Deep Visual Reasoning: Learning to Predict Action Sequences for Task and Motion Planning from an Initial Scene Image

Danny Driess   Jung-Su Ha   Marc Toussaint

Machine Learning and Robotics Lab, University of Stuttgart, Germany
Max-Planck Institute for Intelligent Systems, Stuttgart, Germany
Learning and Intelligent Systems Group, TU Berlin, Germany

Abstract—In this paper, we propose a deep convolutional recurrent neural network that predicts action sequences for task and motion planning (TAMP) from an initial scene image. 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. Due 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 develop a neural network which, based on an initial image of the scene, 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. A key aspect is that our method generalizes to scenes with many and varying number of objects, although being trained on only two objects at a time. This is possible by encoding the objects of the scene in images as input to the neural network, instead of a fixed feature vector. Results show runtime improvements of several magnitudes. Video: https://youtu.be/i8yyEbbvoEk

I. INTRODUCTION

A major challenge of 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, for example nonlinear trajectory optimization or a sampling-based method, tries to find motions that fulfill the requirements induced by the high level action sequence or return that the sequence is infeasible.

Due to the high combinatorial complexity of possible discrete action sequences, a large number of motion planning problems have to be solved to find a solution to the TAMP problem. This is mainly caused by the fact that many TAMP problems are difficult, since the majority of action sequences are actually infeasible, mostly due to kinematic limits or geometric constraints. Moreover, it 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 it exists. 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 heuristic on the symbolic level, we can drastically reduce the number of motion planning
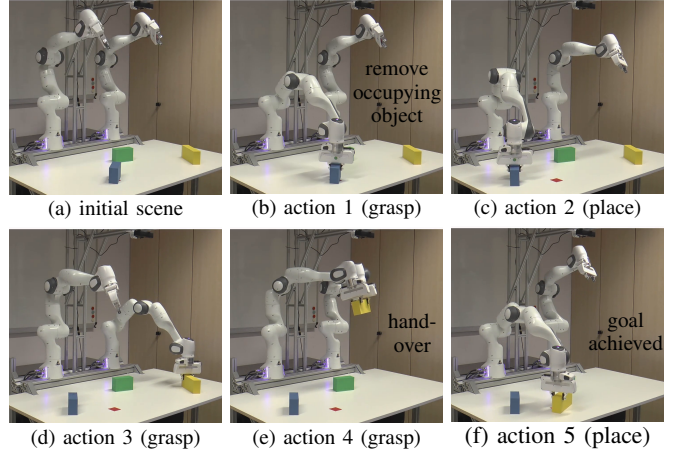

Fig. 1. Typical scene: The yellow object should be placed on the red spot, which is, however, occupied by the blue object. Furthermore, the yellow object cannot be reached by the robot arm that is able to place it on the red spot.

problems that need to be evaluated. Ideally, we seek to directly predict a *feasible* action sequence, requiring only a *single* motion planning problem to be solved.

However, learning to predict such action sequences imposes multiple challenges. First of all, 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 number of objects and goals. Secondly, the large variety of such scenes and goals, especially if multiple objects are involved, makes it difficult to generate a sufficient dataset.

Recently, [48] and [12] propose 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 develop a neural network that predicts action *sequences* from the initial scene *and* the goal as input. An important question is how the objects in the scene can be encoded as input to the predictor in a way that shows similar generalization capabilities of classical TAMP approaches. 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 number of objects with only little runtime increase, although it has been trained

on only a fixed number of objects. Compared to a purely discriminative model, since the predictions of our network are goal-conditioned, we do not need to use the network to search over many sequences, but can directly generate them with it.

The predicted action sequences parameterize a nonlinear trajectory optimization problem that optimizes a globally consistent paths fulfilling the requirements induced by the actions. To summarize, our main contributions are

- A convolutional, recurrent neural network that predicts from an initial 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.

From a methodological point of view, this work contains nonlinear trajectory optimization, first-order logic reasoning and deep convolutional, recurrent neural networks.

II. RELATED WORK

A. Learning to Plan

There is great interest in learning to mimic planning itself. The architectures in [41, 33, 39, 1] resemble value iteration, path integral control, gradient-based trajectory optimization and iterative LQR methods, respectively. For sampling-based motion planning, [23] learn 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 reinforcement learning (RL) in the learned latent space [3, 50, 17, 22, 47, 15, 29, 38]. Another line of research is to learn an action-conditioned predictive model [14, 50, 35, 13, 9, 34, 36]. With this model, the future state of the environment for example in image space conditioned on the action is predicted, which can then be utilized within MPC [14, 50] or to guide tree search [35]. 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 still a search over actions is 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, since future actions have an influence on the trajectory of the past. Therefore, state predictive models cannot directly be applied 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.

B. Learning Heuristics for TAMP and MIP in Robotics

A general approach to TAMP is to combine discrete logic search with a sampling-based motion planning algorithm [24,

7, 40, 6] or constraint satisfaction methods [27, 28, 31]. 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. [24, 37, 10], to efficiently prune the search tree. Another approach to TAMP is Logic Geometric Programming (LGP) [42, 43, 44, 45, 18], 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 to generate handover motions 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 [19]. 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 [16, 5, 26, 25, 46]. 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 [20, 21, 8]. LGP also can be viewed as a generalization of mixed-integer programs. In [4] (footstep planning) and [21] (planar pushing), learning is used to predict the integer assignments, however, this is 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 of classical TAMP. For example, in [35] the considered scene contains always the same four objects with the same colors, which allows them to have a fixed input vector of separate actions for all objects. In [49] convolutional (CNN) and graph neural networks are utilized to learn a state representation for RL, similarly in [30]. In [2], rendered images from a simulator are used as state representation to exploit the generalization ability of CNNs. 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. [48] and Driess et al. [12] is most related to our approach. They both propose 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 [48] or a scene image [12]. While 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 [48] and [12] could be utilized for sequence prediction.

To our knowledge, the 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.

III. LOGIC GEOMETRIC PROGRAMMING FOR TASK AND MOTION PLANNING

This work relies on Logic Geometric Programming (LGP) [42, 43] as the underlying TAMP framework. The main idea behind LGP is a nonlinear trajectory optimization problem over the continuous variable x, in which the constraints and costs are parameterized by a discrete variable s that represents the state of a symbolic domain. The transitions of this variable are subject to a first-order logic language that induces a decision tree. Solving an LGP involves a tree search over the discrete variable, where each node represents a nonlinear trajectory optimization program (NLP). If a symbolic leaf node, i.e. a node which 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. In this section, we briefly describe LGP for the purpose of this work.

Let $\mathcal{X} = \mathcal{X}(S) \subset \mathbb{R}^{n(S)} \times SE(3)^{m(S)}$ be the configuration space of all objects and articulated structures (robots) as a function of the scene S. The idea is to find a global path x in the configuration space which minimizes the LGP

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

s.t.
$$\forall_{t \in [0,KT]}: \quad h_{\text{eq}}\big(x(t), \dot{x}(t), s_{k(t)}, S\big) = 0 \quad (1\text{b})$$
$$\forall_{t \in [0,KT]}: \quad h_{\text{ineq}}\big(x(t), \dot{x}(t), s_{k(t)}, S\big) \le 0 \quad (1\text{c})$$
$$\forall_{k=1,\dots,K}: \quad h_{\text{sw}}\big(x(kT), \dot{x}(kT), a_k, S\big) = 0 \quad (1\text{d})$$
$$\forall_{k=1,\dots,K}: \quad a_k \in \mathbb{A}(s_{k-1}, S) \quad (1\text{e})$$
$$\forall_{k=1,\dots,K}: \quad s_k = \text{succ}(s_{k-1}, a_k) \quad (1\text{f})$$
$$x(0) = \tilde{x}_0(S) \quad (1\text{g})$$
$$s_0 = \tilde{s}_0(S) \quad (1\text{h})$$
$$s_K \in \mathcal{S}_{\text{goal}}(g). \quad (1\text{i})$$

The path is assumed to be globally continuous ($x \in C([0, TK], \mathcal{X})$) 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 [32, 44]. The functions $c, h_{\text{eq}}, h_{\text{ineq}}$ and hence the objectives in phase k of the motion ($k(t) = \lfloor t/T \rfloor$) are parameterized by the discrete variable (or integers in mixed-integer programming) $s_k \in \mathcal{S}$, representing the state of the symbolic domain. The time discrete transitions between s_{k-1} and s_k are determined by the successor function $\text{succ}(\cdot, \cdot)$, which is a function of the previous state s_{k-1} and the discrete action a_k at phase k. The actions are grounded action operators. Which actions are possible at which symbolic state is determined by the logic and expressed in the set $\mathbb{A}(s_{k-1}, S)$. h_{sw} is a function that imposes transition conditions between the kinematic modes. 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. The quantities $\tilde{x}_0(S)$ and $\tilde{s}_0(S)$ are the scene dependent initial continuous and symbolic states, respectively. For fixed s it is assumed

that c, h_{eq} and h_{ineq} are differentiable. Finally, we define the feasibility of an action sequence $a_{1:K} = (a_1, \dots, a_K)$ as

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

A. Multi-Bound LGP Tree Search and Lower Bounds

The logic induces a decision tree (called LGP-tree) through (1e) and (1f). Solving a path problem as a heuristic to guide the tree search is too expensive. A key contribution of [43] 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), while these lower bounds should be computationally faster to compute. Each node in the LGP tree defines several lower bounds of (1). Still, as we will show in the experiments, a large number of NLPs have to be solved to find a feasible solution for problems with a high combinatorial complexity. This is especially true if many decisions are feasible in early phases of the sequence, but then later become infeasible.

IV. DEEP VISUAL REASONING

The main idea of this work is, given the scene and the task goal as input, to predict a promising discrete action sequence $a_{1:K} = (a_1, \dots, a_K)$ 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 will first describe more precisely what should be predicted, then how the scene, i.e. the objects and actions with 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 heuristic to further guide the search without losing the ability to find a solution if one exists. We additionally propose an alternative way to integrate learning into LGP based on a recurrent feasibility classifier.

A. Predicting Action Sequences

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

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

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 would then choose the sequence from $\mathcal{T}(g, S)$ where the discriminative model has the highest prediction. However, computing $\mathcal{T}(\cdot, \cdot)$ (up to a maximum K) and then checking all sequences with the discriminative model is computationally inefficient.
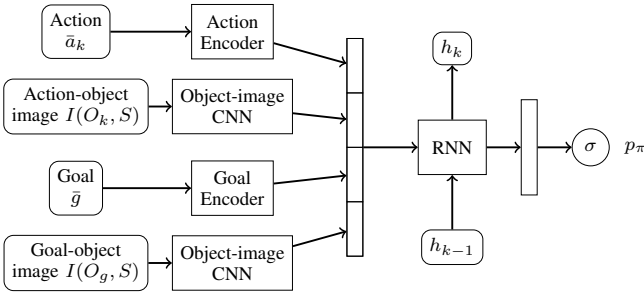
Fig. 2. Proposed neural network architecture.

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 an 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,

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

This way, π generates an action sequence by choosing the action at each step where π has the highest prediction.

B. 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 Sec. IV-D). 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 $a_{1:K^i}^i \in \mathcal{T}(g^i, S^i)$, e.g. with breadth-first search. For each of these sampled sequences, the feasibility of the resulting NLP is determined and saved in the set

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

Based on this dataset, we define the *training* dataset for π as

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

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

$$f_j^i = \begin{cases} 1 & F_{S^i}(a_{1:K^i}^i) = 1 \\ 1 & \exists\ \left(S^l, a_{1:K^l}^l, g^l, F^l \right) \in \mathcal{D}_{\text{data}}: \\ & \qquad F^l = F_{S^l}(a_{1:K^l}^l) = 1 \\ & \qquad \wedge\ g^l = g^i\ \wedge\ a_{1:j}^l = a_{1:j}^i \\ 0 & \text{else} \end{cases} \quad (7)$$

If the action sequence is feasible and solves the problem specified by g^i, then $f_j^i = 1$ 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. $a_{1:j}^l = a_{1:j}^i$, then $f_j^i = 1$ as well (second case). Also in this case the network should suggest to follow this decision, since 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, $f_j^i = 0$, 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.

C. 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 a suitable encoding of a, g and S.

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, similarly for a goal g. While 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. The goal is similarly decomposed into $g = (\bar{g}, O_g)$, $\bar{g} \in \mathcal{G}$, $O_g \in \mathcal{P}(\mathcal{O}(S))$. 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.

Through that separation, since, as mentioned before, the cardinality of $\mathcal{A}$ and $\mathcal{G}$ is constant and independent from the scene, we can input $\bar{a}$ and $\bar{g}$ directly as a one-hot encoding to the neural network.

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 (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 involved in 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 the experiments, the scene image is a depth image, i.e. $n_c = 1$ and the maximum number of objects that are involved in a single action is two, hence $n_O = 2$. If an action takes less objects into account than n_O, this channel is zeroed. Since 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 [12]. However, since each action object image $I(O, S)$ always contains a channel providing information of the complete scene, also the geometric relations to other objects are taken into account. Being able to generate such masks is a reasonable assumption, since there are many methods for that. Please note that these action-object-images always correspond to the *initial* scene.

3) Network Architecture: Fig. 2 shows the network architecture that represents π as a convolutional recurrent neural network. 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 convolutional neural network (CNN). The discrete action/goal symbols $\bar{a}, \bar{g}$ are encoded by fully connected layers with a one-hot encoding as input. Since the only information the network has access to is the *initial* configuration of the scene, a recurrent neural network (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, since 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, since they operate on the same set of object-images. To summarize,

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

D. Relation to Q-Functions

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. However, there are important differences. For example, a Q-function usually relies on a clear notion of state. In our case, the symbolic state s does not contain sufficient information, since it neither includes geometry nor represents the effects of all past decisions on the NLP. Similarly for the continuous state x, which is only defined for a complete action sequence. Therefore, our network has to learn a state representation from the past action-object image sequence, while only observing the initial state

in form of the depth image of the scene as input. Furthermore, we frame learning π as a supervised learning problem.

E. Algorithm

Algo. 1 presents the pseudocode how π is integrated in the tree search algorithm. The main idea is to maintain the set E of expand nodes. A node $n = (s, (\bar{a}, O), k, p_\pi, h, n_{\text{parent}})$ in the tree consists of its symbolic state $s(n)$, action-object pair $(\bar{a}, O)$, 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_{parent}. At each iteration, the algorithm chooses the node n_E^* of the expand list where the network has the highest prediction (line 5). 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 node reaches a symbolic goal state (line 11), it is added to the set of leaf nodes L, else to the expand set. Then the already found leaf nodes are investigated. Since 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). This reduces the number of NLPs that have to be solved. However, one cannot expect that the network never erroneously has a low prediction although the node would be feasible. In order to prevent not finding a feasible solution in such cases, the function adjustFeasibilityThreshold($\cdot$) reduces this threshold with a discounting factor or sets it to zero if all leaf nodes with a maximum depth of K_{max} have been found. This gives us the following.

Proposition 1: Algorithm 1 is complete in the sense that if a scene contains at least one action sequence with maximum length K_{max} for which the nonlinear trajectory optimizer can find a feasible motion, the neural network does not prevent finding this solution, even in case of prediction errors.

As an important remark, we store the hidden state of the recurrent neural network in its corresponding node. Furthermore, the object image and action encodings also have to be computed only once. Therefore, during the tree search, only one pass of the recurrent (and smaller) part of the complete π_{NN} has to be queried in each step.

F. Alternative: Recurrent Feasibility Classifier

The method of [12] and [48] to learn a feasibility classifier considers single actions only, i.e. no sequences. To allow for a comparison we here present an approach to extend the idea of a feasibility classifier to action sequences and how it can be integrated into our TAMP framework. The main idea is to classify the feasibility of an action sequence with a recurrent classifier, independently from the fact whether it has reached a symbolic goal or not. This way, during the tree search solving an NLP can be replaced by evaluating the classifier, which usually is magnitudes faster. Technically, this classifier $\pi_{\text{RC}}(\bar{a}_k, I(O_k, S), h_{k-1})$ has a similar architecture as π_{NN}, but only takes the current action-object-image pair as well as the hidden state of the previous step as input and predicts whether

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:        $(p_\pi, h) = \pi_{\mathrm{NN}} \left(\bar{a}, I(O, S), \bar{g}, I(O_g, S), h(n_E^*) \right)$
9:        $s = \mathrm{succ}(s(n_E^*), (\bar{a}, O)), \quad k = k(n_E^*) + 1$
10:       $n = (s, (\bar{a}, O), k, p_\pi, h, n_E^*)$ ▷ new node
11:       **if** $s \in \mathcal{S}_{\mathrm{goal}}(g)$ **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:       $n_L^* = \underset{n \in L}{\arg\max} \ p_\pi(n)$
19:       **if** $p_\pi(n_L^*) \leq f_{\mathrm{thresh}}$ **then**
20:          $f_{\mathrm{thresh}} \leftarrow \mathrm{adjustFeasibilityThreshold}(f_{\mathrm{thresh}})$
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 $x = P((\bar{a}, O)_{1:K}, g, O_g, S)$
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**
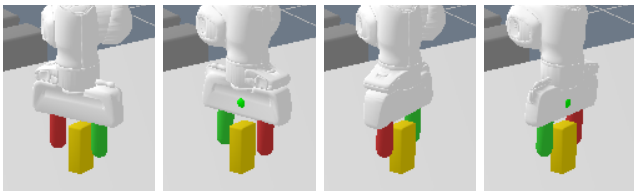29:   **end if**
30:     **end while**
31: **end while**

Fig. 3. The four different integer assignments of the grasp operator.

the action sequence up to this step is feasible. A disadvantage is that just because an action is feasible does not necessarily mean that following it will solve the TAMP problem in the long term. Sec. V-E presents an empirical comparison.

V. EXPERIMENTS

The video https://youtu.be/i8yyEbbvoEk demonstrates the planned motions both in simulation and with a real robot.

A. Setup and Task

We consider a tabletop scenario with two robot arms (Franka Emika Panda) and multiple box-shaped objects, see Fig. 1 for a typical scene, in which the goal is to move an object to different target locations, visualized by a red square in Fig. 1.

1) Action Operators and Optimization Objectives: The logic is described by PDDL-like rules. There are two action operators grasp and place. The grasp action takes as parameters the robot arm and one of four integers η, represented in its discrete action symbol $\bar{a}$, and one object O. Depending on the integer, the end-effector is aligned to different surfaces of the box (equality constraint). Furthermore, an inequality constraint ensures that the center point between the two grippers is inside of the object (with a margin). In Fig. 3 these four discrete ways of grasping are visualized for one robot arm. The exact grasping location relative to the object is still subject to the optimizer. The place action has the robot arm (also encoded in the discrete symbol $\bar{a}$) and two objects as the tuple O as parameters. The effects on the optimization objectives are that the bottom surface of object one touches and is parallel to object two. In our case, object one is a box, whereas object two is the table or the goal location. 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 are squared accelerations on x. There are collisions and joint limits as inequality constraints (with no margin).

2) Properties of the Scene: There are multiple properties which make this (intuitively simple) task challenging for task and motion planning algorithms. First of all, the target location can fully or partially be occupied by another object. Secondly, 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. Thirdly, 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, Tab. I shows 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. 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 [12, 11]. For example, if the object

TABLE I
NUMBER OF ACTION SEQUENCES THAT REACH A SYMBOLIC GOAL STATE

# of objects in the scene	length of the action sequence				
	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

that is occupying the target location is small and the object that should be placed there also, then it can be placed directly, while 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.

B. Training/Test Data Generation and Network Details

We generated 30,000 scenes 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. 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 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 Sec. IV-B. 102,566 of the action sequences in $\mathcal{D}_{\text{data}}$ were feasible, 2,741,573 completely infeasible. This shows the claim of the introduction that the majority of action sequences are actually infeasible. Furthermore, such an imbalanced dataset imposes difficulties for a learning algorithm. With the data transformation from Sec. IV-B, 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.

The network is trained with the ADAM optimizer (learning rate 0.0005) with a batch size of 48. 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. The image encoder consists of three convolutional layers with 5, 10, 10 channels, respectively, and filter size of 5x5. The second and third convolutional layer has a stride of 2. After the convolutional layers, there is a fully connected layer with an output feature size of 100. The same image encoder 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. Since the task is always to place an object at varying locations, we left out the discrete goal encoder in the experiments presented here.

To evaluate the performance and accuracy of our method, we sampled 3000 scenes, again containing two objects each, with the same algorithm as for the training data, but with a different random seed. Using breadth-first search, we determined
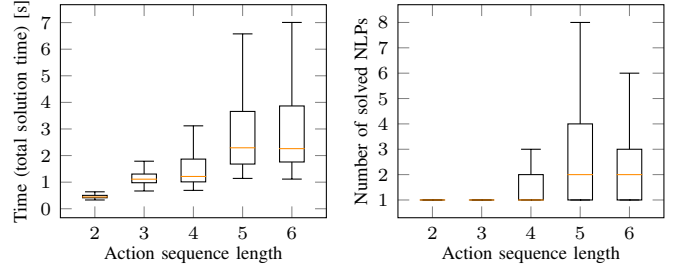

Fig. 4. Total time (left) and number of solved NLPs (right) to find an overall feasible solution for the test scenes with neural network
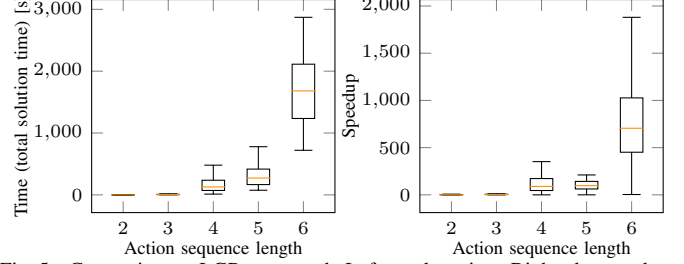

Fig. 5. Comparison to LGP tree search. Left: total runtime. Right: the speedup of our neural network (sol. time with NN / sol. time with LGP tree search).

2705 feasible scenes, which serve as the actual test scenes.

C. Performance – Results on Test Scenarios

Fig. 4 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 computing the image/action encodings, querying the neural network during the search and all involved NLPs that are solved. As one can see, for all cases with sequence lengths of 2 and 3, the first predicted sequence is feasible, such that there is no search and only one 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 two NLPs have to be solved. Generally, 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. Furthermore, the upper whiskers are also below 7 s. All experiments have been performed with an Intel Xeon W-2145 CPU @ 3.70GHz.

D. Comparison to Multi-Bound LGP Tree Search

In Fig. 5 (left) 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 to only 3 with the neural network. Fig. 5 (right) 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 (median). In this plot, only those scenes where LGP tree search and the neural network have found the same sequence lengths are compared.

E. Comparison to Recurrent Classifier

Fig. 6 (left) shows a comparison of our proposed goal-conditioned network that generates sequences to a recurrent classifier that only predicts feasibility of an action sequence, independent from the task goal. As one can see, while such a
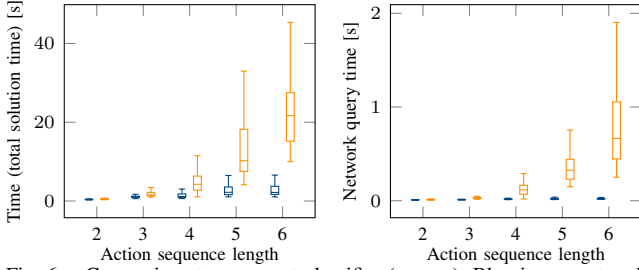
Fig. 6. Comparison to recurrent classifier (orange). Blue is our network.

(a) Generalization to multiple objects (b) Generalization to cylinders

Fig. 7. Generalization experiments.

classifier also leads to a significant speedup compared to 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 to 3 with our approach. While the network query time is neglectable for our network, as can be seen in Fig. 6 (right), the time to query the recurrent classifier becomes visible.

F. 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 Sec. V-B 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. Fig. 7a 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. There was not a single scene where no solution is found. While the upper quartile increases, the median is not significantly affected by the presence of multiple objects. Generally, the performance is remarkable, especially when looking at Tab. I where for sequence length 6 there are nearly half a million possible action sequences. The fact that the runtime increases for more objects is not only caused by the fact that the network inevitable does 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 collision queries and increased non-convexity.

G. Generalization to Cylinders

Although the network has been trained on box-shaped objects only, we investigate if the network can generalize to scenes which contain other shapes like cylinders. Since the objects are encoded in the image space, there is a chance that, as compared to 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
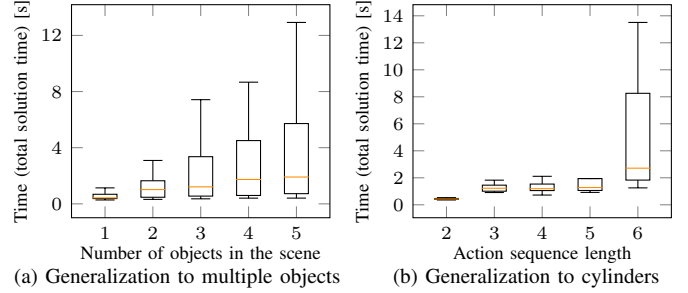
larger radii such that they have a clearly different appearance than what is contained in the dataset. Fig. 7b shows the total solution time with the neural network. As one can see, except for action sequences of length 6, there is no drop in performance compared to box-shaped objects, which indicates that the network is able to generalize to other shapes. Even for length 6, the runtimes are still very low, especially compared to LGP tree search. 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 becomes a problem in its own.

H. Real Robot Experiments

Fig. 1 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. Since the yellow object is large enough, the network proposed a handover solution (Fig. 1c). The presence of an additional object (green) does not confuse the predictions. The planned trajectories are executed open-loop. The images as input to the neural network are rendered from object models obtained by a perception pipeline, therefore, the transfer to the real robot is directly possible.

VI. CONCLUSION

In this work, we proposed a neural network that learned to predict promising discrete action sequences for TAMP 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 could act as a heuristic, there was no real search over the discrete variable and consequently only one trajectory optimization problem had to be solved to find a solution to the TMAP problem.

Although being trained on only two objects present at a time, the learned representation of the network enabled to generalize to scenes with multiple objects (and other shapes to some extend) while still showing a high performance.

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

REFERENCES

[1] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, pages 8289–8300, 2018.

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

[3] Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research*, 2011.

[4] Justin Carpentier, Rohan Budhiraja, and Nicolas Mansard. Learning feasibility constraints for multicontact locomotion of legged robots. In *Robotics: Science and Systems*, 2017.

[5] Rohan Chitnis, Dylan Hadfield-Menell, Abhishek Gupta, Siddharth Srivastava, Edward Groshev, Christopher Lin, and Pieter Abbeel. Guided search for task and motion plans using learned heuristics. In *International Conference on Robotics and Automation (ICRA)*, pages 447–454. IEEE, 2016.

[6] Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. An incremental constraint-based framework for task and motion planning. *International Journal on Robotics Research*, 2018.

[7] Lavindra de Silva, Amit Kumar Pandey, Mamoun Gharbi, and Rachid Alami. Towards combining HTN planning and geometric task planning. *CoRR*, 2013.

[8] Neel Doshi, Francois R Hogan, and Alberto Rodriguez. Hybrid differential dynamic programming for planar manipulation primitive. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[9] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *International Conference on Learning Representations ICLR*, 2017.

[10] Danny Driess, Ozgur Oguz, and Marc Toussaint. Hierarchical task and motion planning using logic-geometric programming (HLGP). RSS Workshop on Robust Task and Motion Planning, 2019.

[11] Danny Driess, Syn Schmitt, and Marc Toussaint. Active inverse model learning with error and reachable set estimates. In *Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[12] Danny Driess, Ozgur Oguz, Jung-Su Ha, and Marc Toussaint. Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[13] Frederik Ebert, Chelsea Finn, Alex X. Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. In *Conference on Robot Learning*, 2017.

[14] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[15] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2016.

[16] Caelan Garrett, Leslie Kaelbling, and Tomas Lozano-Perez. Learning to rank for synthesizing planning heuristics. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2016.

[17] Jung-Su Ha, Young-Jin Park, Hyeok-Joo Chae, Soon-Seo Park, and Han-Lim Choi. Adaptive path-integral autoencoders: Representation learning and planning for dynamical systems. In *Advances in Neural Information Processing Systems*, pages 8927–8938, 2018.

[18] Jung-Su Ha, Danny Driess, and Marc Toussaint. Probabilistic framework for constrained manipulations and task and motion planning under uncertainty. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[19] Valentin N Hartmann, Ozgur S Oguz, Danny Driess, Marc Toussaint, and Achim Menges. Robust task and motion planning for long-horizon architectural construction planning. arXiv:2003.07754, 2020.

[20] François Robert Hogan and Alberto Rodriguez. 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)*, 2016.

[21] Francois Robert Hogan, Eudald Romo Grau, and Alberto Rodriguez. Reactive planar manipulation with convex hybrid MPC. In *International Conference on Robotics and Automation (ICRA)*, pages 247–253. IEEE, 2018. URL https://doi.org/10.1109/ICRA.2018.8461175.

[22] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *Robotics and Automation Letters*, 4(3):2407–2414, 2019.

[23] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[24] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical planning in the now. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[25] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 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*, 2018.

[26] Beomjoon Kim, Zi Wang, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Learning to guide task and motion planning using score-space representation. *The International Journal of Robotics Research*, 38(7):793–812, 2019.

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

[28] Fabien Lagriffoul, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti, and Lars Karlsson. Efficiently combining task and motion planning using geometric constraints. *International Journal on Robotics Research*, 2014.

[29] Sascha Lange, Martin A. Riedmiller, and Arne Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *IJCNN*, 2012.

[30] Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. Towards practical multi-object manipulation using relational reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[31] Tomás Lozano-Pérez and Leslie Pack Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *Proc. of the Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[32] Matthew Mason. The mechanics of manipulation. In *Int. Conf. on Robotics and Automation (ICRA'85)*. IEEE, 1985.

[33] Masashi Okada, Luca Rigazio, and Takenobu Aoshima. Path integral networks: End-to-end differentiable optimal control. *arXiv preprint arXiv:1706.09597*, 2017.

[34] Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sébastien Racanière, David P. Reichert, Theophane Weber, Daan Wierstra, and Peter W. Battaglia. Learning model-based planning from scratch. *CoRR*, abs/1707.06170, 2017.

[35] Chris Paxton, Yotam Barnoy, Kapil D. Katyal, Raman Arora, and Gregory D. Hager. Visual robot task planning. In *International Conference on Robotics and Automation (ICRA)*, pages 8832–8838. IEEE, 2019.

[36] Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in neural information processing systems*, 2017.

[37] Ismael Rodriguez, Korbinian Nottensteiner, Daniel Leidner, Michael Kasecker, Freek Stulp, and Alin Albu-Schäffer. Iteratively refined feasibility checks in robotic assembly sequence planning. *Robotics and Automation Letters*, 2019.

[38] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, 2017.

[39] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning (ICML)*, pages 4739–4748, 2018. URL http://proceedings.mlr.press/v80/srinivas18b.html.

[40] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart J. Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. of the Int. Conf. on Robotics and Automation (ICRA)*, 2014.

[41] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[42] Marc Toussaint. 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*, pages 1930–1936. AAAI Press, 2015. URL http://ijcai.org/Abstract/15/274.

[43] Marc Toussaint and Manuel Lopes. Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In *International Conference on Robotics and Automation (ICRA)*, pages 4044–4051. IEEE, 2017.

[44] Marc Toussaint, Kelsey R Allen, Kevin A Smith, and Josh B Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In *Proc. of Robotics: Science and Systems (R:SS)*, 2018.

[45] Marc Toussaint, Jung-Su Ha, and Danny Driess. Describing physics for physical reasoning: Force-based sequential manipulation planning. arXiv:2002.12780, 2020.

[46] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Active model learning and diverse action sampling for task and motion planning. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 4107–4114. IEEE, 2018.

[47] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin A. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pages 2746–2754, 2015.

[48] Andrew M Wells, Neil T Dantam, Anshumali Shrivastava, and Lydia E Kavraki. Learning feasibility for task and motion planning in tabletop environments. *Robotics and Automation Letters*, 4(2):1255–1262, 2019.

[49] Matthew Wilson and Tucker Hermans. Learning to manipulate object collections using grounded state representations. *Conference on Robot Learning*, 2019. URL https://arxiv.org/abs/1909.07876.

[50] Annie Xie, Frederik Ebert, Sergey Levine, and Chelsea Finn. Improvisation through physical understanding: Using novel objects as tools with visual foresight. In *Proc. of Robotics: Science and Systems (R:SS)*, 2019.