Learning Geometric Reasoning and Control for Long-Horizon Tasks from Visual Input

Danny Driess*[1,2] Jung-Su Ha*[2,3] Russ Tedrake[4] Marc Toussaint[2,3]

***Abstract*— Long-horizon manipulation tasks require joint reasoning over a sequence of discrete actions and their associated continuous control parameters. While Task and Motion Planning (TAMP) approaches are capable of generating motion plans that account for this joint reasoning, they usually assume full knowledge about the environment (e.g. in terms of shapes, poses of objects) and often require computation times not suitable for real-time control.**

To overcome this, we propose a learning framework where a high-level reasoning network predicts, based on an image of the scene, a sequence of discrete actions *and* the parameter values of their associated low-level controllers. These controllers are parameterized in terms of a learned energy function, leading to time-invariant controllers for each phase. We train the whole framework end-to-end using a dataset of TAMP solutions computed using Logic Geometric Programming. A key feature is that the reasoning network determines the parameters of the controllers jointly, such that the overall task can be solved. Despite having no explicit representation of the geometry nor pose of the objects in the scene, our network is still able to accomplish geometrically precise manipulation tasks, including handovers and an accurate pointing task where the parameters of early actions are tightly coupled with those of later actions. Video: `https://youtu.be/AcPWRTkr3_g`

I. Introduction

Long-horizon sequential manipulation problems are challenging since they require decisions both over discrete and high-dimensional continuous aspects of the problem. Contact activities at different phases of the motion often imply a combinatorial problem and the non-smoothness around contacts requires special treatment for motion planning methods to succeed. Given a sequence of discrete actions, a motion planner has to find geometric paths which are consistent not only with one discrete action, but also with all other actions and the goal, e.g., an object or a tool might have to be grasped in a specific way in order for later actions to be realizable.

Task and Motion Planning (TAMP) has developed methods to deal with these issues by combining reasoning on a symbolic level with continuous motion planning, allowing them to compute motion plans for long-horizon problems [1]–[4]. However, TAMP usually assumes full knowledge about the environment, e.g. in terms of shapes and poses of all objects, which is difficult to obtain from perception. At the same time, even if the environment was fully observed, due to the high combinatorial complexity of discrete decisions and the difficulty of motion planning itself, solving TAMP problems often demands computation times which are not suitable for real-time control. In [5], the combinatorial complexity of the discrete decisions has been addressed by learning to predict feasible action sequences for a TAMP problem from an image of the scene, which greatly reduces the number of motion planning problems that have to be solved. To compute the actual motions via trajectory optimization, the shapes and poses of the objects are still necessary.

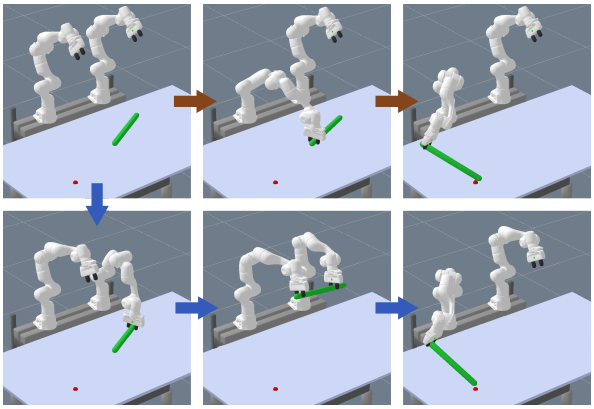

Fig. 1. Typical scene: The goal is to touch the red goal location, which is out of reach of the robots. Therefore, the arms have to utilize the green stick directly (brown path) or coordinate a handover motion (blue path).

The present work fully removes the requirement of pose and shape estimation and realizes a vision-based version of TAMP including control. In order to achieve this, we propose a hierarchical framework that consists of a high-level reasoning/planning module and low-level controllers, both of which are learned jointly. Based on an image of the initial scene, the high-level reasoning module predicts not only the feasibility of a discrete action sequence, but also a sequence of vectors that represent both the actions *and* their associated control parameters which in turn define the closed-loop behavior of the low-level controllers. The action sequence itself and also the feedback motions are computed from observations only. Therefore, our work can be seen as an attempt to bridge perception, TAMP and control.

To generate the training data for the whole network architecture, we utilize a TAMP solver, Logic Geometric Programming (LGP) [6], to generate expert trajectories. The dataset of optimal control transitions is created through model predictive control (MPC) built locally around the LGP solution trajectories. Compared to a standard behavior cloning objective, we additionally exploit the information encoded in the Hessian of the LGP/MPC solution to realize a more informative training objective function.

The low-level controllers are derived from an energy-based model of learned potential functions. Since our reasoning network retains the property of a TAMP solver to provide multiple solutions to the problem (if they exist), connected with the value function from LGP, this energy model allows

*equal contribution

[1]University of Stuttgart. [2]MPI for Intelligent Systems.
[3]TU Berlin. [4]Massachusetts Institute of Technology.

DD thanks the International Max-Planck Research School for Intelligent Systems for the support. MT thanks the Max-Planck Fellowship at MPI-IS.

prioritization of the multiple solutions.

II. Related Work

A. Task and Motion Planning

Many TAMP approaches combine search on a logical level with sampling based motion planning [1], [7]–[10], nonlinear trajectory optimization [4], [11]–[13] or constraint satisfaction methods [14], [15]. Commonly, the interaction between the two domains is realized by the logic solver proposing discrete action sequences for which the motion planner either finds a feasible motion or returns infeasibility. Due to the combinatorics of possible discrete actions and geometric constraints, many motion planning problems have to be solved to find a feasible solution. Therefore, solving TAMP often demands computation times beyond the requirements for turning such plans into real-time, reactive controllers. Overcoming this is an active field of research. For example, in [12] TAMP is defined in object-centric coordinates, which enables to derive controller that can react to (small) perturbations. However, their approach relies on state estimation during execution. Solving mixed-integer programs for control of hybrid systems suffers from similar combinatorial issues [16], [17], which requires, e.g., warm-starting [18] to make it suitable for control. Our work utilizes Logic Geometric Programming (LGP) [11] as an optimization-based approach for TAMP to generate expert solutions for training our framework. While there have been advances in deriving controllers from LGP solutions [19], they still rely on full knowledge of the environment in terms of shapes and poses of objects and one first has to find a feasible action sequence and solve its corresponding trajectory optimization problem. Our approach addresses these three issues. Given an action sequence, our network directly predicts its feasibility, without having to solve a motion planning problem. Further, it outputs the sequence of vectors that parameterize a low-level feedback policy. Since these low-level policies are trained end-to-end to solve the TAMP problems only from such vectors, these vectors can be considered as a latent space encoding, not only of the state of the environment, but also of the solution of the TAMP problem, allowing the low-level policy to be greedy. Finally, all this is done based on an initial image of the scene.

B. Learning for Long-Horizon Tasks/TAMP

Learning planning behaviors is of great interest [20]–[25]. In [26]–[33] action-conditioned forward models, e.g. in the image space, are learned for planning. It is, as mentioned in [34]–[37], however, difficult to scale up image-based forward prediction methods to solve long-horizon tasks. Several works consider learning a heuristic for the symbolic level to speed up finding a solution to a TAMP problem [2], [5], [38]–[44]. Most of these approaches, however, rely on full knowledge of the environment for generating the motions themselves. Addressing planning and low-level control jointly for long-horizon tasks is usually realized in a hierarchical framework consisting of a high-level planning module and low-level controllers [45]–[50]. Most related to our approach is [51], where they propose a learning framework that consists of a sensor-based symbolic state observer and a low-level control policy. A symbolic planner (based on [52]) takes the predicted symbolic states from the observer and produces a discrete plan for the low-level policy. Also in [53], long-horizon tasks are assumed to be decomposed into subtasks represented by action symbols and they propose a framework to predict those symbols from images. However, the tasks most of these works consider do not require joint reasoning over continuous parameters. The problems we address cannot be represented only by such symbols so the geometric aspects of the problem should be considered together. In [54], latent representations of skills are learned in a self-supervised manner, which act as high-level commands. Similarly in [55], [56], high-level modules are trained from play data to predict sub-goals, which guide low-level policies. Compared to these, our framework is trained using more structured, goal-directed data from LGP. Lastly in [57], an affordance model for skills is trained through trial-and-error, which, with a learned transition model, allows for long-horizon reasoning. While they assume to have a policy for parameterized skills, our method concurrently learns control parameters and a policy.

C. Learning from Optimal Control/Trajectory Optimization

There have been several works that leverage optimal control and trajectory optimization to train a neural network control policy. In [58], optimization problems w.r.t. trajectory and policy parameters are addressed together using the Alternating Direction Method of Multipliers (ADMM) and iterative LQG. Guided policy search [59] also formulates policy learning as a combined optimization problem and solves it using ADMM. While these approaches have shown the great opportunity of utilizing optimal control methods as a means of data generation, they only address single-phase smooth problems. When it comes to long-horizon tasks/TAMP, where the optimal behavior often has multiple phases and cannot be represented solely by the continuous state, it becomes unclear how to encode discrete aspects and the reasoning process into the policy network.

III. Background of Logic Geometric Programming for TAMP

In this section, we describe LGP [6] in a discrete time formulation as the TAMP framework for this work. For a scene S, in LGP, a first-order logic planner proposes a sequence of discrete actions $a_{1:K}$, a so-called skeleton, which, via its uniquely implied symbolic state sequence $s_{1:K}$, imposes costs and constraints on a (smooth) nonlinear trajectory optimization problem over the time discretized path x. Solving an LGP for a goal g

$$P(g,S) = \min_{\substack{K\in\mathbb{N} \\ x_{1:KT}, x_t\in\mathcal{X} \\ a_{1:K},\ s_{1:K}}} \sum_{t=1}^{KT} f\big(x_t, x_{t-1}, s_{k(t)}, S\big) \tag{1a}$$

$$\text{s.t.}$$

$$\forall_{t\in 1,\dots,KT}: \quad h_{\text{eq}}\big(x_t, x_{t-1}, s_{k(t)}, S\big) = 0 \tag{1b}$$

$$\forall_{t\in 1,\dots,KT}: \quad h_{\text{ineq}}\big(x_t, x_{t-1}, s_{k(t)}, S\big) \le 0 \tag{1c}$$

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

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

$$x_0 = \tilde{x}_0(S), \quad s_0 = \tilde{s}_0(S), \quad s_K \in \mathcal{S}_{\text{goal}}(g) \tag{1f}$$

means to find an action sequence for which the resulting nonlinear program over the path $x_{1:KT}$ is feasible. The path x in the configuration space $\mathcal{X} \subset \mathcal{Q} \times SE(3)^m$, $\mathcal{Q} \subset \mathbb{R}^n$ of all objects and robots ($\mathcal{Q}$ is the joint space of the robot) consists of $K \in \mathbb{N}$ phases, each discretized with T steps. The functions $f, h_{\text{eq}}, h_{\text{ineq}}$ and hence the costs/constraints in phase k of the motion ($k(t) = \lfloor t/T \rfloor$) are parameterized by the symbolic state $s_k \in \mathcal{S}$. The transitions between s_{k-1} and s_k are determined by a first-order logic language as a function of the previous state s_{k-1} and the discrete action a_k. These actions $a_k \in \mathbb{A}(s_{k-1}, S)$ are grounded action operators.

IV. Reasoning and Control from Visual Input

The overall goal of this work is to find a controller that solves a sequential manipulation task from sensor observations. To make this tractable, we follow the assumption of LGP that such tasks can be divided into K phases that correspond to high-level actions. Instead of trying to learn one monolithic controller, we want to predict, based on an observation of the scene (in this case an image), a sequence of discrete actions $a_{1:K} = (a_1, \dots, a_K)$ and their corresponding controllers π_k that stabilize each phase. K is part of the decision problem. However, a crucial property of such manipulation problems is that the discrete actions typically do not fully determine the exact motions of each phase, such that in many cases the behavior of the controllers, e.g. in terms of their convergence points *and* the path towards them, in each phase has to be coordinated globally to make them consistent with earlier and future actions. An action, e.g., might specify that an object has to be grasped, but there are still (infinitely) many ways of grasping, hence the actual grasp has to be coordinated with the other actions.

Therefore, we introduce a hierarchical framework, visualized in Fig. 2, consisting of a high-level reasoning network that, given a sensor observation I of the initial scene (in our case an image), not only decides the feasibility of a discrete action sequence $a_{1:K}$, but especially also predicts a sequence of vector representations $\hat{c}_{1:K}$, $\hat{c}_k \in \mathbb{R}^{n_c}$ that parameterize the low-level controllers $\pi(\cdot, \hat{c}_k)$ of each phase k. These controllers are time-invariant feedback policies. All information about the scene and the goal (geometry etc.) that is necessary for the current controller is compressed into $\hat{c}_k$, such that the controller is not explicitly a function of the goal or other actions. This way, $\hat{c}_{1:K}$ can be seen as a (latent) representation of the scene for the chosen action sequence. Once the controller of the current phase converges, the system transitions to the next phase where the controller is parameterized by $\hat{c}_{k+1}$. In this sense, the high-level reasoning network can be understood as a planning network that coordinates the motions globally based on an initial observation of the scene. In short, our framework transfers both the logic reasoning and continuous trajectory optimization computations of LGP into a neural network architecture that only relies on sensor data and provides time-invariant controllers for each phase.

A. High-Level Geometric Reasoning Network

Given an initial sensor observation I (e.g. a depth image) of the scene and a candidate action sequence $a_{1:K}$, $a_k \in \mathbb{A}$,

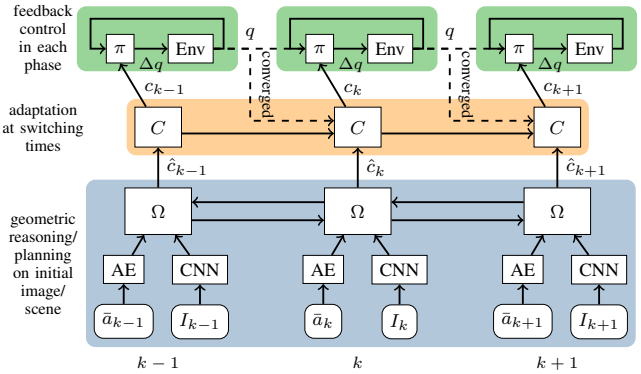

Fig. 2. Network architecture consisting of high-level reasoning network in blue, adaptation network in orange and low-level controllers in green. The reasoning network is queried only once on the initial image of the scene to generate the sequence of vector representations $\hat{c}_{1:K}$. During execution, the controllers of each phase are time-invariant feedback policies. When they converge, the system transitions to the next phase $k+1$, where an observation of the current q is taken into account in C, transforming the initially predicted $\hat{c}_{k+1}$ by Ω into c_{k+1} for the next controller.

the high-level reasoning network Ω

$$\Omega: \ (a_{1:K}, I) \mapsto (\hat{c}_{1:K}, z, \bar{v}_{1:K}) \tag{2}$$

predicts a sequence of vector representations $\hat{c}_{1:K}$, $\hat{c}_k \in \mathbb{R}^{n_c}$ that will parameterize the low-level controllers (see Sec. IV-B.1), an exponentiated cost prediction of that action sequence ($z = \exp(-V_1) \in [0, 1]$ with V_1 cost-to-go) as well as a sequence of cost bias terms $\bar{v}_{1:K}$, $\bar{v}_k \in \mathbb{R}$ (the latter will be explained in Sec. IV-B.2). With the cost prediction z, we can find the set of action sequences $\mathcal{F} = \{a_{1:K} \in \mathcal{T}(S, g) \ : \ z(a_{1:K}) > \beta\}$ which are classified as feasible by Ω for the threshold $\beta > 0$ (for an infeasible sequence we have $V_1 = \infty$ and therefore $z = \exp(-\infty) = 0$, hence infeasibility prediction for $z < \beta$). $\mathcal{T}(S, g)$ is the set of all action sequences that fulfill the constraints (1d)-(1f).

The actions a in LGP and in our framework are grounded action operators, which means that they simultaneously represent the action operator symbol and the objects in the scene they are referring to. The question arises on how to encode a as input to Ω. Following our previous work [5], we split an action into $a = (\bar{a}, O)$ with $\bar{a} \in \mathcal{A}$ being the discrete action operator symbol and $O \in \mathcal{P}(\mathcal{O}(S))$ the objects a operates on. For example, $\bar{a}$ could indicate the abstract action operator `grasp`, whereas O would indicate which actual object in the scene should be grasped. This separation allows us to directly encode $\bar{a}$ in a one-hot encoding (called AE in Fig. 2). Further, we can map the objects O to a so-called action-object-image via $I : O \mapsto \mathbb{R}^{(1+n_O) \times w \times h}$, which is the stacking of a depth image of the complete scene with masks of the objects O that are relevant for the action. Therefore, the reasoning network operates on the those action-object-images that are extracted from the initial depth observation in terms of object masks and the depth channel. As has been argued in [5], an image representation has the advantage that it has a fixed dimension, independent from the number of objects in the scene. Extracting masks is, although still challenging, considered to be more tractable from raw sensory input than pose and complete shape estimation [60]. The action-object images are encoded via a convolutional neural network (CNN in Fig. 2). Based on the sequence of action operator symbols

$\bar{a}_{1:K}$ and action-object images $(I(O_k))_{k=1:K}$, we make the concrete input to the reasoning network more explicit by $(\hat{c}_{1:K}, z, \bar{v}_{1:K}) = \Omega\big(\bar{a}_{1:K}, (I(O_k))_{k=1:K}\big)$. We chose Ω as a bidirectional recurrent network to enable the network to reason over the whole sequence $\hat{c}_{1:K}$ jointly.

B. Learning Energy Functions as Low-Level Policy Network

1) Control via Learned Energy Functions: The idea behind our control framework is a class of policies π, parameterized by the vector c, which map the current joint configuration $q \in \mathcal{Q}$ to joint velocities Δq in discrete time, $q_{\text{next}} = q + \pi(q, c)$. This vector c defines not only the attractor manifold itself where the controller should be in equilibrium, but also the vector field, i.e. the path towards it. This is important for robot manipulation where the controller within one phase of the motion should take care of collision avoidance while converging to an equilibrium state.

Instead of a direct policy that maps the current joint configuration of the robot to joint velocities, our framework learns a neural network-based energy function $E_c(q) = \frac{1}{2}\|\psi(q, c)\|^2$ to shape the attractor behavior, where $\psi : \mathcal{Q} \times \mathbb{R}^{n_c} \to \mathbb{R}^{n_E}$ defines an abstract feature. With this, the control policy is defined through the energy minimization problem

$$q_{\text{next}} = \underset{\tilde{q} \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2}||\tilde{q} - q||_R^2 + \frac{1}{2}||\psi(\tilde{q}, c)||^2, \tag{3}$$

where the control cost matrix R, which comes from (1a), trades off control cost with minimizing the energy. By linearization, the solution of (3) can be approximated as

$$\begin{aligned}\pi(q, c) &\approx \underset{\Delta q \in \mathbb{R}^n}{\operatorname{argmin}} \left(\frac{1}{2}\Delta q^T R \Delta q + \frac{1}{2}||\psi(q, c) + J\Delta q||^2\right) \\ &= -(R + J^T J)^{-1} J^T \psi(q, c),\end{aligned} \tag{4}$$

where $J = \frac{\partial \psi}{\partial q}(q, c)$ is the Jacobian of ψ w.r.t. q at q, c.[1] Furthermore, the energy function can utilize priors in terms of $\frac{1}{2}||\psi(\phi(q), c)||^2$ where ϕ is a task space, e.g. forward kinematics. In our case, we define $\phi(q)$ to be positions of three points on the end-effectors.[2]

2) Skeleton Selection via Value Interpretation of Energy Function: Another view on the energy-based policy (3) is the Bellman equation, where the energy function acts as a value function encoding the optimal cost-to-go. Following this interpretation, we train the energy network also to resemble the value function (Sec. V-B.3), with which the cost-to-go of an action sequence $a_{1:K}$ is predicted as

$$\tilde{V}_{a_{1:K}}(q, k) \approx E_{c_k(a_{1:K})}(q) + \bar{v}_k(a_{1:K}), \tag{5}$$

where the robot state independent bias term $\bar{v}_{1:K}$ is one output of the Ω network. By evaluating the feasible action sequences at the current configuration (which could have been perturbed by disturbance), our framework chooses the one with the minimum $\tilde{V}$ to execute in phase k as

$$\Delta q = \pi(q, c_k(a^*_{1:K})),\ a^*_{1:K} = \underset{a_{1:K} \in \mathcal{F}}{\operatorname{argmin}} \tilde{V}_{a_{1:K}}(q, k), \tag{6}$$

which means that the robot is capable of deciding the optimal skeleton to follow, i.e. whether to stay at the currently executed one or switch to another.

C. Control Parameter Adaptation Network

The reasoning network Ω predicts the sequence of control parameters $\hat{c}_{1:K}$ from an initial observation of the scene. Due to disturbance or other imperfections during execution, however, a reactive controller could for example have chosen a different grasping location than anticipated in the plan (because that was optimal for the perturbed state). This implies that the behavior of the next phase should be adapted accordingly, meaning that $\hat{c}$ predicted with the initial observation only cannot solely define the motion in the next phase. To make the controller Markovian to the past phases, we adjust the parameter during execution by a recurrent filter

$$c_k = C(\hat{c}_k, \hat{c}_{k-1}, \dots, \hat{c}_1, p_k, p_{k-1}, \dots, p_1) \tag{7}$$

that transforms the predicted $\hat{c}_k$ into the filtered $c_k \in \mathbb{R}^{n_c}$ (which is then the actual input to π) by taking an observation p_k at the switching time, i.e. when the controller has converged, into account. In our case, $p_k = \phi(q_k)$ is used, i.e. the position features of the end-effectors at switching time.

V. Training

A. Data Generation via MPC

A solution of a TAMP algorithm is usually a (collision free) trajectory from the initial to the goal configurations of each phase. To train a controller, we need to generate control transitions around the trajectory for perturbed states. To achieve this, we build a model predictive controller (MPC) from the LGP solution. This controller is then capable of generating (approximate) optimal transitions around the solution trajectory by solving one-step nonlinear programs

$$\begin{aligned}x_t^{**} = \ &\underset{x_t \in \mathcal{X}}{\operatorname{argmin}} \big[f_t(x_{t-1:t}) + V_{t+1}(x_t)\big] \\ &\text{s.t.} \quad h_{\text{eq},t}(x_{t-1:t}) = 0,\ h_{\text{ineq},t}(x_{t-1:t}) \le 0,\end{aligned} \tag{8}$$

with $V_{KT+1} \triangleq 0$, where f_t, $h_{\text{eq},t}$, and $h_{\text{ineq},t}$ are the cost, equality and inequality constraints at the current time step t of the LGP (1) for a fixed skeleton, respectively, i.e. $h_{\text{eq},t}(x_{t-1:t}) = h_{\text{eq}}(x_{t-1}, x_t, s_{k(t)}, S)$ (f_t, $h_{\text{ineq},t}$ analogue). The quadratic approximation of the cost-to-go function V_t

$$V_t = \min_{x_{t:KT}} \sum_{\tau=t}^{KT} f_\tau(x_{\tau-1:\tau}) \text{ s.t. } \forall_{\tau=t}^{KT} : \begin{matrix} h_{\text{eq},\tau}(x_{\tau-1:\tau}) = 0 \\ h_{\text{ineq},\tau}(x_{\tau-1:\tau}) \le 0 \end{matrix} \tag{9}$$

can be computed analytically backwards in time via k-order dynamic programming (KODP) [19] using the 1st- and 2nd-order Taylor expansions of the cost and constraints around the optimal solution trajectory $x^*_{1:KT}$ from (1) as $V_{t+1}(x_t) \approx \frac{1}{2}\delta x_t^T \nabla^2 V_{t+1} \delta x_t + v_{t+1}^T \delta x_t + \bar{v}_{t+1}$, $f_t(x_{t-1:t}) \approx \frac{1}{2}\delta x_{t-1:t}^T \nabla^2 f_t^* \delta x_{t-1:t} + (\nabla f_t^*)^T \delta x_{t-1:t} + f_t^*$, $h_t(x_{t-1:t}) \approx (\nabla h_t^*)^T \delta x_{t-1:t}$, where $\delta x_\cdot = x_\cdot - x_\cdot^*$, $f_t^* = f_t(x^*_{t-1:t})$

[1] Since the neural network Jacobians are included in the computation graph, the network should be of class C^2. We use the soft-plus activation for this network instead of commonly used ReLUs which would not work.

[2] π can also be interpreted as Riemannian motion policies [61] or operational space control, where the operational space is the learned ψ, with a pull-back metric $J^T J$, such that $\dot{e} = -e$ in that space and $\dot{q} = 0$ (with metric R) in the joint space describe the closed-loop optimal behavior.

etc., and h_t denotes the equality constraints and activated inequality constraints. The detailed derivation of KODP can be found in our previous work [19]. The controller (8) with the approximate cost-to-go V_t is time-varying and only valid locally around the linearization point. Note that this is not a linear controller since we use the nonlinear f_t, $h_{\text{eq},t}$, $h_{\text{ineq},t}$ in (8). To obtain the optimal transitions from various configurations but not too far from the solution, we sample trajectories with MPC as follows: starting from the initial configuration x_0 and $t = 1$, we first inject Gaussian disturbance into the current configuration, then we solve the MPC optimization (8) to compute $\Delta x_t^{**} = x_t^{**} - x_{t-1}$, save the data and transition to the next time step. Applying this procedure to many different scenes S leads to the dataset

$$\mathcal{D} = \Big\{ \big(S, q_{0:KT-1}, \Delta q^{**}_{1:KT}, (\bar{a}, I)_{1:K}, v^{**}_{2:KT+1}, z^*, \nabla^2 \tilde{V}^{**}_{2:KT+1}, \nabla^2 \tilde{f}^{**}_{1:KT}, \nabla \tilde{h}^{**}_{1:KT}\big)^{(i)} \Big\}_{i=1}^{N}$$

where $(\bar{a}, I)_{1:K}$ is the sequence of discrete actions and their corresponding action-object images, $q_{0:KT-1}$ the robot joint configuration, $\Delta q^{**}_{1:KT}$ their optimal control transition from MPC (subspace of Δx corresponding to the robot joints), $v^{**}_{2:KT+1}$ the cost-to-go at the next time step along the trajectory and $z^* = \exp(-V_1^*) \in [0, 1]$ the exponentiated cost of the skeleton in the noiseless case with $z^* = 0$ ($V_1^* = \infty$) denoting infeasibility of this action sequence (in which case all other quantities are not computed). The last three quantities are submatrices corresponding to q of the Hessians of V, f and the Jacobian of h (all w.r.t. x_t), which will be used to define the loss function for training.

B. Loss Function

The whole architecture of the reasoning network Ω, the adaptation network C, and the energy mapping network ψ are trained end-to-end. For each trajectory i in the dataset from MPC, we have the loss $L^{(i)} = L_{\Delta q} + w_z L_z + w_v L_v$.

1) Δq-Loss: The controllers should learn to replicate the optimal behavior of the MPC transitions. The most natural and widely used loss function for behavior cloning is the squared Euclidean distance $\|\pi(q, c) - \Delta q^{**}\|_2^2$ between the optimal MPC transitions and the prediction of the neural controller. However, the output of the MPC optimization contains more information than just its solution (the optimal transition). Roughly speaking, we want to optimize the neural network weights θ of the controller to learn transitions that minimize the local value functions (9), just like what MPC does w.r.t. the LGP solution (8). Similar to [62], we investigate the augmented Lagrangian of the MPC problem (8) with Levenberg-Marquardt regularization

$$\mathcal{L}(x_t) = f_t^{**} + V_{t+1}^{**} + \lambda^T h_t^{**} + \eta \|h_t^{**}\|^2 + \gamma \|x_t^{**} - x_t\|^2$$

to realize this kind of objective and take the constraints into account. Linearization at x_t^{**} (where the KKT conditions are fulfilled), ignoring constants and only considering the q part of x gives $\mathcal{L}(q_t) \approx \delta q_t^T H_t \delta q_t$ with $H_t = \nabla^2 \tilde{f}_t^{**} + \nabla^2 \tilde{V}_{t+1}^{**} + \eta \nabla \tilde{h}_t^{**} \big(\nabla \tilde{h}_t^{**}\big)^T + \gamma I$ where $\delta q_t = q_t - q_t^{**} = \Delta q_t - \Delta q_t^{**}$ is the deviation from the optimal transition, with which we define the Δq-loss as $L_{\Delta q} = \sum_{t=1}^{KT} \|\Delta q_t - \Delta q_t^{**}\|_{H_t}^2$. The role of $\nabla \tilde{h}_t^{**} \big(\nabla \tilde{h}_t^{**}\big)^T$ in H_t is to enforce the next state to satisfy the constraint while the other terms $\nabla^2 \tilde{f}_t^{**} + \nabla^2 \tilde{V}_{t+1}^{**}$ encode the information about which direction is more relevant for minimizing the cost-to-go. For stable training, we scale the Hessians with the parameters η and γ by their largest eigenvalues along the trajectory for each data sample. If an action sequence is infeasible, we set $L_{\Delta_q} = 0$.

2) Skeleton prediction loss: To enable Ω to detect the (in)feasibility of an action sequence, we train its z output with $L_z = (z - z^*)^2$ to predict the exponentiated cost.

3) Value function loss: The behavior cloning objective trains the energy function such that its natural gradient reproduces the target transition. In addition, we train also the value of the energy function with $L_v = \sum_{t=1}^{KT} \big(\bar{v}_{k(t-1)+1} + E_c(q_{t-1} + \Delta q_t^{**}) - v_{t+1}^{**}\big)^2$, where the joint state independent bias term $\bar{v}_{k(t-1)+1}$ is predicted by the reasoning network Ω.

VI. Experiments

A. Scene and Task

We consider the task of touching a goal location (one can think of pushing a button). As can be seen in Fig. 1, the goal location, indicated in red, can be out of reach for the robot to touch it directly. Therefore, the robot arms have to utilize the stick (green) to touch the goal. However, the stick might also only be graspable by one of the two arms, which means that a handover motion would be necessary. This problem is challenging since the way each individual action has to be executed is tightly coupled with other actions and the goal. For example, the stick has to be grasped at a certain location in order to touch the goal with it. If a handover is involved, then the first grasp often has to be completely different. All this depends on the geometry of the scene.

The LGP problem for data generation contains 6 actions (touch left arm, touch right arm, grasp left arm, grasp right arm, touch stick end 1, touch stick end 2). This leads to 10 different action sequences, 2 of length 1, 4 of length 2, and 4 of length 3. We generate 10,000 scenes by uniformly sampling the goal location, the position, orientation, and length of the stick, leading to a 6-dimensional parameter space for the scenes. Gaussian noise with standard deviation 0.03 is added to the robot joints during control generation. For evaluation, we sample both a validation and test dataset of 1,000 scenes each with the same sampling procedure, but a different random seed. While the data is generated in a kinematic only environment, for evaluating the network, we use the Bullet physics simulator. All results are obtained by training the network three times for 100 epochs. Then, based on the validation dataset, the best epoch of each training is chosen. The numbers in the results are then obtained by evaluating these best networks on the separate test dataset.

B. Results

1) Performance on all action sequences: In the first row of Tab. II, we evaluate the success rate of our framework on a hypothetical execution where all action sequences of a scene can be tested for success. For the test scenes, for 1048 skeletons of length 1, 1807 of length 2, and 1824 of length 3 a feasible solution was found by LGP (there

TABLE I: ACCURACY OF SKELETON FEASIBILITY PREDICTION [%]

	length K of skeleton			total
	1	2	3	
true feasible rate	98.8 ± 0.3	92.9 ± 0.6	94.1 ± 0.4	94.7 ± 0.3
true infeasible rate	96.4 ± 0.2	84.8 ± 0.5	87.1 ± 0.4	87.8 ± 0.3

TABLE II: ABLATION STUDY: SUCCESS RATES [%]

		length K of skeleton			total
		1	2	3	
no disturbance	all	99.9 ± 0.1	90.5 ± 0.5	90.3 ± 0.5	92.5 ± 0.3
	MSE loss	99.7 ± 0.2	90.3 ± 0.6	89.8 ± 0.7	92.2 ± 0.5
	no adaptation	99.9 ± 0.1	90.3 ± 0.7	87.3 ± 0.8	91.3 ± 0.3
	no recurrent Ω	99.3 ± 0.4	40.7 ± 8.8	15.9 ± 6.6	44.1 ± 5.6
with disturbance	all	99.9 ± 0.1	88.7 ± 0.9	84.9 ± 1.6	89.7 ± 1.0
	MSE loss	99.7 ± 0.3	87.3 ± 1.6	81.8 ± 1.0	87.9 ± 0.4
	no adaptation	99.9 ± 0.1	87.5 ± 1.0	81.3 ± 0.6	87.8 ± 0.3
	no recurrent Ω	99.4 ± 0.3	34.6 ± 7.9	10.9 ± 4.9	39.8 ± 4.8

are 10 theoretical skeletons per scene). Among those, our proposed network was successful 99.9% for skeletons of lengths 1, 90.5% for length 2, and 90.3% for length 3 without disturbance, leading to 92.5% in total. With disturbance (Gaussian noise with standard deviation of 0.03 at each time step), the rates are 99.9%, 88.7% and 84.9%, respectively (89.7% in total). We define success if all actions have been completed and the final distance of the robot or the stick to the goal is less than 10 cm. The mean distance to the goal was 1.6 ± 2.1 cm (1.9 ± 2.3 cm with disturbance), which is very precise given how sensitive the position of the end of a long stick is to small errors in the robot joint configuration.

2) Accuracy of action sequence prioritization: A key feature of our approach is that the network predicts if an action sequence is feasible and if yes, its expected cost. Tab. I shows the accuracy for using the prediction as a feasibility classifier. True (in)feasible rate means the percentage of (in)feasible action sequences as found by LGP in the test data which were correctly classified as (in)feasible by the network. In $97.0 \pm 0.4\%$ of all feasible scenes in the test data, the action sequence ranked highest by the network was actually feasible. Additionally, in $91.8 \pm 0.4\%$ of the cases, this highest-ranked sequence was indeed the one with the lowest cost as found by LGP. If our framework was not able to distinguish between feasible and infeasible skeletons, a random selection would lead to a success rate of $43.7 \pm 0.8\%$.

3) Performance on scenes: Evaluating the performance on a scene level, i.e. with the controller (6), and counting how often this leads to success, the success rate is $94.2 \pm 0.2\%$ ($93.0 \pm 0.1\%$ with disturbance). If we allow the method to restart the execution when it failed by going back to the initial robot configuration up to three times, the success rate increases to $95.5 \pm 0.6\%$ ($95.4 \pm 0.6\%$ with disturbance).

4) Switching Behavior: To demonstrate the switching capability of our framework, we consider the scene in Fig. 1. In this scene, four out of ten skeletons (see Fig. 3) are classified as feasible, and `graspR-touchS2`, which corresponds to the upper row of Fig. 1, is predicted to have the lowest cost-to-go. We inject artificial disturbances to obstruct this skeleton to be executed: the right arm is frozen and the left arm is pulled to the stick for the first time steps (grey area in Fig. 3). As shown in Fig. 3, while the robot is being hindered, the cost-to-go of each skeleton increases differently, making `graspL-graspR-touchS2` optimal afterwards, and our network executes it as shown in the lower row of Fig. 1.

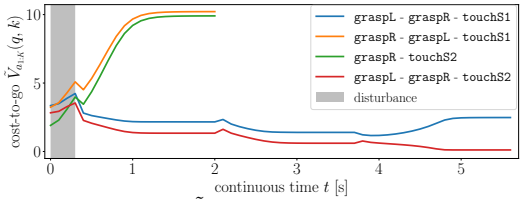

Fig. 3. Predicted cost-to-go $\tilde{V}$ of (initially) feasible action sequences over continuous time t, based on which the controller chooses a discrete action sequence to execute; transitions to next actions are at $t = 2$s and $t = 3.7$s.

C. Ablation Study

Since our framework consists of many parts, in this section we investigate the influence of each part on the performance.

1) Standard behavior cloning loss: In Sec. V-B we introduced a more informative loss function. As seen in Tab. II, the performance with the informative loss function leads to a slightly higher success rate on all (feasible) skeletons than with the standard behavior cloning loss (MSE loss). If there is disturbance, the difference is more significant.

2) Without adaptation: In Sec. IV-C a network that adapts the control parameter to observations at phase transition times was introduced. If we remove this part and directly use the predicted parameter from Ω, i.e. $c_k = \hat{c}_k$, one can see in Tab. II that especially for sequence length 3 the performance without adaptation drops from 84.9% to 81.3% (with disturbance). This can be explained by the fact that the longer the sequence the more imperfections/disturbances could accumulate and hence more adaptation is necessary.

3) No recurrent reasoning network: To support our claim that sequential manipulation tasks require joint reasoning over the sequence of actions and their parameters, we replace the bidirectional recurrent network for Ω by a feed-forward one. This way, the network cannot coordinate the parameters in a globally consistent way. As seen in Tab. II, with an overall success rate of only 39.8% on all skeletons (with disturbance), joint reasoning is very crucial for this task, especially for sequence length 3 (success rate of only 10.9%). For sequence length 1, i.e. direct touch, it has high success rate, which is expected since there are no other actions such that coordination between them would be necessary. Since the image of the scene contains both the location of the stick and the goal, without recurrent reasoning, the network can have some success (34.6%) for sequence length 2.

VII. CONCLUSION

In this work, we have presented a framework that offers the advantages of TAMP (joint reasoning, multiple solutions) for a sequential manipulation problem while relying only on sensor information and which provides controllers. One of the main arguments and what we have also shown empirically is that sequential manipulation tasks require joint reasoning over action parameters. While this is true for many tasks, there are also many long-horizon scenarios where subtasks are mostly independent. This implies that (learning) algorithms should be able to decide which parts need to be reasoned about jointly and which can be handled separately for efficient reasoning [63]. A limitation of this work is that the initial image has to contain all information required for planning the action sequence/continuous parameters.

REFERENCES

[1] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *International Journal on Robotics Research*, 2018.

[2] C. Garrett, L. Kaelbling, and T. Lozano-Perez, "Learning to rank for synthesizing planning heuristics," in *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2016.

[3] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *arXiv preprint arXiv:2010.01083*, 2020.

[4] M. Toussaint, J.-S. Ha, and D. Driess, "Describing physics for physical reasoning: Force-based sequential manipulation planning," *IEEE Robotics and Automation Letters*, 2020.

[5] D. Driess, J.-S. Ha, and M. Toussaint, "Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image," in *Proc. of Robotics: Science and Systems (R:SS)*, 2020.

[6] M. Toussaint, K. R. Allen, K. A. Smith, and J. B. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning." in *Robotics: Science and Systems*, vol. 2, 2018.

[7] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical planning in the now," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[8] L. de Silva, A. K. Pandey, M. Gharbi, and R. Alami, "Towards combining HTN planning and geometric task planning," *CoRR*, 2013.

[9] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. J. Russell, and P. 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.

[10] I. Rodriguez, K. Nottensteiner, D. Leidner, M. Kasecker, F. Stulp, and A. Albu-Schäffer, "Iteratively refined feasibility checks in robotic assembly sequence planning," *Robotics and Automation Letters*, 2019.

[11] M. 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*. AAAI Press, 2015, pp. 1930–1936. [Online]. Available: http://ijcai.org/Abstract/15/274

[12] T. Migimatsu and J. Bohg, "Object-centric task and motion planning in dynamic environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 844–851, 2020.

[13] Z. Zhao, Z. Zhou, M. Park, and Y. Zhao, "Sydebo: Symbolic-decision-embedded bilevel optimization for long-horizon manipulation in dynamic environments," *arXiv preprint arXiv:2010.11078*, 2020.

[14] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *Proc. of the Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014.

[15] 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.

[16] F. R. Hogan, E. R. Grau, and A. Rodriguez, "Reactive planar manipulation with convex hybrid MPC," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 247–253. [Online]. Available: https://doi.org/10.1109/ICRA.2018.8461175

[17] N. Doshi, F. R. Hogan, and A. Rodriguez, "Hybrid differential dynamic programming for planar manipulation primitive," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[18] T. Marcucci and R. Tedrake, "Warm start of mixed-integer programs for model predictive control of hybrid systems," *IEEE Transactions on Automatic Control*, 2020.

[19] J.-S. Ha, D. Driess, and M. Toussaint, "Probabilistic framework for constrained manipulations and task and motion planning under uncertainty," in *Proc. of the Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[20] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2154–2162.

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

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

[23] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks: Learning generalizable representations for visuomotor control," in *International Conference on Machine Learning (ICML)*, 2018, pp. 4739–4748. [Online]. Available: http://proceedings.mlr.press/v80/srinivas18b.html

[24] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," in *Advances in Neural Information Processing Systems*, 2018, pp. 8289–8300.

[25] A. Simeonov, Y. Du, B. Kim, F. R. Hogan, J. Tenenbaum, P. Agrawal, and A. Rodriguez, "A long horizon planning framework for manipulating rigid pointcloud objects," in *Conference on Robot Learning (CoRL)*, 2020.

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

[27] C. Paxton, Y. Barnoy, K. D. Katyal, R. Arora, and G. D. Hager, "Visual robot task planning," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8832–8838.

[28] F. Ebert, C. Finn, A. X. Lee, and S. Levine, "Self-supervised visual planning with temporal skip connections," in *Conference on Robot Learning*, 2017.

[29] L. Manuelli, Y. Li, P. Florence, and R. Tedrake, "Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning," *arXiv preprint arXiv:2009.05085*, 2020.

[30] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," in *International Conference on Learning Representations ICLR*, 2017.

[31] R. Pascanu, Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racanière, D. P. Reichert, T. Weber, D. Wierstra, and P. W. Battaglia, "Learning model-based planning from scratch," *CoRR*, vol. abs/1707.06170, 2017.

[32] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, *et al.*, "Imagination-augmented agents for deep reinforcement learning," in *Advances in neural information processing systems*, 2017.

[33] H. Suh and R. Tedrake, "The surprising effectiveness of linear models for visual foresight in object pile manipulation," *arXiv preprint arXiv:2002.09093*, 2020.

[34] S. Nair and C. Finn, "Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation," *arXiv preprint arXiv:1909.05829*, 2019.

[35] K. Pertsch, O. Rybkin, F. Ebert, C. Finn, D. Jayaraman, and S. Levine, "Long-horizon visual planning with goal-conditioned hierarchical predictors," *arXiv preprint arXiv:2006.13205*, 2020.

[36] S. Nasiriany, V. Pong, S. Lin, and S. Levine, "Planning with goal-conditioned policies," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 843–14 854.

[37] J.-S. Ha, Y.-J. Park, H.-J. Chae, S.-S. Park, and H.-L. Choi, "Adaptive path-integral autoencoders: Representation learning and planning for dynamical systems," in *Advances in Neural Information Processing Systems*, 2018, pp. 8927–8938.

[38] B. Kim, Z. Wang, L. P. Kaelbling, and T. Lozano-Pérez, "Learning to guide task and motion planning using score-space representation," *The International Journal of Robotics Research*, vol. 38, no. 7, pp. 793–812, 2019.

[39] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *Robotics and Automation Letters*, vol. 4, no. 2, pp. 1255–1262, 2019.

[40] D. Driess, O. Oguz, J.-S. Ha, and M. 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.

[41] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 447–454.

[42] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Active model learning and diverse action sampling for task and motion planning," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4107–4114.

[43] M. Burke, K. Subr, and S. Ramamoorthy, "Action sequencing using visual permutations," *arXiv preprint arXiv:2008.01156*, 2020.

[44] T. Silver, R. Chitnis, A. Curtis, J. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Planning with learned object importance in large problem instances using graph neural networks," *arXiv preprint arXiv:2009.05613*, 2020.

[45] K. Fang, Y. Zhu, A. Garg, S. Savarese, and L. Fei-Fei, "Dynamics learning with cascaded variational inference for multi-step manipulation," *arXiv preprint arXiv:1910.13395*, 2019.

[46] R. Chitnis, S. Tulsiani, S. Gupta, and A. Gupta, "Efficient bimanual manipulation using learned task schemas," in *IEEE International*

Conference on Robotics and Automation (ICRA), 2020, pp. 1149–1155.

[47] F. Xie, A. Chowdhury, M. Kaluza, L. Zhao, L. L. Wong, and R. Yu, "Deep imitation learning for bimanual robotic manipulation," *arXiv preprint arXiv:2010.05134*, 2020.

[48] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters, "Towards learning hierarchical skills for multi-phase manipulation tasks," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1503–1510.

[49] A. Bagaria and G. Konidaris, "Option discovery using deep skill chaining," in *International Conference on Learning Representations*, 2019.

[50] J.-S. Ha, Y.-J. Park, H.-J. Chae, S.-S. Park, and H.-L. Choi, "Distilling a hierarchical policy for planning and control via representation and reinforcement learning," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[51] K. Kase, C. Paxton, H. Mazhar, T. Ogata, and D. Fox, "Transferable task execution from pixels through deep planning domain learning," in *Proc. of the Int. Conf. on Robotics and Automation (ICRA)*, 2020.

[52] C. Paxton, N. Ratliff, C. Eppner, and D. Fox, "Representing robot task plans as robust logical-dynamical systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 5588–5595.

[53] S. Pirk, K. Hausman, A. Toshev, and M. Khansari, "Modeling long-horizon tasks as sequential interaction landscapes," *arXiv preprint arXiv:2006.04843*, 2020.

[54] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet, "Learning latent plans from play," in *Conference on Robot Learning*, 2019, pp. 1113–1132.

[55] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," in *Conference on Robot Learning*, 2020, pp. 1025–1037.

[56] B. Ichter, P. Sermanet, and C. Lynch, "Broadly-exploring, local-policy trees for long-horizon task planning," *arXiv preprint arXiv:2010.06491*, 2020.

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

[58] I. Mordatch and E. Todorov, "Combining the benefits of function approximation and trajectory optimization." in *Robotics: Science and Systems*, vol. 4, 2014.

[59] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[60] M. Tuscher, J. Hoerz, D. Driess, and M. Toussaint, "Deep 6-dof tracking of unknown objects for reactive grasping," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[61] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," *arXiv preprint arXiv:1801.02854*, 2018.

[62] J. Carius, F. Farshidian, and M. Hutter, "Mpc-net: A first principles guided policy search," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2897–2904, 2020.

[63] J. Ortiz-Haro, V. N. Hartmann, O. S. Oguz, and M. Toussaint, "Learning efficient constraint graph sampling for robotic sequential manipulation," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021.