), i.e., where the network predicted that choosing this action leads to an overall feasible solution for the task goal g in the future. For all possible next actions, i.e., children of n_E^*, the network is queried to predict their probability leading to a feasible solution, which creates new nodes (line 10).

If a child node reaches a symbolic goal state (line 11), it is added to the set of leaf nodes L, otherwise to the expand set.

Then those already found leaf nodes from the set L are investigated. This set of leaf nodes contains the candidates for a feasible solution whose corresponding trajectory optimization problems have not yet been solved and checked for feasibility (L is related to a subset of $\mathcal{T}(g, S)$). In line 18, the algorithm chooses the leaf node n_L^* from L with the highest prediction of the neural network as the first for which the trajectory optimization problem is solved (line 25). If the NLP $P\big((\bar{a}, O)_{1:K}(n_L^*), (\bar{g}, O_g), S\big)$ corresponding to the action sequence of the leaf node n_L^* is feasible, a solution of the overall TAMP problem in terms of the trajectory $x = \arg P\big((\bar{a}, O)_{1:K}(n_L^*), (\bar{g}, O_g), S\big)$ in the configuration space $\mathcal{X}$ is found. If it is not feasible and there are still nodes in L, the one with the highest prediction of the remaining ones is tested for feasibility by solving the

Algorithm 1 LGP with Deep Visual Reasoning

1: **Input:** Scene S, goal g and max sequence length $K_{\max}$
2: $L = \emptyset$ ▷ set of leaf nodes
3: $E = \{n_0\}$ ▷ set of nodes to be expanded, n_0 is root node
4: **while** no solution found **do**
▷ choose node from expand set with highest prediction
5: $n_E^* = \underset{n \in E \wedge k(n) < K_{\max}}{\arg\max} \; p_\pi(n)$
6: $E \leftarrow E \backslash \{n_E^*\}$
7: **for all** $(\bar{a}, O) \in \mathcal{AO}(s(n_E^*), S)$ **do**
8: $(p_\pi, h) = \pi_{\text{NN}}\left(\bar{a}, I(O, S), \bar{g}, I(O_g, S), h(n_E^*)\right)$
9: $s = \text{succ}(s(n_E^*), (\bar{a}, O)), k = k(n_E^*) + 1$
10: $n = \left(s, (\bar{a}, O), k, p_\pi, h, n_E^*\right)$ ▷ new node
11: **if** $s \in \mathcal{S}_{\text{goal}}(g)$ **then**
12: $L \leftarrow L \cup \{n\}$ ▷ if goal state, add to leaf node set
13: **else**
14: $E \rightarrow E \cup \{n\}$ ▷ if no goal state, add to expand set
15: **end if**
16: **end for**
17: **while** $|L| > 0$ **do** ▷ consider already found leaf nodes
▷ choose node from leaf node set with highest prediction
18: $n_L^* = \arg\max_{n \in L} p_\pi(n)$
19: **if** $p_\pi(n_L^*) \leqslant f_{\text{thresh}}$ **then**
20: $f_{\text{thresh}} \leftarrow \text{adjustFeasibilityThreshold}(f_{\text{thresh}})$
21: **break**
22: **end if**
23: $L \leftarrow L \backslash \{n_L^*\}$
24: $(\bar{a}, O)_{1:K} = (\bar{a}, O)_{1:k(n_L^*)}(n_L^*)$ ▷ extract action seq.
25: solve NLP $x = \arg P\left((\bar{a}, O)_{1:K}, (\bar{g}, O_g), S\right)$
26: **if** feasible, i.e., $F_S((\bar{a}, O)_{1:K}) = 1$ **then**
27: solution $(\bar{a}, O)_{1:K}$ with trajectory x found
28: **break all and return** solution x, $(\bar{a}, O)_{1:K}$
29: **end if**
30: **end while**
31: **end while**

corresponding NLP. Otherwise, the expansion of the tree continues up to a maximum sequence length $K_{\max}$.

4.6.1. Feasibility threshold and completeness of the algorithm. As during the expansion of the tree, leaf nodes which are unlikely to be feasible are also found, only those trajectory optimization problems are solved where the prediction p_π is higher than the feasibility threshold f_{thresh} (set to 0.5 in the experiments). Otherwise, if the highest prediction in the set L is below this threshold, the investigation of the leaf nodes is stopped (lines 19 and 21) until new leaf nodes are found in the expansion step.

This greatly reduces the number of NLPs that have to be solved, because the information provided by the network is not only used to guide the tree search to find promising leaf nodes quickly, but also to discard leaf nodes that are predicted to be infeasible by the network.

However, one cannot expect that the network never erroneously has a low prediction although a leaf node would be feasible. In order to prevent not finding a feasible solution in such cases, the function adjustFeasibilityThreshold(·) (line 20) reduces this threshold with a discounting factor $\gamma < 1$, i.e., $f_{\text{thresh}} \rightarrow \gamma \cdot f_{\text{thresh}}$, or sets it to zero if all leaf nodes with a maximum depth of $K_{\max}$ have been found. This allows us to provide a completeness guarantee of the algorithm, under the following assumption.

Assumption 1. *If a scene for a given goal contains at least one action sequence up to a maximum length $K_{\max}$ that corresponds to a feasible nonlinear program, the trajectory optimizer numerically converges to a feasible solution for that action sequence.*

Proposition 1. *Under Assumption 1, Algorithm 1 is complete.*

Proof. Clear by construction of Algorithm 1 and the feasibility threshold discounting through adjustFeasibility *Threshold*(·). □

This means that if a scene contains at least one action sequence that reaches the symbolic goal for which the nonlinear trajectory optimizer can find a feasible motion, Algorithm 1 finds this solution. In particular, the neural network does not prevent finding a solution, even in case of prediction errors. Another way to interpret Proposition 1 is that the network π together with the threshold mechanism is a goal-conditioned admissible heuristic for a tree search algorithm.

In Assumption 1 we assumed a maximum sequence length $K_{\max}$ to be given for which the task can be solved. If $K_{\max}$ is not known, one can simply wrap Algorithm 1 in an outer loop that increases $K_{\max}$ and add a stopping criteria for line 4 when $K_{\max}$ is reached. Then Algorithm 1 is also complete for an unknown $K_{\max}$.

If we set f_{thresh} to a negative value, then the algorithm is automatically complete without the adjustment of the threshold, under the penalty of potentially solving unnecessary NLPs by not exploiting all the information π can provide. In the experiments in Section 5.2.8, we investigate the influence of the threshold and its adjustment both on completeness and performance.

4.6.2. Implementation. As an important remark, for the implementation we store the hidden state of the RNN in its corresponding node. Furthermore, the action–object images and action encodings also have to be computed only once, because all action–object images correspond to the initial scene configuration. Therefore, during the tree search, only one pass of the recurrent (and smaller) part of the complete π_{NN} has to be queried for each new child node, which is very fast.

4.7. Comparative alternative: goal-independent recurrent feasibility classifier

The methods of Driess et al. (2020b) and Wells et al. (2019) learn a feasibility classifier for single actions only, i.e., independent of a goal, and do not consider sequences. To allow for a comparison, we present in this section an approach to extend the idea of a feasibility classifier to

action sequences and how it can be integrated into our TAMP framework.

The basic idea is to classify the feasibility of a motion planning problem that results from an action sequence with a recurrent classifier, independently from and agnostic to a goal. In this way, during the tree search, solving an NLP as a lower bound to guide the search can be replaced by evaluating the classifier, which usually is order of magnitudes faster. Formally, we seek to learn a function that, given a scene description S and the past decisions $a_{1:K-1}$, predicts whether choosing an action a_K at the time step K leads to a feasible NLP for the resulting action sequence $a_{1:K}$

$$\pi_{\mathrm{RC}}(a_K, a_1, \ldots, a_{K-1}, S) = p(F_S(a_{1:K}) = 1 | S). \tag{9}$$

This is independent from the overall goal of the TAMP problem. Instead, π_{RC} should predict the feasibility for arbitrary action sequences $a_{1:K}$ of different lengths K. As it therefore also operates on subsequences that do not reach the symbolic goal yet, querying π_{RC} can be interpreted as a heuristic that has a similar role as the lower bounds to guide the tree search. Note that this classifier is slightly different to a discriminative model that is trained only on complete goal-reaching sequences.

Regarding the dataset for training such a classifier, assume that we have sampled a scene S^i and a goal-reaching action sequence $a^i_{1:K^i} \in \mathcal{T}(g, S)$. Then we include in the dataset not only the feasibility of $\left(S^i, a^i_{1:K^i}, F_{S^i}(a_{1:K^i})\right)$, but also the feasibility of the subsequences up to $\tilde{K}^i \leqslant K^i$, i.e.,

$$\mathcal{D}_{\mathrm{RC}} = \bigcup_{i=1}^{n} \bigcup_{j=1}^{\tilde{K}^i} \left\{ \left(S^i, a^i_{1:j}, F_{S^i}(j) \right) \right\} \tag{10}$$

with

$$F^i_{S^i}(j) = F_{S^i}\left(a^i_{1:j}\right) \tag{11}$$

where $F_{S^i}(0) = 1$ and the index

$$\tilde{K}_i = \max\{ j \leqslant K^i : F^i_{S^i}(j) = 1 \vee \left(F^i_{S^i}(j-1) = 1 \wedge F^i_{S^i}(j) = 0 \right) \}. \tag{12}$$

Superscripts here denote dataset indexing. For the actual implementation, we basically choose the same architecture as for π_{NN}, but $\pi_{\mathrm{RC_{NN}}}$ only takes the current action–object image pair as well as the hidden state of the previous step as input and predicts whether the action sequence up to this step is feasible, i.e.,

$$\pi_{\mathrm{RC_{NN}}}(\bar{a}_k, I(O_k, S), h_{k-1}) = \pi_{\mathrm{RC}}(a_K, a_1, \ldots, a_{K-1}, S). \tag{13}$$

Section 5.2.6 presents an empirical comparison of this recurrent classifier with the goal-conditioned predictor π_{NN}.

4.8. Predicting trajectory cost

So far, we were only interested in finding a feasible action sequence. The full LGP formulation (1), however, seeks not only for feasibility, but also optimality with respect to the trajectory costs measured by the function c in (1a). For typical pick-and-place scenarios as considered in Section 5.2, the trajectory costs penalize accelerations of the robot joints. Therefore, the trajectory costs mainly depend on the action sequence length K. Owing to the way the network is integrated into the tree search algorithm, cf. Section 4.6 and Algorithm 1, it empirically turned out that in many cases the network already finds solutions with low costs.

However, for the more dynamic pushing experiment of Section 5.3, there can be more significant differences in the trajectory costs even for action sequences of the same sequence length. Therefore, we present in this section a simple modification of the network and especially the training targets to enable the network to predict the expected trajectory cost for an action sequence. This way, we can utilize the network to find more cost optimal solutions compared to the case where we only take feasibility into account.

Let $P(a_{1:K}, g, S)$ denote the cost of the NLP (1) when fixing the action sequence $a_{1:K}$. In order to represent infeasibility as a finite value, we exponentiate the cost, hence an infeasible sequence has a value of $\exp(-\infty) = 0$. Therefore, the network, which we call cost prediction π_{cost}, should be trained to predict for the scene S, the goal g, and the past decisions $a_{1:k-1}$ at step k of the sequence whether an action a_k is promising in the sense of there exist future actions $a_{k+1:K}$ such that the complete sequence $a_{1:K}$ not only leads to a feasible solution, but also the minimum achievable cost of a complete future sequence when following a_k, i.e.,

$$\begin{aligned} \pi_{\mathrm{cost}}(a_k, g, a_1, \ldots, a_{k-1}, S) = \\ \max_{a_{k+1:K}} \exp(-P(a_{1:K}, g, S)) \\ \text{s.t.} \quad F_S(a_{1:K}) = 1 \\ a_{1:K} \in \mathcal{T}(g, S). \end{aligned} \tag{14}$$

The training targets of the i th data point for this network are now a sequence of positive values $f^i \in \mathbb{R}^{K_i}_{\geqslant 0}$, where their j th component $f^i_j \geqslant 0$ is defined as

$$f^i_j = \max_{\substack{(S^l, a^l_{1:K^l}, g^l, F^l) \in \mathcal{D}_{\mathrm{data}} \\ F^l = F_{S^l}(a^l_{1:K^l}) = 1 \\ g^l = g^i \\ a^l_{1:j} = a^i_{1:j}}} \exp(-P(a^l_{1:K^l}, g^l, S^l)) \tag{15}$$

If there exists no future feasible action sequence when taking a_k or $a_{1:k}$ is already infeasible, $f^i_j = 0$, because we defined $P(a_{1:K}, g, S) = \infty$ for an infeasible action sequence.

Analogously to the discussion in Section 4.5, the role of π_{cost} is very similar to the cost-to-go prediction expressed by a Q-function, where there are only terminal costs (the

costs of the trajectory optimization problem for a complete action sequence). Instead of performing Q-iteration, we calculate this cost-to-go on the dataset explicitly, which leads to a stable supervised learning problem. Although the feasibility network π_{NN} has a sigmoid output and is trained with a weighted binary cross-entropy loss, the cost prediction network π_{cost} has a linear output and is trained with a squared error loss. The network architecture, i.e., especially the input encoding and the recurrent structure, of π_{NN} and π_{cost} is the same, with the only difference of the output, as mentioned.

When integrating π_{cost} in the tree search algorithm, the exact same algorithm as for the feasibility network can be used. However, the notion of the feasibility threshold f_{thresh} and its adjustment (lines 19 and 20 in Algorithm 1) has to be changed slightly to reflect the fact that the outputs of π_{cost} are now not probabilities but negatively exponentiated costs and a value of zero corresponds to infeasibility. Therefore, if π_{cost} is below a certain value, the sequence is classified as infeasible.

5. Experiments

We demonstrate our proposed framework on two different tasks. Please also refer to Extension 1 that demonstrates the planned motions both in simulation and with a real robot.

For the first experiment, presented in Section 5.2, we consider a typical kinematic pick-and-place TAMP problem where two robot arms have to collaborate to solve the tasks. We investigate the generalization capabilities of our approach to more objects in the environment than during training.

In the second experiment (Section 5.3), we show that the method can also be applied to tasks that involve dynamic models. More specifically, we demonstrate a tool-use scenario where the robot has to use a hook-shaped object to push and/or pull an object to a desired target location.

As a general remark, the quantitative results are visualized using boxplots that show the median, the upper and lower quartiles as well as whiskers. The whiskers in all boxplots correspond to datapoints that are nearest to 1.5 times the interquartile range (IQR). When we obtained time measurements, we ensured to run them on the same machine which did not compute anything else at the time. All experiments in Section 5.2 that report run times have been performed with an Intel Xeon W-2145 CPU @ 3.70 GHz, whereas in Section 5.3 they were run with an Intel Xeon E5-2630v4 CPU @ 2.20 GHz.

5.1. Network details

Both experiments share the same network architecture with the same hyperparameters. The network is trained with the ADAM optimizer (learning rate 0.0005) with a batch size of 48 for the pick-and-place experiment and 40 for the pushing experiment. To account for the aforementioned imbalance in the dataset, we oversample feasible action sequences such that at least 16 out of the 48 samples in one batch come from a feasible sequence for the pick-and-place experiment (8 out of 40 for the pushing one). More specifically, 32 are sampled without replacement from the whole dataset, whereas the additional 16 (or 8) are solely sampled from the feasible ones with replacement. We additionally weight the feasible samples in the loss function with a weight of 2. In general, accounting for the imbalance with this oversampling turned out to be crucial for the performance. However, there was no extensive tuning of hyperparameters necessary at all.

The image encoder consists of three convolutional layers with 5, 10, and 10 channels, respectively, and filter size of 5×5. The second and third convolutional layers have a stride of 2. After the convolutional layers, there is a fully connected layer with an output feature size of 100 and linear activation. The inner layers are followed with rectified linear unit (ReLU) activations. The same image encoder with shared weights is used to encode the action images and the goal image. The discrete action encoder is one fully connected layer with 100 neurons and ReLU activations. The recurrent part consists of one layer with 300 GRU cells, followed by a linear layer with output size 1 and a sigmoid activation as output for π or linear activation for π_{cost}. Except for the output, π and π_{cost} have the exact same architecture. As the task is always to place an object at or to push it to varying locations, we left out the discrete goal encoder in the experiments presented here.

5.2. Pick-and-place experiment

For the first experiment, we consider a tabletop scenario with two robot arms (Franka Emika Panda) and multiple box-shaped objects, see Figure 1a, 4, or 5 for typical scenes, in which the goal is to move an object to different target locations. The target locations are visualized by red squares in these figures.

5.2.1. Action operators and optimization objectives. The logic language is defined by rules similar to PDDL (Planning Domain Definition Language). For the pick-and-place experiment, there are two action operators `grasp(R` η O`)` and `place(R` O_a O_b`)`.

The `grasp` action takes as parameters the robot arm $\texttt{R} \in \{\texttt{left}, \texttt{right}\}$, one of four integers $\eta \in \{1, 2, 3, 4\}$, and a (single) box-shaped object $O \in \mathcal{P}(\mathcal{O}(S))$ that should be grasped. A precondition ensures that O is an object. The robot arm and integer assignment are represented in the discrete action symbol $\bar{a}$, leading to eight different discrete action symbols $\bar{a}$ for the grasp action. As discussed in Section 4.3, the object O is encoded in the action–object image $I(O, S)$. See also Figure 1 for a visualization of the action–object images for the two action operators. The `grasp` action imposes the following constraints on the phase of the trajectory where it is active. The end-effector `R` is always aligned vertically to the box. Depending on the integer, the grippers are additionally aligned in parallel to

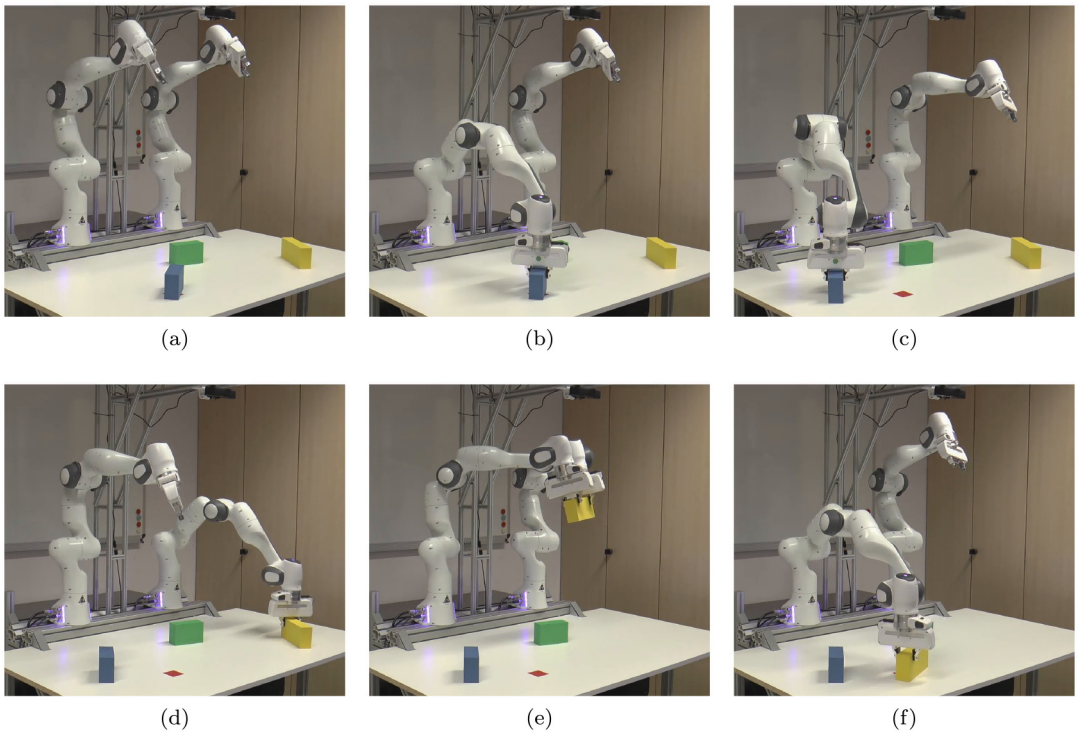


Figure 4. Typical scene of the pick-and-place experiment: (a) initial scene from which the image is captured; (b) action 1 (grasp), remove occupying object; (c) action 2 (place); (d) action 3 (grasp); (e) action 4 (grasp), handover; (f) action 5 (place), goal achieved. The yellow object should be placed on the red spot, which is, however, occupied by the blue object. Furthermore, the yellow object cannot be reached by the robot arm that is able to place it on the red spot. Therefore, the two arms have to collaborate to solve the task. In this case, the network decides for a handover solution.

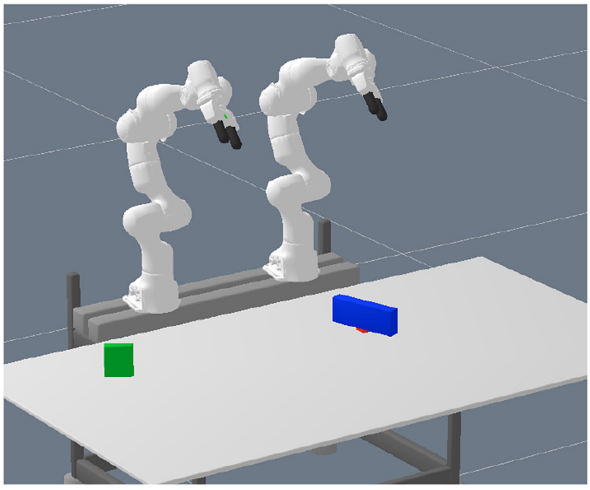


Figure 5. Training scene example of the pick-and-place experiment. The sizes, positions, and orientations of the two boxes as well as the target location (red spot) are sampled randomly. Here the green box should be placed on the red spot.

different surfaces of the box through two equality constraints. Furthermore, an inequality constraint ensures that the center point between the two grippers of the end-effector is inside of the object with a margin. In this work, we only consider grasps from the top, leading to four different discrete ways of grasping a box from the top. Figure 6 visualizes these four discrete ways of grasping for one robot arm. One might think that only two of the four grasps (the first and third from the left in Figure 6) are necessary for symmetry reasons. However, owing to the kinematic limits of the robot arms, all four are indeed required for this task. The coordinate system to define the alignments is defined from a robot perspective, which avoids symmetry issues of the box looking the same in the image if rotated by, e.g., 180° and therefore uniquely defines each discrete grasp type.

Note that the exact grasping location in two degrees of freedom relative to the object is not defined completely by the `grasp` action and therefore subject to the optimizer. Handover motions are just two consecutive `grasp` actions, for which the optimizer takes care of finding a suitable handover pose, if a handover is feasible. As seen, for example, in Figure 4b, when an object should be grasped by only one arm, then the optimizer grasps it in the middle of the object. In contrast, for a handover, as seen in Figure 4e, the fact that there are two consecutive grasp actions on the same object in the action sequence leads to the first grasp to happen near the boundary of the box to allow the other arm to grasp the box later during the handover as the second grasp. This capability for generating handovers is

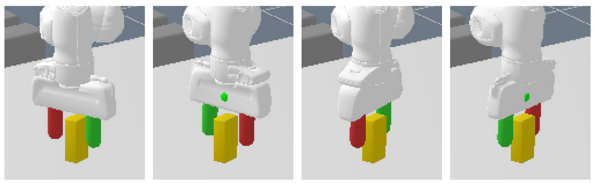


Figure 6. The four different integer assignments of the `grasp` operator for the pick-and-place experiment. The gripper is always aligned vertically to the box, but the integers determine which of the side of the box the grippers are parallel to. Owing to the kinematic limits, all four ways have to be included.

another reason why π is not a standard Q-function (see the discussion in Section 4.5).

Finally, the `grasp` action imposes the constraint on the relative velocity between the end-effector and the object to be zero, modeling a stable grasp, during the time it is active.

The `place` action has as parameters the robot arm $\texttt{R} \in \{\texttt{left}, \texttt{right}\}$ as well as two objects O_a and O_b, $(O_a, O_b) \in \mathcal{P}(\mathcal{O}(S))$. The robot arm again is expressed in the discrete action symbol $\bar{a}$, i.e., there are two `place` action symbols. The object O_b on which O_a should be placed is encoded in the action–object image $I((O_a, O_b), S)$. The effects of the `place` action on the optimization objectives are that the bottom surface of the object O_a touches and is parallel to O_b. In our case, we have preconditions that O_a is a box and O_b is a table or the goal location (although this is not strictly necessary). Similar to the `grasp` action, the `place` action does not specify where exactly to place the object. It is only specified that it should be placed somewhere on the O_b. The optimizer then chooses a placement position that is consistent with the other constraints imposed by the past and future actions in the sequence, see, for example, Figure 4c.

Preconditions for `grasp` and `place` ensure that one robot arm attempts to grasp only one object simultaneously and that an arm can only place an object if it is holding one.

Path costs c penalize squared accelerations of the robot joints of the path x. Finally, there are collisions and joint limits as inequality constraints with no margin.

In total, $|\mathcal{A}| = 10$, i.e., there are 10 discrete actions operator symbols. The number of objects and therefore the search space over the action sequences or the size of the LGP tree depends on the number of objects $|\mathcal{O}(S)|$ in the scene (Table 1).

5.2.2. Properties of the scene. There are multiple properties which make this (intuitively simple) task challenging for TAMP algorithms. First, the target location can fully or partially be occupied by another object. Second, the object and/or the target location can be out of reach for one of the robot arms. Hence, the algorithm has to figure out which robot arm to use at which phase of the plan and the two robot arms possibly have to collaborate to solve the task. Third, apart from position and orientation, the objects vary in size, which also influences the ability to reach or place an object. In addition, grasping box-shaped objects introduces a combinatorics that is not handled well by nonlinear trajectory optimization due to local minima and also joint limits. Therefore, as described in the last paragraph, we introduce integers as part of the discrete action that influence the grasping geometry. This greatly increases the branching factor of the task. For example, depending on the size of the object, it has to be grasped differently or a handover between the two arms is possible or not, which has a significant influence on the feasibility of action sequences.

Indeed, Table 1 lists the number of action sequences with a certain length that lead to a symbolic goal state over the number of objects in the scene. This number corresponds to *candidate* sequences for a feasible solution (the set $\mathcal{T}(g, S)$) which demonstrates the great combinatorial complexity of the task, not only with respect to sequence length, but also number of objects. Only a very small subset of these candidate sequences actually correspond to a feasible trajectory optimization problem, cf. Section 5.2.3. Furthermore, it also shows that one cannot expect to generate a dataset which covers the complete search tree for a single scene configuration. This means that a dataset can neither contain a dense coverage of the scene variation, nor all possible action sequences for each individual scene.

One could argue that an occupied and reachability predicate could be introduced in the logic to reduce the branching of the tree. However, this requires a reasoning engine which decides those predicates for a given scene, which is not trivial for general cases. More importantly, both reachability and occupation by another object is something that is also dependent on the geometry of the object that should be grasped or placed and, hence, not something that can be precomputed in all cases (Driess et al., 2020b, 2019b). For example, if the object that is occupying the target location is small and the object that should be placed there as well, then it can be placed directly, whereas a larger object that should be placed requires to first remove the occupying object. Our algorithm does not rely on such non-general simplifications, but decides promising action sequences based on the real relational geometry of the scene, encoded in the action–object images and the goal image.

5.2.3. Training/test data generation. We generated 30,000 scenes randomly with two objects present at a time. The sizes, positions, and orientations of the objects as well as the target location are sampled uniformly within a certain range. In total, the parameter scene space is 14-dimensional. See Figure 5 for an example of a typical training scene. For half of the scenes, one of the objects (not the one that is part of the goal) is placed directly on the target, to ensure that at least half of the scenes contain a situation where the target location is occupied. The dataset $\mathcal{D}_{\text{data}}$ is

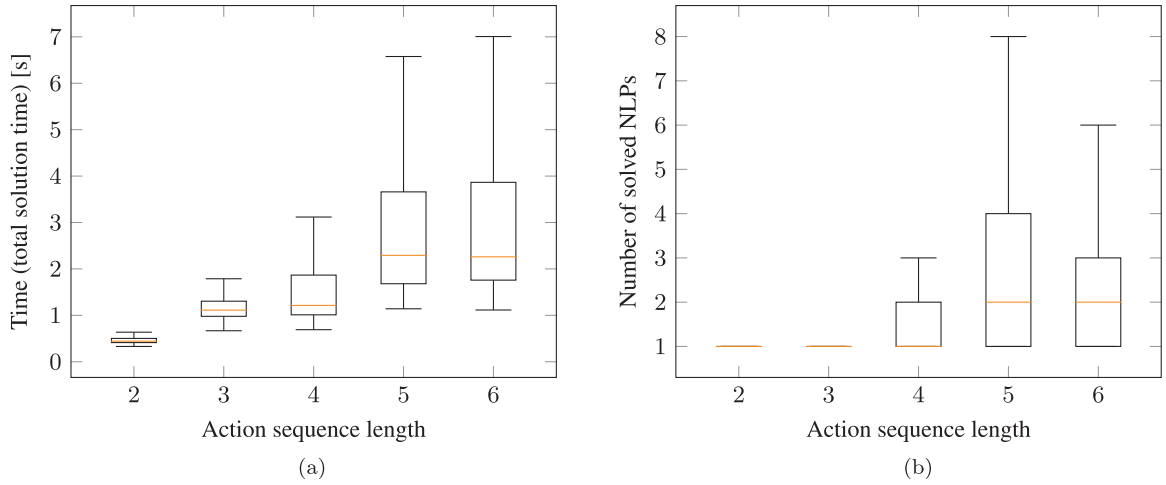


Figure 7. Performance on test scenarios for pick-and-place experiment (two objects in the scene) with neural network integrated into the tree search algorithm: (a) total time to find a feasible solution; (b) number of NLPs that have to be solved to find a solution. As can be seen in (b), in many cases only a single NLP has to be solved or 2 (median) for more challenging tasks that require action sequence lengths of 5 and 6.

determined by a breadth-first search for each scene over the action sequences, until either 4 solutions have been found or 1,000 leaf nodes have been considered. In total, for 25,736 scenes at least one solution was found, which were then the scenes chosen to create the actual training dataset $\mathcal{D}_{\text{train}}$ as described in Section 4.2. A total of 102,566 of the action sequences in $\mathcal{D}_{\text{data}}$ that reach the symbolic goal were feasible, 2,741,573 completely infeasible. This shows the claim of the introduction and Section 5.2.2 that the majority, namely 96.4%, of the candidate action sequences are actually infeasible. Furthermore, such an imbalance between feasible and infeasible sequences imposes difficulties for a learning algorithm. With the data transformation from Section 4.2, there are 7,926,696 $f_j = 0$ and 1,803,684 $f_j = 1$ training targets in $\mathcal{D}_{\text{train}}$, which is more balanced. Still, as mentioned in Section 5.1, oversampling the feasible action sequences was necessary to ensure that enough $f_j = 1$ training targets are presented to the network.

To evaluate the performance and accuracy of our method, we sampled 3,000 scenes, again containing 2 objects each, with the same algorithm as for the training data, but with a different random seed. Using breadth-first search, we determined 2,705 feasible scenes, which serve as the actual test scenes.

5.2.4. Performance: results on test scenarios. This section presents the performance of our network when integrated into the tree search algorithm to find solutions for the 2,705 test scenarios that contain 2 objects each.

Figure 7 shows both the total runtime and the number of NLPs that have to be solved to find a feasible solution. When we report the total runtime, we refer to everything, meaning capturing the image, computing the image/action encodings, querying the neural network during the search and the time for all involved NLPs that are solved. As one can see in Figure 7b, for *all* cases with sequence lengths of 2 and 3, the first predicted action sequence is feasible, such that there is no search necessary and only one single NLP has to be solved. For length 3, the median is still 1, but also for sequences of lengths 5 and 6 in half of the cases less than 2 NLPs have to be solved.

In general, with a median runtime of about 2.3 s for even sequence length of 6, the overall framework with the neural network has a high performance and the task can be solved in reasonable runtime. Furthermore, the upper whiskers are also below 7 s.

Of the 2,705 test scenes, in 35% of the cases the network finds a solution with sequence length 2, in 18% of length 3, in 27% of length 4, in 7% of length 5, and in 13% of length 6. Therefore, it is important that we report the runtimes and the number of solved NLPs separated as a function of the sequence length. Otherwise, the simpler cases of sequence length 2 and 3, where there is a smaller combinatorial complexity, would cover 53% of all test scenarios. This allows us to show that also for the harder cases of sequence length 5 and 6, which are only 20% of the test scenes, the network performs well.

5.2.5. Comparison with multi-bound LGP tree search. Here we compare the performance of multi-bound LGP tree search that utilizes the lower bounds from Toussaint and Lopes (2017) as heuristics to guide the search with our proposed framework where the network acts as a heuristic or ideally directly predicts a feasible action sequence.

In Figure 8a the runtimes for solving the test cases with LGP tree search are presented, which shows the difficulty of the task. In 132 out of the 2,705 test cases, LGP tree search is not able to find a solution within the timeout, compared with only 3 times when we use our proposed framework with the neural network.

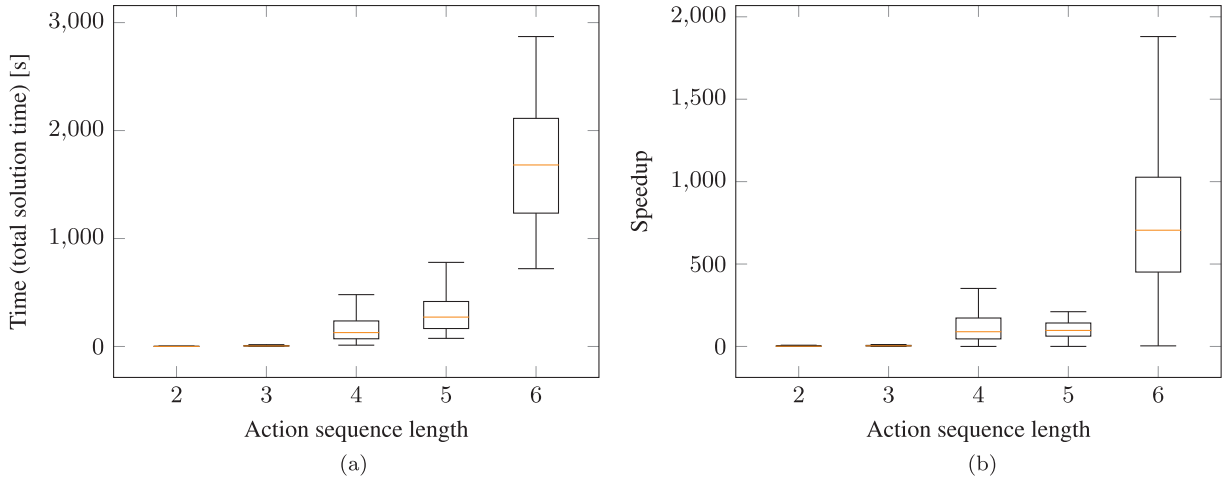


Figure 8. Comparison of our framework with the neural network to multi-bound LGP tree search that relies on computing lower bounds to guide the search: (a) total time with LGP to find a feasible solution; (b) speedup gained when using our proposed network over multi-bound LGP tree search. The speedup in (b) is the solution time with multi-bound LGP tree search divided by the solution time with the neural network. Test scenes of pick-and-place experiment with two objects in the scene.

Figure 8b shows the speedup that is gained by using the neural network. For sequence length 4, the network is 46 times faster, 100 times for length 5, and for length 6 even 705 times faster (median). In this plot, only those scenes where LGP tree search and the neural network have found solutions with the same sequence lengths are compared, which is the case in 78% of the test scenes. This means that especially for the difficult cases, where it is most relevant, utilizing the network also leads to a significant speedup.

5.2.6. Comparison with the recurrent classifier. Figure 9a shows a comparison of our proposed goal-conditioned network that generates sequences with the recurrent classifier described in Section 4.7 that only predicts the feasibility of an action sequence, independent from the task goal. As one can see, although such a classifier also leads to a significant speedup compared with LGP tree search, our goal-conditioned network has an even higher speedup, which also stays relatively constant with respect to increasing action sequence lengths. Furthermore, with the classifier 22 solutions have not been found, compared with 3 with our approach within the timeout. Although the network query time is neglectable for our network, as can be seen in Figure 9b, the time to query the recurrent classifier becomes visible. Indeed, one can see the exponential increase in the network query time for increasing sequence lengths. This is caused by the fact that without goal conditioning, the network has to be queried much more often, because it can only assess the feasibility of an action sequence up to the point it has been queried without the ability to judge whether it eventually leads to the goal. Therefore, the speedup compared with LGP tree search with the recurrent classifier is achieved by replacing the analytically defined lower bounds which can, especially if the problem is infeasible, take up to several seconds to compute, with a learned model acting as a lower bound that is not only much faster to evaluate, but also has a constant query time. The goal conditioning of our proposed framework does not show this exponential increase for longer sequence lengths, because it directly guides the search towards achieving the goal.

5.2.7. Data efficiency. One could argue that the 25,736 scenes used for training are a high number of scenes for this task. However, one also has to take into account that the parameter space from which the scenes are sampled is 14-dimensional, which means that the training set does not cover this space densely at all. Nevertheless, we trained our proposed network on a subset of 4,442 and 8,090 scenes of the original training dataset. In Figure 10, we report the number of solved NLPs to find a solution for the networks that are trained on 4,442, 8,090, and 25,736 scenes, split over the length of the found action sequence. As one can see, the median is the same for all networks. For longer sequence lengths of 5 and 6, the upper quartiles and the upper whiskers for the networks trained on the smaller datasets increase. Nevertheless, this experiment shows that with nearly six times less data, one still achieves a high performance, which is also still orders of magnitudes better than without the network.

5.2.8. Feasibility threshold and completeness. In Section 4.6, we have discussed how the network can be integrated into the tree search algorithm without preventing a solution being found if one exists even in case of prediction errors of the network. This is achieved by adjusting the feasibility threshold with a discounting factor $\gamma = 0.9$ if the network suggests to not consider a found leaf node as a promising solution candidate. Another way to realize this is to remove the feasibility threshold and just compute every NLP corresponding to a found leaf node. Here we investigate the performance impact of these strategies.

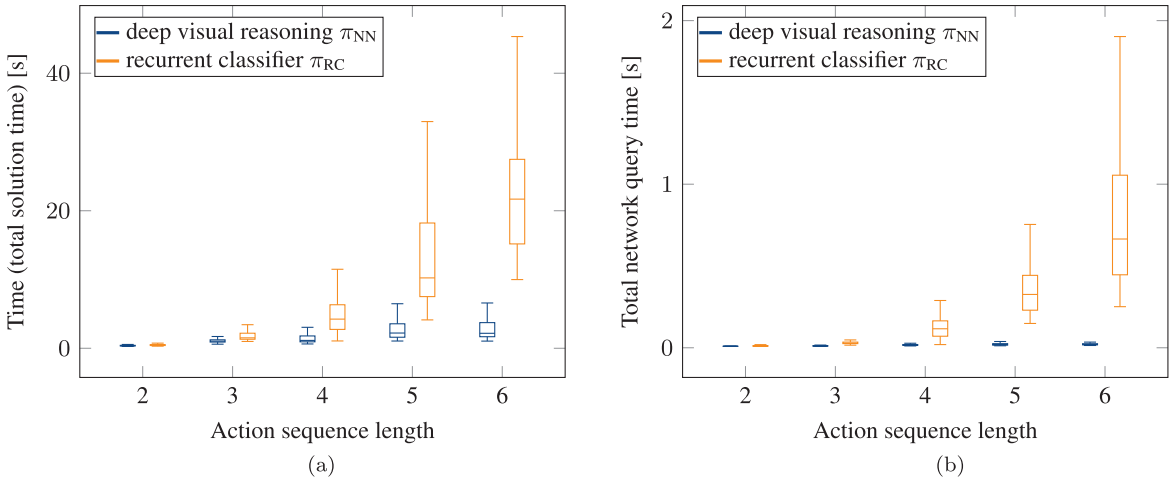


Figure 9. Comparison of recurrent classifier (orange) to our proposed goal-conditioned network (blue) for pick-and-place experiment on test scenes with two objects in the scene: (a) total time to find a feasible solution with recurrent classifier; (b) total, i.e., summed up, time to query the networks to find a solution for a scene (part of the total solution time of (a)). Although the recurrent classifier is competitive compared to LGP tree search (see Figure 8a), one can see the exponential increase in the runtime for longer sequence lengths. In contrast, the runtime with our proposed framework increases much less for longer sequence lengths.

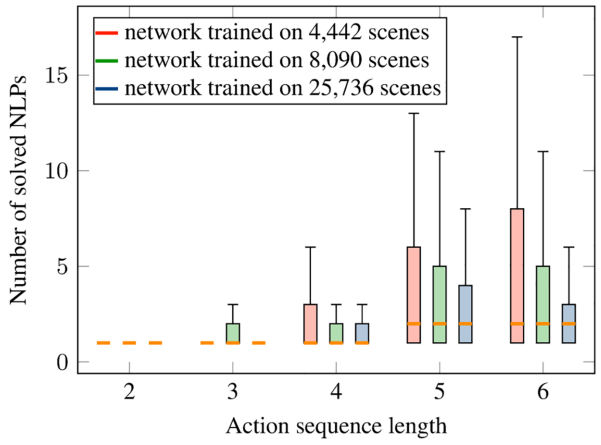


Figure 10. Investigation of the data efficiency. Number of solved NLPs to find an overall feasible solution for the test scenes with neural network integrated into the tree search algorithm for networks trained on different numbers of training scenes. Pick-and-place experiment. The median is the same for all networks.

Figure 11 shows the total solution time to find a feasible solution with the network when using the feasibility threshold together with the adjustment of the threshold (blue), which is our proposed approach, the case without the adjustment (red), which implies that solutions can be missed in case of prediction errors, and the case where there is no threshold at all (green).

As can be seen, the case without the threshold has the highest runtime, whereas the case with the threshold but no discounting has the lowest runtime. However, our proposed approach with the threshold and the discounting (blue) has only a non-significant difference to the case with threshold but no discounting. Even without the threshold, the performance is also very good, although clearly worse than with our proposed approach.

As we discussed in Proposition 1, the approach with adjusting the feasibility threshold guarantees that a solution can be found if it exists. Indeed, in only 3 of the 2,705 test cases, no solution was found with this method within the timeout. The fact that this number is not zero is due to the timeout. If we increase the timeout, then indeed a solution for all test cases was found. The same holds true for the case without the threshold, i.e., for the initial timeout 3 were not found, then with the increased timeout all were. However, even if we increased the timeout, still in 5 test cases no solution was found when using the threshold without the adjustment mechanism.

For the networks that have been trained on a smaller dataset (Section 5.2.7), adjusting the feasibility threshold turned out to be even more important. Without the feasibility adjustment, the network trained on 4,442 (8,090) scenes did not find a solution in 30 (17) cases. In contrast, with the feasibility adjustment, the networks trained on the smaller datasets did not find a solution in only 2 (1) cases. When increasing the timeout, always a solution was found in the latter case.

To summarize, the feasibility threshold adjustment presented in Section 4.6.1 maintains completeness, while showing very little performance penalty.

The threshold in the experiments was $f_{\text{thresh}} = 0.5$ and the discounting $\gamma = 0.9$.

5.2.9. Generalization to multiple objects. Creating a rich enough dataset containing combinations of different numbers of objects is infeasible. Instead, we now take the network that has been trained as described in Section 5.2.3

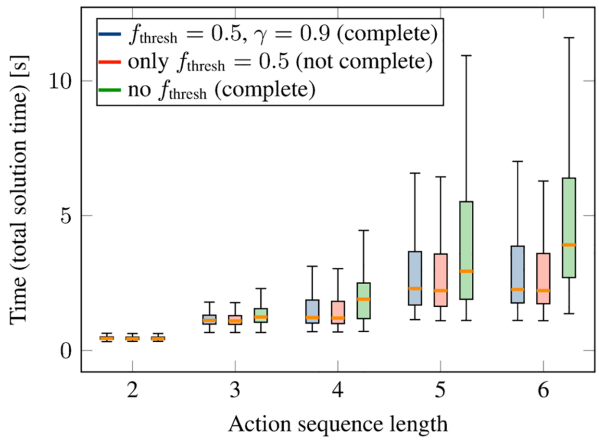


Figure 11. Performance on test scenarios depending on the feasibility threshold strategy for the pick-and-place experiment (two objects in the scene) with neural network integrated into the tree search algorithm. Blue is our proposed approach that combines the advantages of using the feasibility threshold for performance while maintaining completeness.

with only two objects present at a time and test whether it generalizes to scenes with more than two (and also only one) objects. The 200 test scenes are always the same, but more and more objects are added. In Figure 1a and 12, such test scenes with 4 and 5 objects are shown.

Figure 13 reports the total runtime to find a feasible solution with our proposed neural network over the number of objects present in the scene. These runtimes include all scenes with different action sequence lengths. In all cases a solution was found. Although the upper quartile increases, the median is not significantly affected by the presence of multiple objects.

The observation that the runtime increases for more objects is not only caused by the fact that the network inevitable makes some mistakes and, hence, more NLPs have to be solved. Solving (even a feasible) NLP with more objects can take more time due to increased runtime costs for collision queries and increased non-convexity of the optimization landscape.

In general, the performance is remarkable, especially when observing that the network was able to find solutions for scenes with many objects in a reasonable amount of time for sequence length 6, where, according to Table 1, nearly half a million possible action sequences for 5 objects in the scene exist.

To further demonstrate the advantages of the network when generalizing to multiple objects, in Figure 14, we compare the proposed approach to multi-bound LGP tree search. We split the evaluation between scenes that have been solved with action sequence lengths of 2 or 3 (Figure 14a) and 4, 5, or 6 (Figure 14b). For the more challenging scenes requiring sequence lengths 4, 5, or 6, the runtime is between 2 and 3 orders of magnitudes smaller with the network. Note that the runtimes in Figure 14 for LGP are only reported for those where LGP was able to find a solution within the timeout. As indicated in Table 2, for the challenging scenes that require action sequence length 6, if the scene contains 3 objects, then in only 6% of the cases the network was able to find a solution within the timeout. If the scene contains 4 or 5 objects, then LGP without the network was not able to solve a single scene within the timeout. Therefore, the runtime results in Figure 13 are in favor of LGP. As mentioned, with our proposed approach, in all cases a solution was found.

These results emphasize that the capability of the network to reason about which objects are relevant to solve the task is crucial, even if only two objects have to be manipulated. Therefore, the network shows important generalization capabilities to multiple objects.

5.2.10. Generalization to more objects to be manipulated and longer sequence lengths. In the last section, we have shown that the network is able to generalize to more objects in the scene than it has been trained on. In those scenes, it was, however, sufficient to manipulate two objects to solve the task, as in the training distribution. As increasing the number of objects in the scene increases the number of candidate action sequences significantly, determining the relevant objects is, as we have shown, an important capability of the network to maintain high performance when generalizing to multiple objects.

Going beyond that, the question arises if the network is also useful in situations where more than two objects have

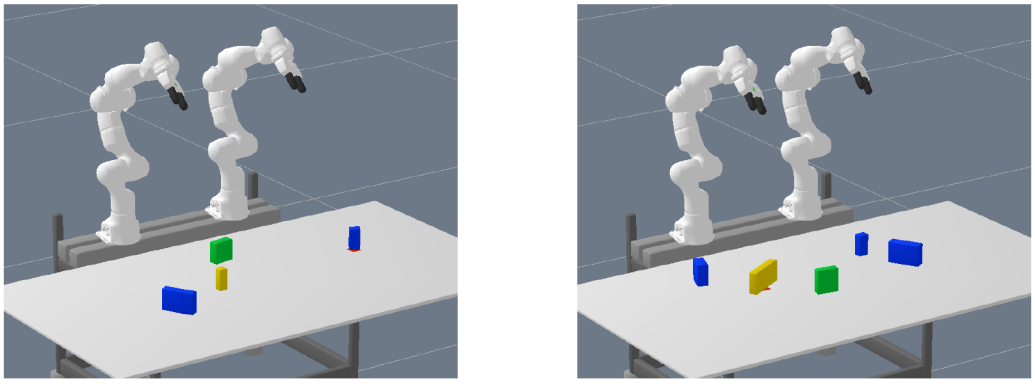


Figure 12. Example scenes for generalization to multiple objects (pick and place experiment). Object colors have no meaning.

Table 2. Comparison between network and multi-bound LGP tree search for generalization experiment to multiple objects in the scene. Percentage of solved scenes within the timeout depending on both the number of objects in the scene and the action sequence length to solve the task. LGP is not able to find a solution within the timeout for challenging scenes with many objects and long action sequence length. With the network, all scenes can be solved within the timeout.

	Number of objects in the scene	Length of the action sequence				
		2	3	4	5	6
Multi-bound LGP tree search without network	1	100%	100%	100%	100%	–
	2	100%	100%	100%	88%	27%
	3	100%	100%	100%	100%	6%
	4	100%	100%	100%	100%	0%
	5	100%	100%	100%	75%	0%
Deep visual reasoning π_{NN}	1					
	2					
	3			100%		
	4					
	5					

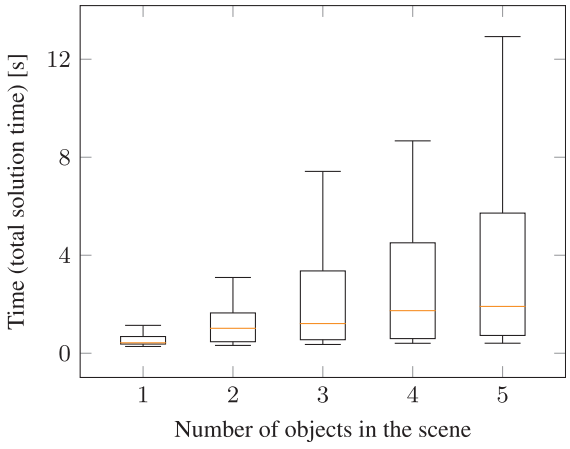


Figure 13. Generalization to multiple objects for pick-and-place experiment with the proposed deep visual reasoning neural network. During training, only and exactly two objects have been present in the training scenes. Although the number of candidate action sequences increases exponentially with more objects in the scene (see Table 1), the runtime to find a feasible solution with the neural network increases only slightly when more objects are added.

to be manipulated to solve the task. We do not expect the network to generalize to these settings with the same performance as in the other experiments.

In order to investigate this question, we generated five test scenes where not only the goal is obstructed by an object, but also the target object that should be placed on the goal cannot be grasped directly because another object is placed in a way that prevents grasping. See Figure 15a for an example scene. We call this experiment 1 in the following. This means that all three objects have to be manipulated to solve the task. Further, we generated three test scenes (see Figure 15b for an example) where, more challenging, the objects are placed in a way such that at least eight discrete actions are necessary due to the reachable set of the robot arms (and, hence, they have to collaborate). We call this experiment 2 in the following. These numbers of test scenes are not as statistically significant as the evaluations presented in the other parts of this work. This is due to the fact that randomly sampling scenes where three objects have to be manipulated is unlikely given our data-generation method. Therefore, we created these test scenes manually, without any bias to their solvability for the method.

Table 3 presents the runtimes for this generalization experiment for all scenes individually. We increased the timeout by a factor of 10 for this experiment compared with the others. Out of the five scenes for experiment 1, in four cases a solution was found using our network within the timeout. LGP was also able to find a solution for these four cases. Regarding experiment 2, in all three test scenes a solution was found with our network, whereas LGP was not able to find a solution for two of the three cases.

Although the network for these scenes cannot reduce the number of optimization problems that have to be solved as much as in the other experiments, in comparison with LGP, the speedup between 10 and 932 is still remarkable and the network makes it possible to solve some scenes at all. There are scenes where the task could be solved in less than 10 s with the network, but others can take of the order of hours. Therefore, the network acts as a heuristic to speed up the search, but does not eliminate it.

The column "π_{NN} with bound" is a slight variant of our proposed algorithm where the bound P_1 (cf. Section 3.2) in line 5 of Algorithm 1 is computed to check whether a selected node from the expand list is feasible according to bound P_1 (which checks mainly reachability of a single action). Including this check into the other experiments of this work where we evaluated the performance with the network did not make any statistically meaningful difference, hence we have not reported those results. However, in this case, where the network is a less perfect heuristic, having this check in the algorithm can prevent the search from

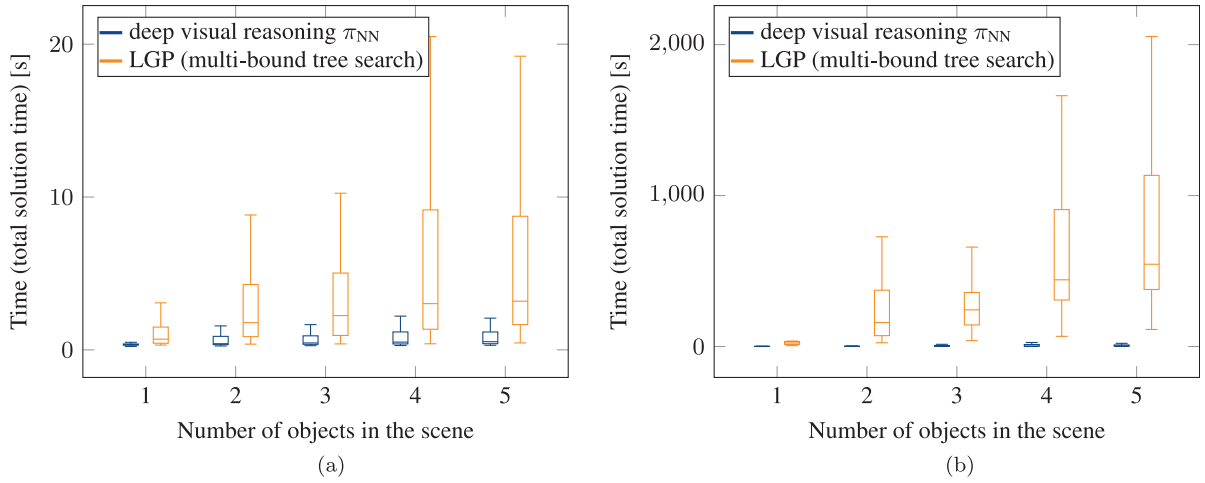


Figure 14. Comparison with multi-bound LGP tree search for the generalization to multiple objects experiment (pick-and-place scenario). Plots show the total runtime to find a feasible solution, split over scenes that could be solved with (a) an action sequence length of 2 or 3 and (b) an action sequence length 4, 5, or 6. The results show that the increased number of candidate action sequences for more objects in the scene (Table 1) leads to significantly longer solution times for LGP. In contrast, the runtime to find a feasible solution with the neural network increases only slightly when more objects are added.

Table 3. Runtime results for generalization experiment where three objects have to be manipulated to solve the task

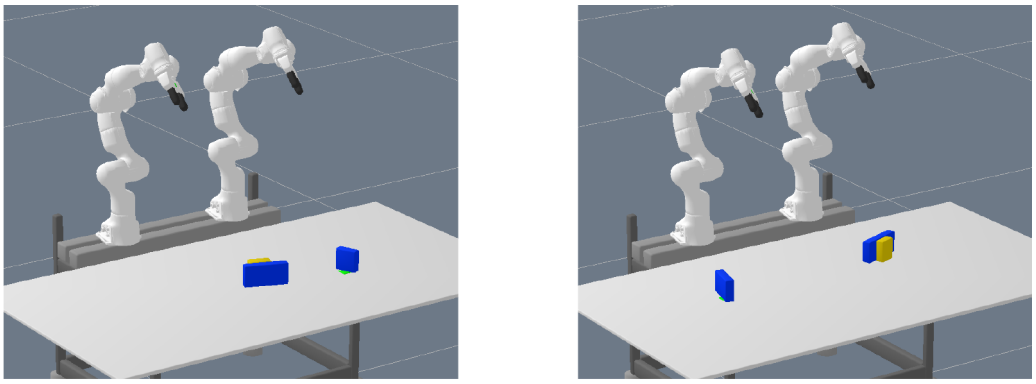
	Scene	π_{NN}	π_{NN} with bound	LGP	Speedup LGP/π_{NN}	Speedup LGP/π_{NN} with bound
Min seq. length 5 or 6	1	8 s	8 s	1,913 s	239	238
	2			— not found —		
	3	6 s	6 s	2,154 s	347	351
	4	179 s	19 s	17,750 s	99	932
	5	381 s	211 s	3,813 s	10	18
Min seq. length 8	6	14,121 s	3,326 s	not found	"∞"	"∞"
	7	10,041 s	2,384 s	not found	"∞"	"∞"
	8	114 s	93 s	1,182 s	10	13

going into infeasible parts of the LGP tree, leading to better performance.

5.2.11. Generalization to cylinders. Although the network has been trained on box-shaped objects only, we investigate whether the same network can generalize to scenes which contain other shapes such as cylinders. As the objects are encoded in the image space, there is a chance that, as compared with a feature space which depends on a less-general parameterization of the shape, this is possible. We generated 200 test scenes that either contain two cylinders, three cylinders or a mixture of a box and a cylinder, all of different sizes/positions/orientations and targets. If the goal is to place a cylinder on the target, we made sure in the data generation that the cylinder has an upper limit on its radius in order to be graspable. These cylinders, however, have a relatively similar appearance in the rasterized image as boxes. Therefore, the scenes also contain cylinders which have larger radii such that they have a clearly different appearance than what is contained in the dataset. An example of such a scene can be seen in Figure 16. These larger cylinders cannot, of course, be grasped and, hence, are not placed on the target or are the target itself. The network correctly does not attempt to grasp these cylinders with too large radii.

Figure 17a shows the total solution time with the neural network. As one can see, there is no drop in performance compared with box-shaped objects, which indicates that the network is able to generalize to other shapes. For sequence length 6, the runtimes are a bit higher, but are still very low, especially compared with LGP tree search without the network. In Figure 17b, the number of NLPs that have to be solved show that, except for sequence length 6, the median is 1 for all other sequence lengths, which means that in the majority of the cases, only a single optimization problem has to be solved to find a feasible solution.

Please note that our constraints for the nonlinear trajectory optimization problem are general enough to deal with boxes and cylinders. However, one also has to state that for even more general shapes the trajectory optimization for grasping becomes a problem in its own (grasping



(a) (b)

Figure 15. Generalization experiment where three objects have to be manipulated to solve the task, because not only is grasping the target object (yellow) obstructed by another object, but the goal location is also occupied with yet another object: (a) minimum required sequence length 5 or 6; (b) minimum required sequence length 8. The training distribution contained scenes with two objects only. Hence, only a maximum of two objects within an action sequence length of up to 6 have to be manipulated. Therefore, for the scene in (b), we ask for generalization in two ways beyond the training distribution, namely three objects to be manipulated and longer sequence length.

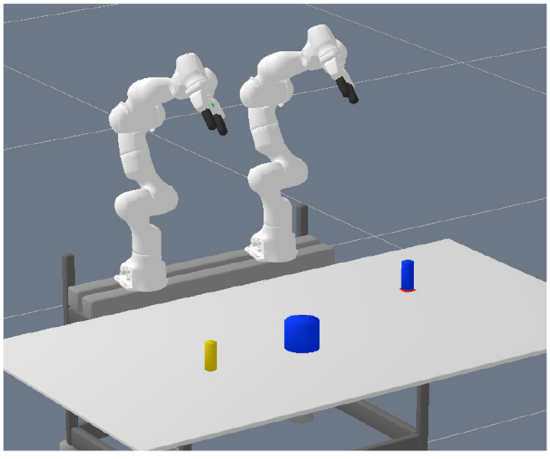


Figure 16. Test scene with cylinders. During training, the network has only seen box-shaped objects. The cylinder in the middle has a large enough radius to have a clearly different appearance in the image space than boxes. Pick-and-place experiment.

can be considered as an own subfield of robotics research).

Of the 200 test scenes, in 41% of the cases the network finds a solution with sequence length 2, in 9% of length 3, in 30% of length 4, in 3% of length 5, and in 17% of length 6, which also shows that we have not artificially created simple problems. As for these cylinders no handovers are possible, there are very few of sequence length 3 and 5.

5.2.12. Real robot experiments. Figure 4 shows our complete framework in the real world. In this scene the blue object occupies the goal location and the target object (yellow) is out of reach for the robot arm that is be able to place it on the goal. As the yellow object is large enough, the network proposed a handover solution (Figure 4e). The presence of an additional object (green) does not confuse the predictions. Indeed, in all real-world experiments, the network always predicted a feasible action sequence directly. The planned trajectories are executed open-loop, which implies that, although all planned trajectories are feasible, some executions fail. There were two main failure modes. On the one hand, if there was a handover, sometimes the grasp was not stable enough and the box rotated a bit, such that the other robot arm then could not successfully grasp the box. On the other hand, because there is no collision margin in the trajectory optimization method, sometimes a planned trajectory moves the gripper with a distance of only a few millimeters to a box. Small errors in the perception pipeline then could lead to unexpected movements of the boxes.

Note that the images as input to the neural network are rendered from object models obtained by a perception pipeline. This allows the trained network in simulation to be transferred to the real robot directly. The perception pipeline is a simple background subtraction method in the depth space to get object masks and then fitting box models to the segmented pointclouds. One could argue that the fact that we use rendered images from the object models for the real robot experiments is a limitation of the proposed approach compared with utilizing real images for the network predictions. However, because the trajectory optimization needs object models, there would be no advantage in the current approach to using real images for the network prediction.

5.3. Pushing experiment

In this experiment, we consider a scenario where objects on a table should be moved to a desired goal configuration

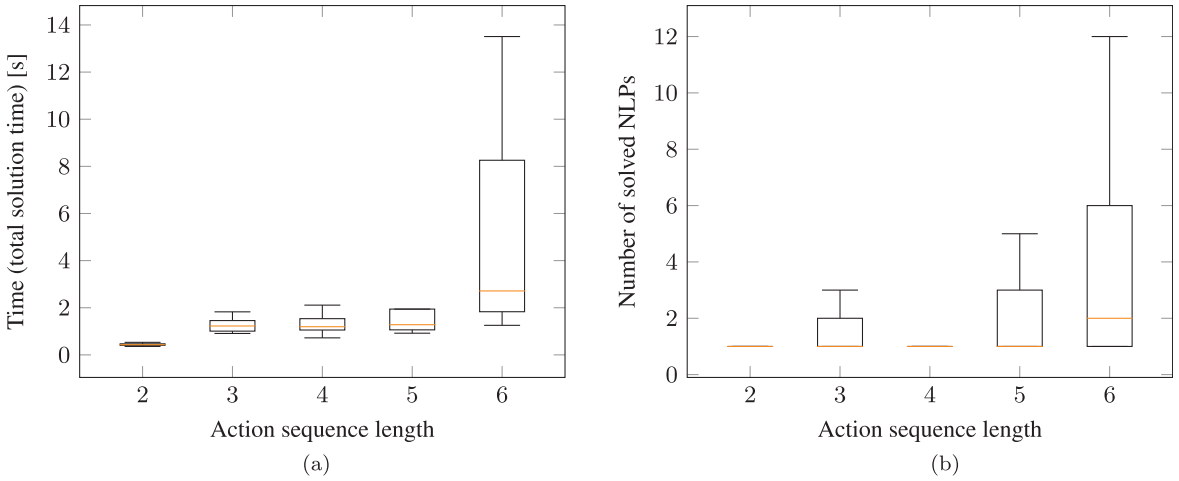


Figure 17. Generalization to cylinders for pick and place experiment with neural network integrated into the tree search algorithm: (a) total time to find a feasible solution; (b) number of NLPs that have to be solved to find a solution. During training, the network has never seen cylinder-shaped objects. Test scenarios include one, two cylinders, or a mixture of cylinder and a box. As can be seen in (b), except for sequence length 6, the median of the number of NLPs that have to be solved is one, meaning the first predicted action sequence is feasible in the majority of the cylinder test cases, i.e., no search necessary.

(position and orientation on the table) with one robot arm. Compared with the last experiment (Section 5.2), the objects are too large to be grasped. Furthermore, they can be completely out of reach of the robot. Therefore, the robot has to use a hook-shaped tool to push and/or pull the object to the desired location. See Figure 18a or 19 for a typical example scene. The goal pose in these scenes is visualized in transparent green.

5.3.1. Action operators and optimization objectives. The equations of motion of the object that should be pushed are taken from Toussaint et al. (2020), where it is modeled with quasi-static dynamics based on the Newton–Euler equations in the plane with friction. These equations of motion enter the optimization problem in terms of equality constraints.

We define three action operators, `grasp`, `pushSide(H` O ν`)`, and `toGoal(`O O_g`)`.

A `grasp` action enables the robot to grasp the hook by constraining the center point between the two grippers of the end-effector to be inside of the hook with an inequality constraint and a cost term that aligns the grippers parallel to the long side of the stick. The exact grasping location along the stick is not defined by the `grasp` action and, therefore, chosen by the optimizer. This action allows one degree of freedom more than the grasp action from Section 5.2.1 the optimizer has to fill in, because the hook has a cylindrical shape. As there is only one hook in this experiment, the `grasp` action does not explicitly have a parameter. Accounting for different hooks, however, would be a straightforward extension.

The second action operator `pushSide(H` O ν`)` models force exchange between the hook and a side of an object. The hook has two geometric features $\texttt{H} \in \{\texttt{H}_1, \texttt{H}_2\}$ that can be used for pushing, visualized as the two green balls in Figure 17. The parameter $O \in \mathcal{P}(\mathcal{O}(S))$ denotes the object that should be pushed. Following Toussaint et al. (2020), the `pushSide` action introduces multiple constraints and regularization objectives to the optimization problem. Most importantly, it introduces a six-dimensional decision variable $(f_{\text{PoA}}, p_{\text{PoA}}) \in \mathbb{R}^6$ to the path optimization problem that represents the total wrench exchange between the `H` part of the hook and the object O in terms of a linear force $f_{\text{PoA}} \in \mathbb{R}^3$ (constrained to be positive) and the so-called point of attack (PoA) $p_{\text{PoA}} \in \mathbb{R}^3$. Together, they define the wrench $(f_{\text{PoA}}, f_{\text{PoA}} \times p_{\text{PoA}})$ that enters the Newton–Euler equation for O. Please refer to Toussaint et al. (2020) for details about the PoA mechanism.

As an important extension to Toussaint et al. (2020), where the PoA is constrained to be *anywhere* on the surface of `H` and O at the same time for contact interactions to happen, we introduce the additional discrete parameter ν for the `pushSide` action. Let $\mathcal{N}(O) = \{\nu_1, \ldots, \nu_{n_\nu}\}$, $\nu_i \subset \mathbb{R}^3$ be a decomposition of the surface $\mathcal{N}(O)$ of O into $n_\nu < \infty$ many connected parts. In our case, we consider the vertical faces of prism objects (boxes and triangular prisms). Then the parameter ν of the `pushSide` action constraints the PoA to be on one of those faces, i.e., $p_{\text{PoA}} \in \nu$ as an equality constraint. This means that the choice of ν indicates from which side the object should be pushed.

Although the PoA mechanism of Toussaint et al. (2020) can, in principle, figure out from which side to push an object, we found that without introducing ν as a discrete decision, the resulting NLP is prone to local optima and the robustness of the optimizer finding a solution significantly drops due to the non-convexity of the problem, especially with respect to collision avoidance. The discrete variable ν

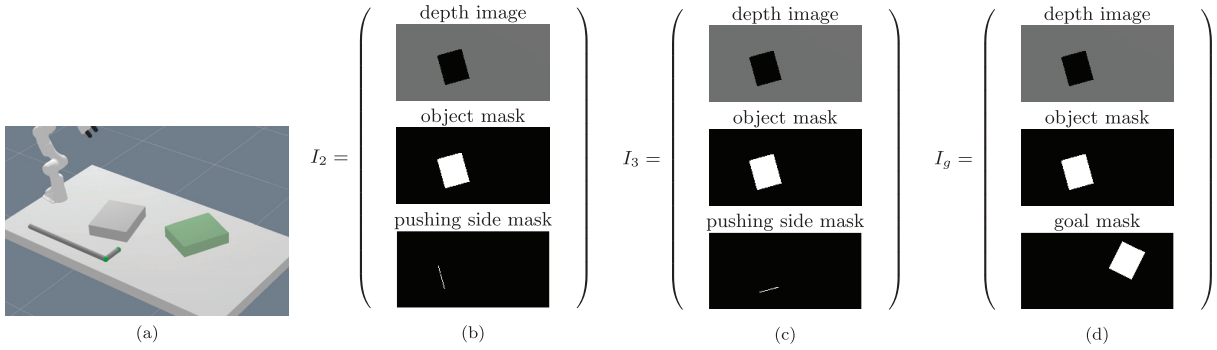


Figure 18. Visualization of the action–object image encodings $I(O, S)$ for an example test scene of the pushing experiment. (a) Initial scene. The two green balls of the hook are the two parts $\texttt{H}_1$ and $\texttt{H}_2$ of the hook that can be used for pushing. (b) Action–object image for the `pushSide` action utilizing $\texttt{H}_1$ of the hook for pushing. (c) Action–object image for the `pushSide` action utilizing $\texttt{H}_2$ (the tip of the hook) for pushing. (d) Goal–object and action–object image for `toGoal` action representing that the objects should be moved to the goal mask. In all images, the first channel is a depth channel of the scene. The images always refer to the initial scene configuration (a). The push side masks in (b) and (c) represent the side of the object as an edge mask where the robot should push.

singulates local optima and thereby greatly increases the robustness of the solver.

The PoA allows to model both sliding and sticking contacts. However, here we consider the sliding case only, which means that an equality constraint is added that ensures that the tangential component of the force on the surface of O chosen by ν is zero. A regularization term on the relative sliding velocity of the PoA penalizes the hook for sliding over the surface of the object, which increases the physical plausibility of the sliding contact assumption.

The two different parts $\{\texttt{H}_1, \texttt{H}_2\}$ of the hook that can be used for pushing are represented in $\bar{a}$, leading to two different discrete action symbols for the `pushSide` action. The object O that should be pushed as well as the pushing side ν is encoded in the action–object image. For O it is a mask of the object, for ν a mask of the surface, which for the top down view as in this experiment leads to pixels that represent an edge of the object. Figure 18 visualizes these action–object images.

Note that one could also represent those geometric features of the stick in the image space (then the `pushSide` action–object image would be a four-channel image), but this was not necessary for this experiment, because the hook was always the same.

Finally, the `toGoal(`O O_g`)` action simply models a pose equality constraint between the object O and its desired target pose O_g. Both O and O_g are encoded as masks in the action–object image.

We consider a maximum of two consecutive pushes. Therefore, in summary, if n_ν is the number of faces of an object, then there are $2n_\nu$ action sequences of length 3 and $(2n_\nu)^2$ action sequences of length 4. The first action is always a grasp of the hook.

5.3.2. Training/test data generation. We sampled 10,000 scenes containing boxes of different sizes, positions, and orientations, as well as 10,000 scenes with triangular prisms, again of different sizes, positions and orientations. Both are combined into one dataset, leading to 20,000 scenes in total, i.e., we train *one* network on both boxes and triangular prisms. For the boxes, the scene parameter space has dimension eight, and for the triangular prisms, the scene parameter space has dimension seven.

As, for this experiment, the LGP tree is relatively small (but still too large for satisfactory performance at test time), we can actually compute all solutions within reasonable time. Specifically, there are 72 action sequences for each box scene and 42 for each triangular prism scene. Eight of the 72 for the box are action sequences of length 3, and 64 are of length 4. For the triangular prisms, 6 are of length 3, and 36 are of length 4. Therefore, the dataset contains the feasibility of all possible action sequences of each sampled scene.

Representing the pushing actions in the image space allows us to have the same input encoding for both the triangular prisms and the boxes, although they have a different number of faces.

For the test data, we generated 1,000 scenes with the same method, but a different random seed, 500 with boxes and 500 with triangular prisms. For the boxes, 462 scenes contained at least one feasible solution, for the triangular prisms only 369, which then serve as the actual test scenes. In the test dataset, 14.4% of the action sequences for the boxes are feasible, 9% for the triangular prisms. Among those, 5.7% (9%) are of length 3 and 94.3% (91%) are of length 4 for the boxes (triangular prisms).

Note that for every feasible action sequence of length 3, there exists also a feasible one of length 4, namely by pushing two times from the same side. This pushing from the same side twice, however, can also be useful to find a cost-effective (or even feasible) solution, because, owing to the regularization cost term, sliding movements are penalized when the contact between the tool and the object is established. Two consecutive pushes from the same side allow the robot to reposition the contact location without creating large regularization costs.

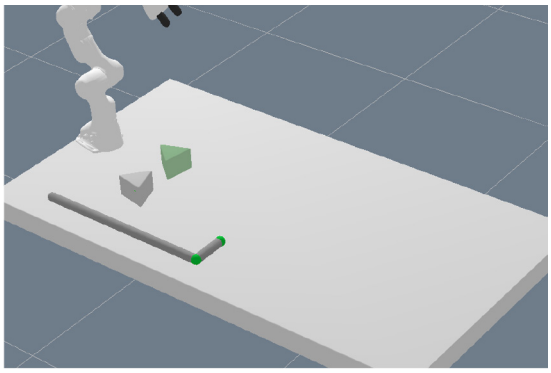


Figure 19. Example test scene of the pushing experiment containing a triangular prism as the object. The green prism visualizes the target location.

5.3.3. Performance: results on test scenarios. In Figure 20, we visualize a typical sequence of motions for the pushing scenario.

Figure 21 shows the number of NLPs that have to be solved to find a feasible solution with the neural network for the test scenarios. As one can see, for sequence length 3, for both boxes and triangular prisms the network always finds a feasible solution as its first prediction. For sequence length 4, the median of NLPs that have to be solved for boxes is one, for triangular prisms the performance is slightly worse with a median of 2, but still a high performance.

The network finds in 30% (23%) of the box (triangular prism) test cases a solution with sequence length 3 and in 70% (77%) of cases a solution with sequence length 4.

For this evaluation, the network has been trained on both boxes and triangular prisms. When we train separate networks for the two different shapes and then evaluate on only those different shapes separately, we do not see a significant difference in the performance.

5.3.4. Comparison with LGP tree search. Compared with kinematic problems as considered in Section 5.2, it is less clear how to define lower bounds that are useful for the pushing scenario that contains dynamic models. Therefore, we compare the performance with LGP tree search without lower bounds, i.e., where directly the full path optimization problem for a chosen action sequence is computed. When we report the number of NLPs that have to be solved to find a feasible solution with LGP in this section, we refer to the expected value when randomly selecting action sequences, which we can determine because, as mentioned in Section 5.3.2, the search tree is small enough for us being able to check the feasibility of all possible action sequences of the test scenes for comparison.

As can be seen in Figure 22, utilizing the network leads for both the triangular prisms and the boxes to a great speedup compared with randomly selecting action sequences. Note that in Figure 22b, LGP has an expected value of exactly 6 for sequence length 3, because for all triangular prisms in the test scenes only a single one of the 6 possible action sequences each was feasible.

5.3.5. Generalization to other shapes. The network is trained on box-shaped objects and triangular prisms. Our proposed image based representation for expressing the

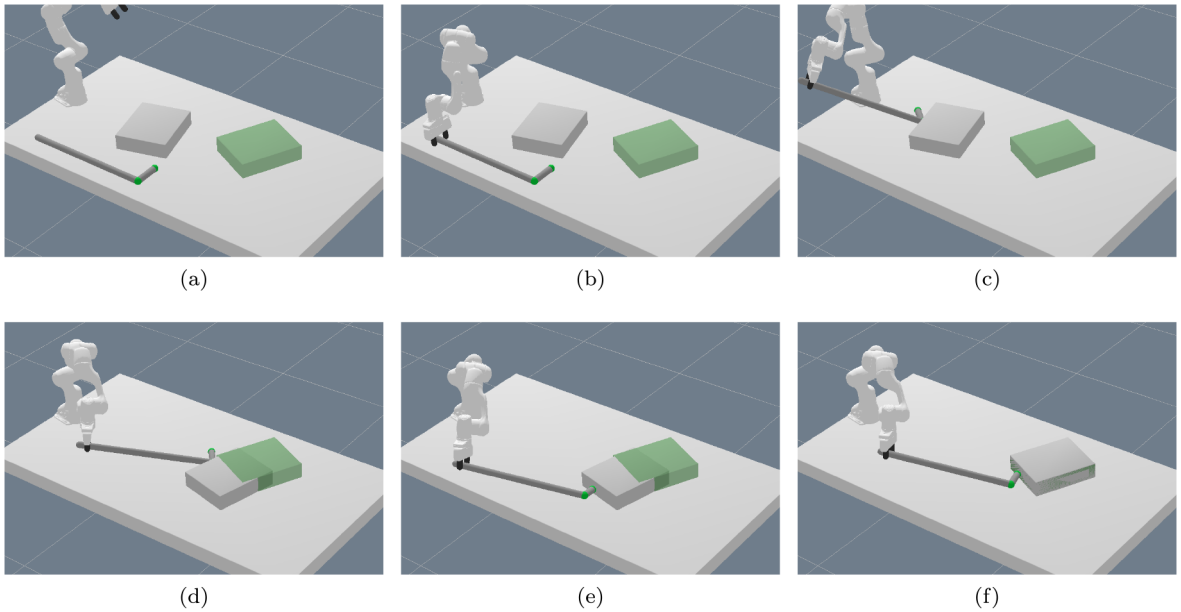


Figure 20. Typical sequence of motions for the pushing experiment: (a) initial scene; (b) `grasp` action; (c) `pushSide` action with H_1 begin; (d) pushing action ongoing; (e) `pushSide` action with H_2 begin; (f) `toSide` action to realize the goal. The gray box should be moved to the green goal location. The network directly predicts a feasible action sequence consisting of two pushes from different sides of the object with different parts (H_1 and H_2, indicated by the green balls) of the hook. Action sequence length is 4.

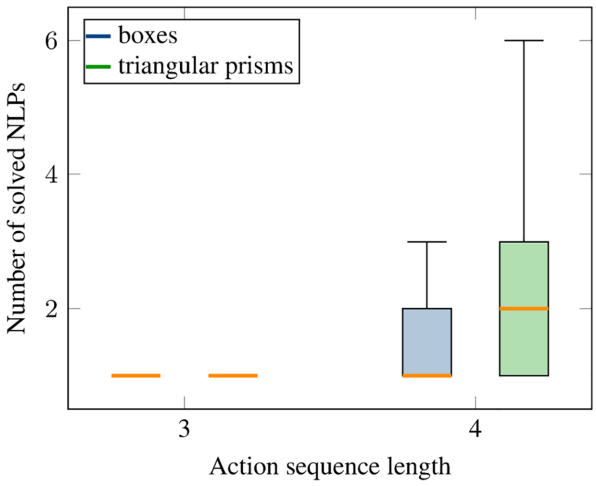


Figure 21. Performance on test scenes for the pushing experiment with our proposed neural network. The network was trained on both boxes and triangular prisms.

sides from which the object should be pushed and the desired goal pose allow in principle also other shapes as input to the neural network. Even if there are more push-faces, the algorithm can directly handle this. We therefore tested whether the network can generalize with good performance to other shapes than boxes and triangular prisms, in this case penta prisms.

In order to do so, we generated 200 test scenes that contain penta prisms of different sizes, positions, and orientations as well as different goal poses. See Figure 23 for an example test scene. For penta prisms, there are 110 different action sequences up to length 4 (10 of length 3, 100 of length 4). By solving the NLPs corresponding to all 110 different action sequences, we obtained 188 feasible scenes, which then are the test scenes for this experiment. A total of 13.7% of the action sequences are feasible, among which 3.5% are of length 3 and 96.5% of length 4.

Figure 24a reports the number of solved NLPs to find a feasible solution. This figure also contains a comparison with the expected value with LGP. As one can see, the network literally just generalizes to penta prisms, although during training the network only has seen boxes and triangular prisms. The performance is identical to the performance of solving scenes containing boxes (see Figure 21), which is remarkable. For sequence length 3, in all cases the first predicted sequence was feasible, for sequence length 4 the median was 1, the upper whisker 3. Further, in all of the 188 test scenes a solution was found. However, it also has to be noted that the optimizer has full access to the shape of the penta prism. Nevertheless, the network is capable of predicting the push sequence and from which side with high performance for penta prisms, which clearly are not contained in the dataset.

In addition, we tested whether a network that was trained only on boxes, i.e., no triangular prisms, can also generalize to penta prisms. As can be seen in Figure 24b, although the performance is slightly worse for sequence length 4 (median 2 compared with 1 and upper whisker 6 compared with 3), it still generalizes.

5.3.6. Completeness and feasibility threshold. In Table 4 we report the number of test scenes where the network could *not* find a solution for different feasibility threshold strategies. According to Proposition 1, when using the feasibility threshold together with the discounting adjustment (see Section 4.6.1), prediction errors of the network cannot prevent a feasible solution from being found if it exists. Indeed, as can be seen in the first row of Table 4, in all test scenarios, the adjustment mechanism of the threshold enables the network to not miss a single solution. This

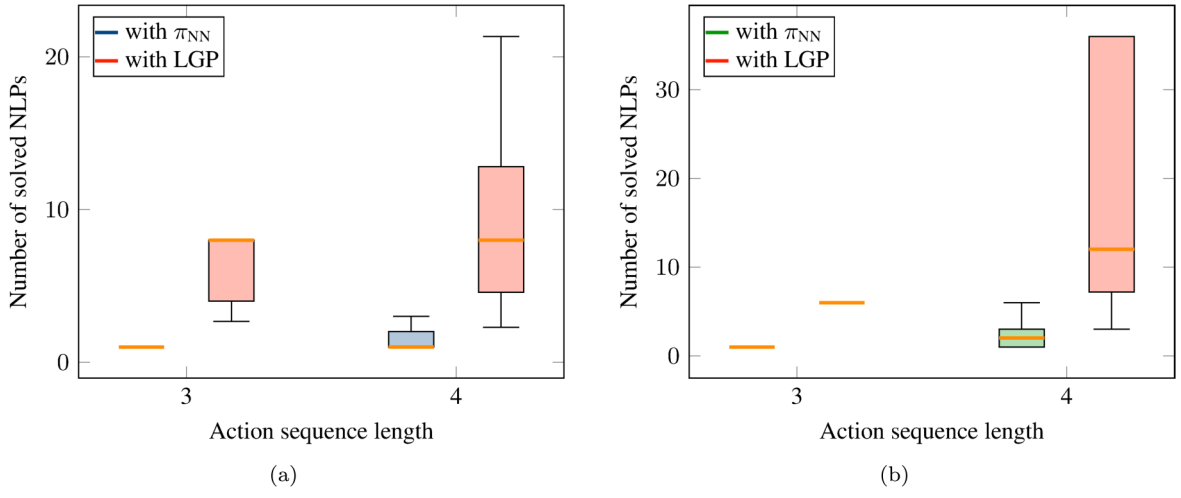


Figure 22. Comparison of LGP with our proposed neural network on test scenes for the pushing experiment: (a) comparison with LGP on box scenes; (b) comparison with LGP on triangular prisms scenes. The network was trained on both boxes and triangular prisms. In (b) the second median line from the left corresponds to LGP.

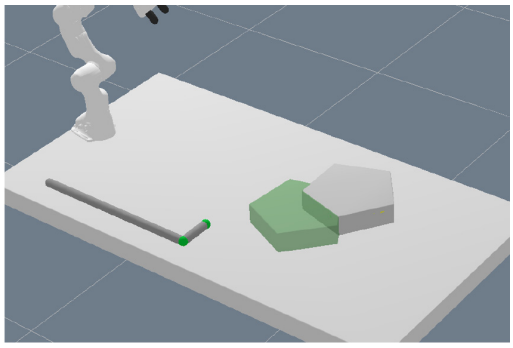


Figure 23. Generalization to other shapes. Example test scene of pushing experiment containing a penta prism. During training the network has only seen boxes and triangular prisms.

Table 4. Number of test scenes (pushing experiment) for which *no* solution was found with the network for different feasibility threshold strategies. The first row with $f_{\text{thresh}} = 0.5$ and $\gamma = 0.9$ is our proposed framework that has completeness guarantees and, hence, always finds a solution if it exists.

	Boxes	Triangular prisms	Penta prisms
$f_{\text{thresh}} = 0.5$ $\gamma = 0.9$	0	0	0
$f_{\text{thresh}} = 0.5$	20 (4%)	78 (21%)	28 (15%)
$f_{\text{thresh}} = 0.1$	8 (2%)	19 (5%)	5 (3%)
no f_{thresh}	0	0	0

holds true for testing on both boxes and triangular prisms as in the training set, but also for penta prisms which shape the network has never seen during training.

In comparison, if only the feasibility threshold $f_{\text{thresh}} = 0.5$ is used without the adjustment mechanism, for some (up to 21% for the triangular prism or 15% for the penta prism) test scenes, the network prevented a solution from being found by classifying feasible solutions as infeasible. Even when reducing the threshold to $f_{\text{thresh}} = 0.1$, still some feasible test scenes are classified infeasible and, hence, not solved. Only if the threshold is removed completely can all scenes again be solved. This, however, comes with a performance penalty.

5.3.7. Cost prediction. In Section 4.8, we proposed how the framework cannot only be used to find feasible solutions, but also to take the trajectory costs into account. This section analyzes both the performance of the cost prediction network to find a feasible solution and determines whether lower cost solutions can be found.

First, the performance in terms of the number of NLPs that have to be solved to find a feasible solution is exactly the same as with the feasibility network (the performance boxplot is identical to Figure 21 and therefore not shown again) and also for every feasible scene a solution was found.

In addition, utilizing the cost prediction network, in 78.6% of the test scenarios with boxes and 88.1% with triangular prisms, the first found solution had a lower cost than with the feasibility prediction network.

For the pick-and-place experiment of Section 5.2, it is intractable to compute all possible action sequences to find the cost optimal one. As, for this scenario, we can actually compute the costs for all sequences, we can investigate how close the first found solution utilizing the cost prediction network is to the real lowest cost over all possible action sequences. Figure 25(a) and (b) show this for the box and triangular prism test scenarios, respectively. These figures report the cost achieved with the network divided by the optimal cost. This means that a value of 1 corresponds to

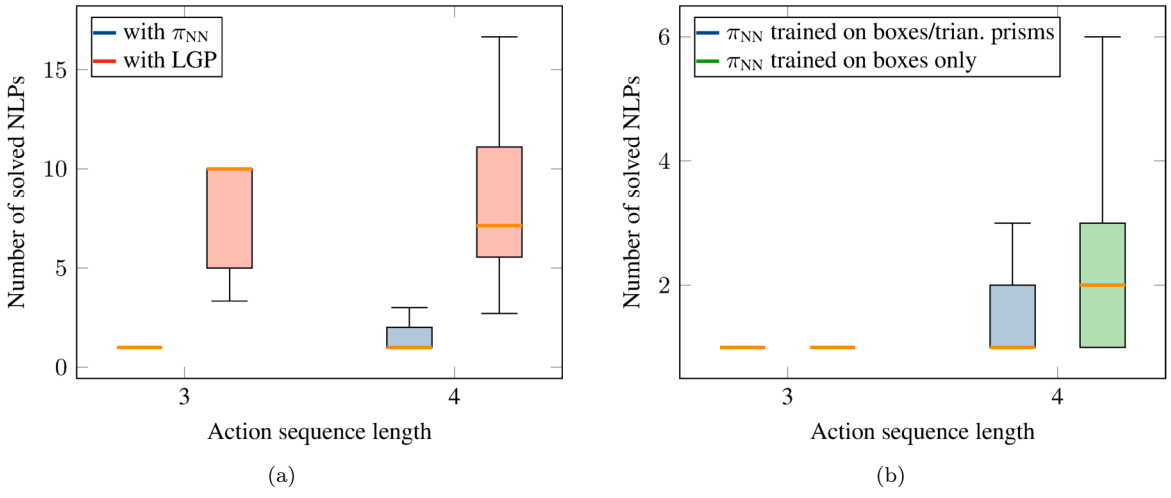


Figure 24. Generalization to penta prism shapes for the pushing experiment. (a) Performance with a network trained on boxes and triangular prisms as well as comparison with LGP without the network. (b) Performance of a network trained on boxes and triangular prisms or boxes only. During training of the network, only boxes and triangular prisms were present in the scene.

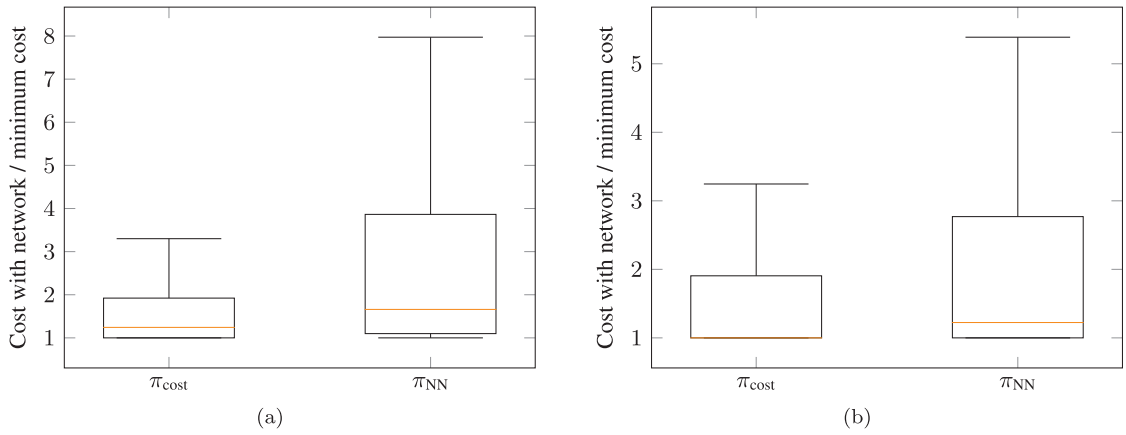


Figure 25. Achieved trajectory costs in comparison with the overall optimum for the first found feasible solution with the cost prediction network π_{cost} and with the feasibility prediction network π_{NN}: (a) evaluation on scenarios with boxes; (b) evaluation on scenarios with triangular prisms. A value of 1 corresponds to the optimal cost, a value of 2 to twice the optimal cost, etc.

the case where the cost optimal solution was found as the first prediction, a value of 2 corresponds to the case where the found cost was twice as high as the optimum, etc.

One can clearly see that by utilizing the cost prediction network one can achieve considerably lower costs with the first prediction compared with performing feasibility prediction only. The median for the boxes is very close to 1 and for the triangular prisms the median is 1, i.e., in half of the cases the first found solution was the cost optimal solution.

6. Discussion

Sequential manipulation problems as considered with TAMP approaches are difficult for several reasons. First, the sequential nature not only implies a huge combinatorial complexity of possible high-level action sequences, but also challenging non-convex motion planning problems, where multiple constraints at different phases of the motion have to be coordinated globally (Dantam et al., 2018; Driess et al., 2019a; Garrett et al., 2020; Orthey et al., 2020; Xu et al., 2020). For example, the hook has to be grasped in a certain way in order for it to be possible to push the object to an intermediate position from which the final push to the goal can then be executed.

LGP is one approach to address this by introducing discrete variables that are subject to logic rules to make the trajectory optimization for sequential manipulation problems more tractable, while retaining the property of being able to coordinate the motions in the different phases of the trajectory with global consistency. This, however, implies a combinatorial complexity, significantly increasing the computation time to find a solution, because many, mostly infeasible optimization problems have to be solved.

A key property of TAMP algorithms is that they show remarkable generalization capabilities with respect to different scenes with many and different numbers of objects, changing geometries, etc. This is another reason why sequential manipulation problems are difficult, because the variety of tasks in different scenes demands strongly generalizing algorithms. From a learning point of view, it is challenging to create datasets that cover such a variety of scene parameters and goals.

We make essential contributions in several of those regards. First, we address the combinatorial complexity that is introduced through the discrete variables by learning a goal conditioned network that guides the search over the discrete variables in a way that most of the time the search is eliminated, leading to large speedups in solution times. Further, we demonstrated our method not only on a single problem instance, but on a variety of scenes, including different object parameters, goals, but especially increasing numbers of objects, although only a fixed number of objects was present in the training set. Moreover, our proposed image representation enables generalization to different shapes of objects than those present in the training data.

The generalization to more objects than during training is a major advantage of the image-based encoding of the scene we proposed compared with a fixed feature representation. As shown in Section 5.2.9, the network is able to maintain high performance if more objects are added to the scene. However, one also has to state that this kind of generalization to multiple objects has to be understood in the sense that the network correctly chooses up to two objects which have to be manipulated (multiple times) to solve the task. The network realizes a kind of attention mechanism, operating on only the relevant parts of the environment. This could also be observed in the generalization experiment to cylinders (Section 5.2.11), where the network does not attempt to grasp cylinders which are too large, although it has never seen cylinder-shaped objects during training. Being able to identify which objects are relevant to solve the task is crucially important and the reason for why the network can find solutions quickly even in scenarios where there are nearly half a million candidate action sequences,

cf. Table 1 and Figure 13. We believe that such an attention mechanism is the key to tackling even more realistic environments with far greater combinatorial complexities such as household scenarios.

Although we do not expect the network to generalize with high accuracy to scenarios where more than two objects have to be manipulated in order to solve the task without a broader training distribution, we have investigated in Section 5.2.10 and Table 3 that for scenes where three objects have to be manipulated with action sequence length up to eight (training distribution has a maximum sequence length of six and contains only two objects), the network still leads to a speedup in finding a solution up to several orders of magnitude compared with LGP without the network. However, in this case, the network acts as a heuristic that does not eliminate search, since more optimization problems had to be solved to find a feasible solution compared with the scenarios where only two objects in the scene have to be manipulated.

Our approach assumes that we are able to extract segmentation masks of each object from the raw image of the initial scene. Many methods such as Mask R-CNN (He et al., 2017) have been developed for object segmentation. Although definitively challenging, we believe that the ability to segment objects in an image is a necessary condition for many robot applications that rely on perception in uncontrolled environments and, hence, a reasonable assumption to make. In particular, without being able to detect objects, defining the TAMP problem in the first place is unclear. There are also recent approaches that estimate the state of the symbolic domain from image segmentations (Kase et al., 2020; Mukherjee et al., 2020; Zhu et al., 2020).

As already mentioned in Section 5.2.12, although the generation of the discrete action sequences only relies on images and these object segmentations, the trajectory optimization still requires object models in terms of their shapes and poses. Hence, a perception pipeline is required to extract these properties from the raw image observations. Future work could investigate how to make the motion generation given a discrete action sequence less reliant on exact object models (Driess et al., 2021a,b; Manuelli et al., 2019; Qin et al., 2019; Simeonov et al., 2020; Suh and Tedrake, 2020; Wang et al., 2020).

Despite this, we have shown that the image representation has advantages (generalization to multiple objects/shapes, encoding geometry information) independently from whether object models are known or not. This means that one way to interpret the images in this work is not solving manipulation from raw perception, but a flexible state/action representation for long-horizon reasoning. Still, we believe that this representation is also one step to connect TAMP more closely to real perception.

One of the main limitations of this work, in our opinion, is that the initial scene image has to contain all information that is necessary to reason about the action sequence.

Therefore, we have focused on tabletop manipulation scenarios, because, in this case, the assumption that the image contains sufficient information about the scene/task is reasonable for the considered tasks. Total occlusions, for example, if an object is inside a cabinet, cannot be handled with the current approach. Partial occlusions, as long as the image still contains sufficient information, can in principle work. In a scenario where objects are stacked on top of each other in the initial scene, the top-down camera view as considered here is insufficient, but could be replaced with an angled view. As long as object masks and the image are provided in such a way that still the relevant object geometries can be inferred, we expect the method to work just as well. In mobile manipulation setups, the proposed methodology would not be applicable for solving the whole problem, because it is unrealistic to assume that a single image contains all necessary information. However, we believe that the ideas of this work could also be useful for solving parts of the manipulation problem (subgoals) quickly and repeatedly in an online setup. Most existing TAMP approaches assume *full* knowledge about the initial state, even in mobile manipulation settings. Only recent work starts to address these issues by belief-space planning in the context of TAMP (Garrett et al., 2019; Phiquepal and Toussaint, 2019).

In general, a major strength of the LGP formulation is that the actions imply constraints that only partially determine the behavior of the trajectory. The overall trajectory is then optimized with global consistency, filling in the remaining degrees of freedoms, or action parameters as they are called in other TAMP approaches. This property of LGP allows us, for example, to efficiently generate handover motions or removing of obstacles without explicitly needing to enumerate handover poses or the placement positions of the occupying obstacle. However, scaling to significantly longer action sequence lengths, it becomes clear that it is neither necessary nor feasible to optimize trajectories with complete global consistency. Introducing further hierarchies (Kaelbling and Lozano-Pérez, 2011) or breaking the manipulation down into (less-dependent) subgoals (Driess et al., 2019a; Hartmann et al., 2020, 2021) becomes necessary.

Instead of a black-box reward function, we specify the manipulation planning goal in terms of object masks similar to the action–object images. This goal–object image contains both the mask of the target object and the goal, as well as a channel of the whole scene. Therefore, the goal specification can encode the target object, the goal and other objects in the scene in such a way to take their geometric relations into account. Figure 26 shows a scenario where the goal is only partially occluded by the blue object. In this case, the network is able to realize that the yellow object can be placed on the green goal region without removing the blue object first and proposes this as the first solution. For a stacking scenario, one could think of having multiple object masks as part of the goal specification to encode which objects should be stacked. However, such

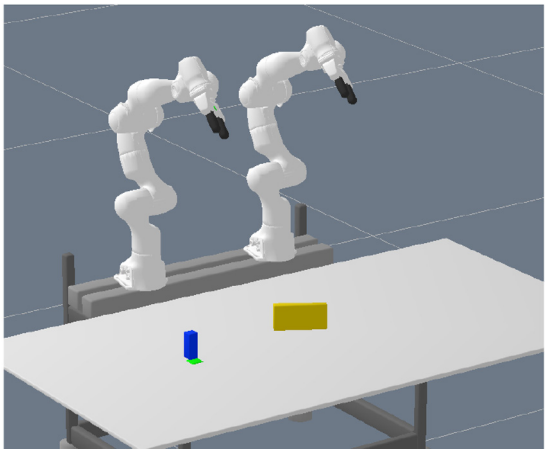


Figure 26. Goal region (green) is only partially occluded by the blue object, such that the yellow object can be placed on the goal without removing the blue object first. The network find this as its first proposed solution.

goal specifications assume that masks of the *unobstructed* goal region for the pick-and-place experiment and the mask of the desired target pose for the pushing experiment can be generated. Investigating other goal specifications is an important topic for future research.

To show the completeness of our framework (proposition 1), we assumed (assumption 1) that the nonlinear trajectory optimizer numerically converges to a feasible solution if the problem is indeed theoretically feasible. Although such assumptions usually only hold for convex problems and our trajectory optimization problems are, even for a fixed action sequence, non-convex, we empirically found that the optimizer converges reliably. The introduction of the additional discrete decisions for the pushing scenario was important for this robustness.

We considered boxes, cylinders, triangular, and penta prisms, all of different sizes, in this work. Although we argue that an advantage of the input images is their principal capability of representing and generalizing to different shapes, we assumed that the underlying trajectory optimization method can handle those shapes. In order to achieve this, the paradigm of this work is to introduce a set of discrete decisions for each subtask (grasping, pushing, and placing). These decisions can (not strictly) be understood as a means to enumerate local optima such that, given the discrete decisions, the optimizer is then able to find the remaining degrees of freedom for global consistency robustly. In the case of grasping, those decisions enumerate graspings from different sides of the object. For pushing, they specify the side from which the object should be pushed. In principle, one could think of introducing more such discrete decisions to deal with a greater variety of shapes. However, generalizing this idea to arbitrary shapes is not directly straightforward for multiple reasons. On the one hand, it becomes less clear how to define constraints realizing the subtasks for arbitrary shapes in a way that, in the paradigm of this work, benefits from parameterizing them with discrete decisions to make trajectory optimization with those constraints tractable. On the other hand, if too many discrete decisions are introduced, then the branching factor of the LGP tree increases significantly. This might not only lead to a harder learning problem, but it also becomes more challenging to generate data containing enough feasible action sequences in a reasonable amount of computation time. As illustrated in Figure 6, we considered four different graspings from above the object. In our previous work (Driess et al., 2020b), we have shown for a single action that the feasibility of not only top, but additionally also side grasps can be predicted from a similar input representation as in the present work. We anticipate that our network would, in principle, be able to take side grasps into account as well. In order to address the data generation problem in this case due to an increased branching factor, bootstrapping the learned network for data generation would be one way. We believe that integrating grasping of complex objects within long-horizon tasks where the way an early grasp is executed has to be coupled with later phases of the motion is an important future research topic.

7. Conclusion

In this work, we have proposed a neural network that learns to predict promising discrete action sequences for sequential manipulation problems from an initial scene image and the task goal as input. In most cases, the first sequence generated by the network was feasible. Hence, despite the fact that the network can act as a search heuristic, there was very little search over the discrete decisions required and, consequently, often only one trajectory optimization problem had to be solved to find a solution to the TAMP problem.

Although being trained on only two objects present at a time, the learned representation of the network was able to be generalized to scenes with multiple objects and other shapes while still showing a high performance.

We have shown that the approach can be applied not only to kinematic pick-and-place problems as is typical in TAMP, but also to a scenario where an object has to be pushed with a tool to a desired target location. Here one can see another advantage of the image representation, because the network generalized to other shapes than during training.

A main assumption and, therefore, main limitation of the proposed method is that the initial scene image has to contain sufficient information to solve the task, which means that there should be no total occlusions or other ambiguities.

Acknowledgements

We thank the anonymous reviewers for their helpful comments.

ORCID iD

Danny Driess https://orcid.org/0000-0002-8258-1659

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship and/or publication of this article: Danny Driess thanks the International Max-Planck Research School for Intelligent Systems (IMPRS-IS) for support. Marc Toussaint is grateful for the Max-Planck Fellowship at the Max-Planck Institute for Intelligent Systems in Stuttgart. This research has been supported by the German Research Foundation (DFG) under Germany's Excellence Strategy (grant number EXC 2002/1–390523135 "Science of Intelligence").

References

Amos B, Jimenez I, Sacks J, Boots B and Kolter JZ (2018) Differentiable MPC for end-to-end planning and control. In: *Advances in Neural Information Processing Systems*, pp. 8289–8300.

Bejjani W, Dogar M and Leonetti M (2019) Learning physics-based manipulation in clutter: Combining image-based generalization and look-ahead planning. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Boots B, Siddiqi SM and Gordon GJ (2011) Closing the learning–planning loop with predictive state representations. *The International Journal of Robotics Research* 30(7): 954–966.

Carpentier J, Budhiraja R and Mansard N (2017) Learning feasibility constraints for multicontact locomotion of legged robots. In: *Robotics: Science and Systems*.

Chitnis R, Hadfield-Menell D, Gupta A, et al. (2016) Guided search for task and motion plans using learned heuristics. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 447–454.

Chitnis R, Silver T, Kim B, Kaelbling LP and Lozano-Perez T (2020) CAMPS: Learning context-specific abstractions for efficient planning in factored MDPs. *arXiv preprint arXiv:2007.13202*.

Dantam NT, Kingston ZK, Chaudhuri S and Kavraki LE (2018) An incremental constraint-based framework for task and motion planning. *The International Journal on Robotics Research* 37(10): 1134–1151.

de Silva L, Pandey AK, Gharbi M and Alami R (2013) Towards combining HTN planning and geometric task planning. *arXiv preprint arXiv:1307.1482*.

Doshi N, Hogan FR and Rodriguez A (2020) Hybrid differential dynamic programming for planar manipulation primitive. In: *International Conference on Robotics and Automation (ICRA)*. IEEE.

Dosovitskiy A and Koltun V (2017) Learning to act by predicting the future. In: *International Conference on Learning Representations ICLR*.

Driess D, Ha JS and Toussaint M (2020a) Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. In: *Proceedings of Robotics: Science and Systems (R:SS)*.

Driess D, Ha JS, Tedrake R and Toussaint M (2021a) Learning geometric reasoning and control for long-horizon tasks from visual input. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Driess D, Ha JS, Toussaint M and Tedrake R (2021b) Learning models as functionals of signed-distance fields for manipulation planning. In: *Proceedings of the Annual Conference on Robot Learning (CoRL)*.

Driess D, Oguz O and Toussaint M (2019a) Hierarchical task and motion planning using logic-geometric programming (HLGP). In: *RSS Workshop on Robust Task and Motion Planning*.

Driess D, Oguz O, Ha JS and Toussaint M (2020b) Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Driess D, Schmitt S and Toussaint M (2019b) Active inverse model learning with error and reachable set estimates. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*.

Ebert F, Finn C, Lee AX and Levine S (2017) Self-supervised visual planning with temporal skip connections. In: *Conference on Robot Learning*.

Finn C and Levine S (2017) Deep visual foresight for planning robot motion. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2786–2793.

Finn C, Tan XY, Duan Y, Darrell T, Levine S and Abbeel P (2016) Deep spatial autoencoders for visuomotor learning. In: *International Conference on Robotics and Automation (ICRA)*. IEEE.

Garrett C, Kaelbling L and Lozano-Perez T (2016) Learning to rank for synthesizing planning heuristics. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Garrett CR, Chitnis R, Holladay R, et al. (2020) Integrated task and motion planning. *arXiv preprint arXiv:2010.01083*.

Garrett CR, Paxton C, Lozano-Pérez T, Kaelbling LP and Fox D (2019) Online replanning in belief space for partially observable task and motion problems. *arXiv preprint arXiv:1911.04577*.

Ha JS, Driess D and Toussaint M (2020) Probabilistic framework for constrained manipulations and task and motion planning under uncertainty. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Ha JS, Park YJ, Chae HJ, Park SS and Choi HL (2018) Adaptive path-integral autoencoders: Representation learning and planning for dynamical systems. In: *Advances in Neural Information Processing Systems*, pp. 8927–8938.

Hartmann VN, Oguz OS, Driess D, Toussaint M and Menges A (2020) Robust task and motion planning for long-horizon architectural construction planning. *arXiv preprint arXiv:2003.07754*.

Hartmann VN, Orthey A, Driess D, Oguz OS and Toussaint M (2021) Long-horizon multi-robot rearrangement planning for construction assembly. *arXiv preprint arXiv:2106.02489*.

He K, Gkioxari G, Dollár P and Girshick R (2017) Mask R-CNN. In: *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969.

Hogan FR, Grau ER and Rodriguez A (2018) Reactive planar manipulation with convex hybrid MPC. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 247–253.

Hogan FR and Rodriguez A (2016) Feedback control of the pusher–slider system: A story of hybrid and underactuated contact dynamics. In: *Proceedings of the Workshop on Algorithmic Foundation Robotics (WAFR)*.

Ichter B, Harrison J and Pavone M (2018) Learning sampling distributions for robot motion planning. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 7087–7094.

Ichter B and Pavone M (2019) Robot motion planning in learned latent spaces. *Robotics and Automation Letters* 4(3): 2407–2414.

Kaelbling LP and Lozano-Pérez T (2011) Hierarchical planning in the now. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Kase K, Paxton C, Mazhar H, Ogata T and Fox D (2020) Transferable task execution from pixels through deep planning domain learning. *arXiv preprint arXiv:2003.03726*.

Kim B, Kaelbling LP and Lozano-Pérez T (2018) Guiding search in continuous state-action spaces by learning an action sampler from off-target search experience. In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Kim B, Wang Z, Kaelbling LP and Lozano-Pérez T (2019) Learning to guide task and motion planning using score-space representation. *The International Journal of Robotics Research* 38(7): 793–812.

Kloss A, Bauza M, Wu J, Tenenbaum JB, Rodriguez A and Bohg J (2020) Accurate vision-based manipulation through contact reasoning. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6738–6744.

Lagriffoul F, Dimitrov D, Bidot J, Saffiotti A and Karlsson L (2014) Efficiently combining task and motion planning using geometric constraints. *The International Journal on Robotics Research* 33(14): 1726–1747.

Lagriffoul F, Dimitrov D, Saffiotti A and Karlsson L (2012) Constraint propagation on interval bounds for dealing with geometric backtracking. In: *International Conference on Intelligent Robots and Systems*.

Lang T and Toussaint M (2009) Relevance grounding for planning in relational domains. In: *Proceedings of the European Conference on Machine Learning (ECML 2009)* (*Lecture Notes in Computer Science*, Vol. 5781). New York: Springer, pp. 736–751.

Lange S, Riedmiller MA and Voigtländer A (2012) Autonomous reinforcement learning on raw visual input data in a real world application. In: *Proceedings of IJCNN*.

Li R, Jabri A, Darrell T and Agrawal P (2020) Towards practical multi-object manipulation using relational reinforcement learning. In: *International Conference on Robotics and Automation (ICRA)*. IEEE.

Lozano-Pérez T and Kaelbling LP (2014) A constraint-based method for solving sequential manipulation planning problems. In: *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*.

Manuelli L, Gao W, Florence P and Tedrake R (2019) KPAM: Keypoint affordances for category-level robotic manipulation. *arXiv preprint arXiv:1903.06684*.

Mason M (1985) The mechanics of manipulation. In: *International Conference on Robotics and Automation (ICRA'85)*. IEEE.

Mukherjee S, Paxton C, Mousavian A, Fishman A, Likhachev M and Fox D (2020) Sim-to-real task planning and execution from perception via reactivity and recovery. *arXiv preprint arXiv:2011.08694*.

Okada M, Rigazio L and Aoshima T (2017) Path integral networks: End-to-end differentiable optimal control. *arXiv preprint arXiv:1706.09597*.

Orthey A, Akbar S and Toussaint M (2020) Multilevel motion planning: A fiber bundle formulation. *arXiv preprint arXiv:2007.09435*.

Pascanu R, Li Y, Vinyals O, et al. (2017) Learning model-based planning from scratch. *CoRR* abs/1707.06170.

Paxton C, Barnoy Y, Katyal KD, Arora R and Hager GD (2019) Visual robot task planning. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 8832–8838.

Phiquepal C and Toussaint M (2019) Combined task and motion planning under partial observability: An optimization-based approach. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Qin Z, Fang K, Zhu Y, Fei-Fei L and Savarese S (2019) Keto: Learning keypoint representations for tool manipulation. *arXiv preprint arXiv:1910.11977*.

Racanière S, Weber T, Reichert D, et al. (2017) Imagination-augmented agents for deep reinforcement learning. In: *Advances in Neural Information Processing Systems*.

Rodriguez I, Nottensteiner K, Leidner D, Kasecker M, Stulp F and Albu-Schäffer A (2019) Iteratively refined feasibility checks in robotic assembly sequence planning. *Robotics and Automation Letters* 4(2): 1416–1423.

Silver D, van Hasselt H, Hessel M, et al. (2017) The predictron: End-to-end learning and planning. In: *International Conference on Machine Learning*.

Silver T, Chitnis R, Curtis A, Tenenbaum J, Lozano-Perez T and Kaelbling LP (2020) Planning with learned object importance in large problem instances using graph neural networks. *arXiv preprint arXiv:2009.05613*.

Simeonov A, Du Y, Kim B, et al. (2020) A long horizon planning framework for manipulating rigid pointcloud objects. *arXiv preprint arXiv:2011.08177*.

Srinivas A, Jabri A, Abbeel P, Levine S and Finn C (2018) Universal planning networks: Learning generalizable representations for visuomotor control. In: *International Conference on Machine Learning (ICML)*, pp. 4739–4748.

Srivastava S, Fang E, Riano L, Chitnis R, Russell SJ and Abbeel P (2014) Combined task and motion planning through an extensible planner-independent interface layer. In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*.

Suh H and Tedrake R (2020) The surprising effectiveness of linear models for visual foresight in object pile manipulation. *arXiv preprint arXiv:2002.09093*.

Tamar A, Wu Y, Thomas G, Levine S and Abbeel P (2016) Value iteration networks. In: *Advances in Neural Information Processing Systems*, pp. 2154–2162.

Toussaint M (2015) Logic-geometric programming: An optimization-based approach to combined task and motion planning. In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI Press, pp. 1930–1936.

Toussaint M, Allen KR, Smith KA and Tenenbaum JB (2018) Differentiable physics and stable modes for tool-use and manipulation planning. In: *Proceedings of Robotics: Science and Systems (R:SS)*.

Toussaint M, Ha JS and Driess D (2020) Describing physics for physical reasoning: Force-based sequential manipulation planning. *arXiv preprint arXiv:2002.12780*.

Toussaint M and Lopes M (2017) Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4044–4051.

Wang J, Lin S, Hu C, Zhu Y and Zhu L (2020) Learning semantic keypoint representations for door opening manipulation. *IEEE Robotics and Automation Letters* 5(4): 6980–6987.

Wang Z, Garrett CR, Kaelbling LP and Lozano-Pérez T (2018) Active model learning and diverse action sampling for task and motion planning. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4107–4114.

Watter M, Springenberg JT, Boedecker J and Riedmiller MA (2015) Embed to control: A locally linear latent dynamics model for control from raw images. In: *Advances in Neural Information Processing Systems*, pp. 2746–2754.

Wells AM, Dantam NT, Shrivastava A and Kavraki LE (2019) Learning feasibility for task and motion planning in tabletop environments. *Robotics and Automation Letters* 4(2): 1255–1262.

Wilson M and Hermans T (2019) Learning to manipulate object collections using grounded state representations. In: *Conference on Robot Learning*.

Xie A, Ebert F, Levine S and Finn C (2019) Improvisation through physical understanding: Using novel objects as tools with visual foresight. In: *Proceedings of Robotics: Science and Systems (R:SS)*.

Xu D, Mandlekar A, Martn-Martn R, Zhu Y, Savarese S and Fei-Fei L (2020) Deep affordance foresight: Planning through what can be done in the future. *arXiv preprint arXiv:2011.08424*.

Zhu Y, Tremblay J, Birchfield S and Zhu Y (2020) Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. *arXiv preprint arXiv:2012.07277*.

Appendix. Index to multimedia extensions

Archives of IJRR multimedia extensions published prior to 2014 can be found at http://www.ijrr.org, after 2014 all videos are available on the IJRR YouTube channel at http://www.youtube.com/user/ijrrmultimedia

Table of Multimedia Extensions

Extension	Media type	Description
	Video	Demonstration of the planned motions both in simulation and with a real robot