Proceedings of Machine Learning Research vol 144:1–14, 2021

Self-Supervised Learning of Long-Horizon Manipulation Tasks with Finite-State Task Machines

Junchi Liang JL2068@CS.RUTGERS.EDU
Robotics Lab, 1 Spring Street, New Brunswick, NJ 08901

Abdeslam Boularias AB1544@CS.RUTGERS.EDU
Robotics Lab, 1 Spring Street, New Brunswick, NJ 08901

Abstract

We consider the problem of a robot learning to manipulate unknown objects while using them to perform a complex task that is composed of several sub-tasks. The robot receives 6D poses of the objects along with their semantic labels, and executes nonprehensile actions on them. The robot does not receive any feedback regarding the task until the end of an episode, where a binary reward indicates success or failure in performing the task. Moreover, certain attributes of objects cannot be always observed, so the robot needs to learn to remember pertinent past actions that it executed. We propose to solve this problem by simultaneously learning a low-level control policy and a high-level finite-state *task machine* that keeps track of the progress made by the robot in solving the various sub-tasks and guides the low-level policy. Several experiments in simulation clearly show that the proposed approach is efficient at solving complex robotic tasks without any supervision.

1. INTRODUCTION

The state of the art in robotic manipulation falls short of the capabilities that are needed for performing various tasks in open worlds. One of these critical capabilities is the use of tools in unstructured and unknown environments such as those encountered in households and small manufacturing workshops. A major source of failure in robotic manipulation is the high variety of the objects and their configurations and poses in such environments. Consequently, a robot needs to be autonomous and to adapt to changes. However, developing the software necessary for performing every single new task autonomously is costly and can be accomplished only by experienced robotics engineers. This is an issue that is severely limiting the popularization of robots today.

Ideally, robots should be multi-purpose, polyvalent and able to learn new skills in a self-supervised manner. For example, a robot in a factory setting should be able to learn on its own a new skill such as replacing a tire, from trial and error. The robot explores various random manipulation actions on different components of the tire, and receives a reward signal only when the tire is removed. Eventually, the robot learns the causal link between loosening each lug nut of the wheel and the long-term effect of receiving a reward at the end of the trial. The robot should not simply memorize the successful sequence of controls, but it should learn a general policy that maps every possible new image into a low-level action in a closed-loop control.

Model-based reinforcement learning, in conjunction with manipulation planning, has shown promise in generalizing learned skills to new setups. However, most existing works in this area assume that the state of the manipulated objects is fully observable. This is rarely the case in robotics. In our previous example, for instance, the robot cannot easily observe if the lug nuts have been loosened. The robot can however learn to remember pertinent past controls/feedbacks to overcome the partial observability problem. For example, *LSTM* and *GRU* architectures are general-purpose tools for solving problems of partial observability by discovering and remembering pertinent information. They tend, however, to require exorbitant amounts of training data.

To address these issues, we propose in this work[1] to structure the memory of the robot as a *Finite-State Machine* (FSM). The finite states of the machine are auxiliary variables that are used in conjunction with the observations as inputs to the control policy. Intuitively, each state corresponds to a specific subtask, or stage. For example, state A corresponds to turning on a switch, state B corresponds to loosening a lug nut, and so on. The FSM transitions from one state to another whenever a subtask is successfully accomplished, and transitions to a terminal state when the full task is correctly performed. The robot does not know in advance the number of the FSM states nor their interpretation as subtasks. No prior information about the task is provided. The only inputs given to the robot at each time-step are its joint angles, the 6D poses of the objects, and their semantic labels, such as "lug nut", "paint container", "tire", and so on. Semantic labels are necessary to generalize across scenes. The robot applies changes to its joint angles at each time-step. The robot does not receive any external signal except a binary reward at the end of the executed trajectory that indicates if the full task was performed successfully or not. Based on that alone, the robot learns the finite-state machine of the desired task and a policy for performing the task.

2. PRELIMINARIES

A *Markov Decision Process* (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is a set of states and $\mathcal{A}$ is a set of actions. T is a transition function with $T(s'|a,s) = P(S_{t+1} = s'|s_t = s, A_t = a)$ for $s, s' \in \mathcal{S}, a \in \mathcal{A}$. R is a reward function where $r_t = R(s_t) \in \mathbb{R}$ is the reward received in s_t, and $\gamma \in [0,1[$ is a discount factor. A policy π_θ is a distribution on the action to be executed in each state, defined as $\pi_\theta(s,a) = P(A_t = a|S_t = s)$. The value V^{π_θ} of a policy π_θ is the expected sum of rewards that will be received if π_θ is followed, i.e., $V^{\pi_\theta}(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | S_0 = s, \pi_\theta, T, R]$.

In several application domains, such as robotics, states are not fully observable in general. A robot perceives partial observations o_t, in the form of images for example. The resulting process is a Partially Observable MDP (POMDP). Formally, a POMDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma)$ where $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ is an MDP, $\mathcal{O}$ is a set of observations, and Z is an observation likelihood function.

3. RELATED WORK

POMDPs are traditionally learned by using the expectation-maximization technique known as the *Baum-Welch* algorithm (Rabiner, 1990; Kontorovich et al., 2013). This algorithm is however sensitive to the initial values of transition and observation probabilities, it typically gets stuck in local maxima, and requires the number of hidden variables to be known in advance. Predictive State Representations (PSRs) (Singh et al., 2004; Boularias and Chaib-Draa, 2009) are an alternative representation that can be learned from observations without reasoning about hidden variables. PSRs are however sensitive to certain parameters, such as the number of *core tests*. The learning complexity of PSRs is also exponential in the length of history that needs to be remembered to predict future observations. This problem was later alleviated with spectral techniques (Boots et al., 2011)

1. This work is supported by NSF awards IIS-1734492 and IIS-1846043.

which generalize PSRs by including features of trajectories, instead of a stream of raw observations. In (Icarte et al., 2018), a type of finite-state machine that supports the specification of reward functions was presented and used to accelerate reinforcement learning of structured policies. In contrast to our proposed approach, the structure of the reward machine in (Icarte et al., 2018) was assumed to be known. Several techniques for learning partially observable dynamical models are based on recurrent neural networks (RNN), and LSTM in particular (Downey et al., 2017; Choromanski et al., 2018; Hafner et al., 2018; Finn and Levine, 2017; Finn et al., 2016). LSTM typically requires large numbers of training data and often fails to capture time-delayed causal relations. To solve this problem, long-term dependencies in temporal models were considered in some recent works (Neitz et al., 2018; Trinh et al., 2018). Such dependencies are learned, for example, by using the reconstruction loss in recurrent neural nets as an auxiliary objective (Trinh et al., 2018).

The problem of learning long-term dependencies is also addressed with *attention mechanisms*, which are used for selecting specific features dynamically according to the specified task. Attention weights were defined in (Xu et al., 2015; Jiang et al., 2018; Anderson et al., 2018) as functions of features of different parts of an image and memory units that allow the agent to focus on pertinent regions of the image as it generates a corresponding caption. The same mechanism was adopted in a more recent work (Jiang et al., 2018). Duan et al. (2017) employed an attention mechanism to compress information from demonstrated trajectories in the context of imitation learning. LSTM also employs attention mechanisms since the forget and input gate can be interpreted as attention weights (Duan et al., 2017). We show in our experiments that LSTM's attention tends to forget old events, unless colossal numbers of training trajectories are used.

While we assume in the current work that 6D poses and labels of objects are provided from a vision module, other recent works have shown that complex tasks can be completed by learning directly from pixels (Kalashnikov et al., 2018; Fox et al., 2018; Xu et al., 2017; Huang et al., 2018; Nair et al., 2020; Andrychowicz et al., 2017; Nair and Finn, 2019). This objective is typically accomplished by using compositional policy structures that are learned by imitation (Kalashnikov et al., 2018; Fox et al., 2018), or that are manually specified (Xu et al., 2017; Huang et al., 2018). Some of these methods have been used for simulated control tasks (Bacon et al., 2017; Nachum et al., 2018; Eysenbach et al., 2019). These promising end-to-end techniques still require orders of magnitude more training trajectories compared to methods like ours that separate the object detection and planning problems. Long-horizon manipulation tasks have also been solved by using symbolic representations and Task and Motion Planning (TAMP) (Toussaint et al., 2019; Kaelbling, 1993; Kaelbling and Lozano-Pérez, 2010). However, all the variables of the reward function in these works are assumed to be known and fully observable. In contrast, our method is *fully self-supervised*, with no intermediate rewards and signals besides the configuration of the robot, the 6D poses of the objects, and a binary reward at the end of a trajectory that indicates success or failure.

4. APPROACH

4.1. Finite-State Task Machines

We focus in this work on a special type of object-oriented POMDPs that is appropriate to robotic manipulation tasks. A state S is described by visible attributes of the objects in the scene and the robot, in addition to task states that are hidden and unknown a priori. Specifically, we denote the configuration of the robot in a given world coordinate frame at time t by $c_t \in \left(\mathbb{R}^3 \times \mathbb{SO}(3)\right)^J$, where J is the number of joints of the robotic manipulator and end-effector. We assume that the manipulated objects are rigid, the 6D pose (position and orientation) of the end-effector at time t in the reference frame of object i is denoted by $p_t^i \in \mathbb{R}^3 \times \mathbb{SO}(3)$, and a semantic label $l_t^i \in \mathcal{L}$ for object i is obtained from a vision module. At each time-step, the robot receives an observation $o_t = (c_t, \langle p_t^1, l_t^1 \rangle, \dots, \langle p_t^n, l_t^n \rangle)$ wherein n is the total number of objects that are present in the scene.

For the sake of simplicity and without loss of generality, we assume that the end-effector is a tool (e.g., painting brush, wrench, suction cup, etc.) that is already grasped by the robot. Therefore, manipulation actions are nonprehensile and can be performed by controlling the 6D pose of the end-effector relative to the objects in the scene. A manipulation action is defined as $a_t = \langle i, \Delta p_t^i \rangle$, where $i \in \{1, \dots, n\}$ is a manipulated object, and $\Delta p_t^i \in \mathbb{R}^3 \times \mathbb{SO}(3)$ is a desired change in the pose p_t^i of the end-effector in object i's coordinates system. Each object i has a fixed anchor (*home*) 6D pose $c^i \in \mathbb{R}^3 \times \mathbb{SO}(3)$ that the robot's end-effector moves to before starting to manipulate it. Anchor points are chosen arbitrarily, and slightly away from the corresponding objects to avoid collisions before manipulation starts.

When two consecutive actions $a_{t-1} = \langle i, \Delta p_{t-1}^i \rangle$ and $a_t = \langle j, \Delta p_t^j \rangle$ aim to manipulate two different objects, i.e. $i \neq j$, the RRT motion planner (Kuffner and Lavalle, 2000) is used to move the robot from its last configuration c_{t-1} to a configuration c_t that places the end-effector at the anchor pose c^j of object j, while avoiding collisions. Once the end-effector is in the anchor pose, subsequent moves $\{\Delta p_{t+k}^j\}_{k=0}^K$ of the end-effector relative to the object's frame of reference are executed by using a PID controller that connects way-points $\{c^j + \sum_{k=0}^K \Delta p_{t+k}^j\}_{k=0}^K$. Note that the number of moves K is not constant. This process is repeated until the robot switches to manipulating a different object. The accuracy of the PID controller is reflected by the transition model of the observed part of the state, denoted by $T^o(o_t, a_t, o_{t+1})$. Transition function $T^o(o_t, a_t, o_{t+1})$ is the probability of observing o_{t+1} at time $t+1$ after observing o_t at time t and executing action a_t. We define this probability as a Gaussian distribution, where $p_{t+1}^i = p_t^i + \Delta p_t^i + \epsilon$ for action $a_t = \langle i, \Delta p_t^i \rangle$, with $\epsilon \sim \mathcal{N}(0, \Sigma)$. The other attributes of o_{t+1} are computed based on p_{t+1}^i. The semantic labels of the objects remain constant. Noise covariance Σ is estimated from a small number of data points.

Unlike the poses of the robot and the objects, task states are not observable, and their number is unknown in advance. The robot should infer these abstract states from raw trajectories of actions, observed poses of objects and terminal binary rewards. Task states are *internal*, and thus do not play any role in the transition function T^o of the poses. We denote the set of task states by $\mathcal{G}$. Transition function $T^g(g_t, o_t, g_{t+1})$ is the probability of transiting from task state g_t to task state g_{t+1} at time $t+1$ after observing o_t at time t. A terminal task state g^* is reached when the task is successfully accomplished, where $T^g(g^*, o, g^*) = 1, \forall o$ and $T^g(g^*, o, g) = 0, \forall o, g \neq g^*$.

The state of the system at time t is defined as $s_t = \langle o_t, g_t \rangle \in \mathcal{O} \times \mathcal{G}$. Only o_t is observed by the robot. The transition function is given as $T(s_t, a_t, s_{t+1}) = T^o(o_t, a_t, o_{t+1}) T^g(g_t, o_t, g_{t+1})$. Finally, we assume that the rewards for manipulation tasks are binary. Reward function R is defined as $R(\langle o_t, g_t \rangle) = 1$ if $g_t = g^*$ and $R(\langle o_t, g_t \rangle) = 0$ if $g_t \neq g^*$. Consequently, rewards indicate only success or failure of a manipulation task. Discount factor $\gamma \in [0, 1[$ ensures that shorter trajectories are preferred over longer ones. Therefore, the manipulation problem consists in finding a policy π_θ^* that satisfies $\pi_\theta^* = \arg\max_{\pi_\theta \in \Pi} V^{\pi_\theta}(s), \forall s \in \mathcal{S}$, given a set $\mathcal{D} = \{\tau^m\}_{m=1}^M$ of data trajectories $\tau^m = (o_0^m, a_0^m, r_0^m, \dots, o_h^m, a_h^m, r_h^m)$.

4.2. Model

Policy Network. We present here a probabilistic model for simultaneously learning a finite-state task machine and a policy π_θ that maximize V^{π_θ} with self-supervision and no human input beyond terminal rewards. The policy model π_θ is implemented as a neural network that estimates the probability of selecting an action $a_t = \langle i, \Delta p_t^i \rangle$ for any state $s_t = \langle o_t, g_t \rangle$. Specifically, the network returns a distribution on $i \in \{1, \dots, n\}$ that indicates which object needs to be manipulated next, based on the current state $s_t = \langle o_t, g_t \rangle$. The network also returns the mean and variance of a Gaussian distribution based on the current s_t and the previous one s_{t-1}, such that $\Delta p_t^i \sim \mathcal{N}\big(\mu_i(s_{t-1}, s_t; \theta), \Sigma_i(s_{t-1}, s_t, \theta)\big)$ where θ is the set of weights of the neural network. We found from our empirical investigation that the addition of history, in the form of s_{t-1}, helps capturing the direction of the motion of the end-effector and yields better results, despite the fact that the task-state g_t part of state s_t is already a memory of past actions and observations.

Temporal Modulation. Temporal modulation is needed for controlling the velocities of the movements. We utilize a phase variable ϕ_t to provide a time signal to policy π_θ. Therefore, we redefine policy π_θ as $\pi_\theta(s, a, \phi) = P(a_t = a | s_t = s, \phi = \phi_t)$. But for ease of notation, we drop ϕ from the inputs of π_θ in this article. Policy π_θ is then a time-dependent function where time is given as an input implicitly through phase signal ϕ. Phase variables are typically used in dynamic motor primitive in order to generate faster or slower motions (Paraschos et al., 2013). They are also used to generate rhythmic and stroke-based movements. In this work, we define phase variables as $\phi_t = \sum_{i=1}^{n} \omega_i \exp\big(\frac{(t - \hat{t}_{(g_t,0)} - \hat{t}_{(g_t,i)})^2}{h}\big)$, where $(\hat{t}_{(g_t,0)}, \hat{t}_{(g_t,1)}, \dots)$ is a sequence of time-steps that is specific to the movement associated with the current task state g_t, and $\{\omega_i\}$ are their corresponding weights. Both time-steps $\{\hat{t}_{(g_t,i)}\}_{i=1}^{n}$ and weights $\{\omega_i\}_{i=1}^{n}$ are variables, and learned along the other weights in θ of the policy network π_θ. Starting time $\hat{t}_{(g_t,0)}$ is obtained during the execution as the time when the movement associated with the current task state g_t has started.

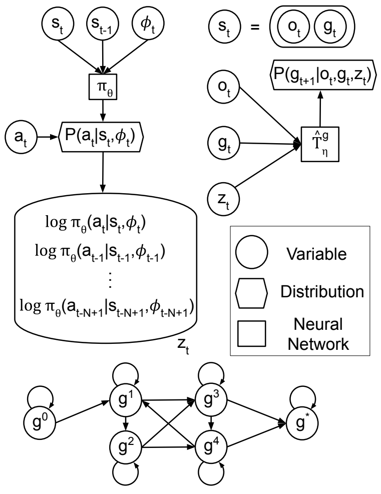


Figure 1: Proposed model

Finite-state Machine Network. A second neural network is used to predict transitions between task states. Since task states are discrete and finite, the neural network returns a transitions matrix $\hat{T}_\eta^g$, where $\hat{T}_\eta^g(g_t, o_t, g_{t+1}, z_t)$ is the probability of switching from g_t to g_{t+1} and η is the set of weights of the neural network. $z_t = [\log \pi_\theta(s_t, a_t), \log \pi_\theta(s_{t-1}, a_{t-1}), \dots, \log \pi_\theta(s_{t-N+1}, a_{t-N+1})]$ is a vector that contains the log-probabilities of the last N executed actions according to the policy model π_θ explained above. These probabilities are computed based on the means and variances returned by the policy network at the corresponding time-steps. z_t can be interpreted as an indicator of the progress in executing a specific sub-task associated with task state g_t. For example, z_t tracks the actions executed by the robot as it dips a brush in a paint bucket, and g_t is interpreted as the dipping sub-task. Note that the number of task states as well as their interpretation as sub-tasks, or primitive skills such as dipping or painting, is unknown to the robot. Progress vectors z_t are introduced to reduce the size of the set of task states $\mathcal{G}$, so that the system can remain in a self-loop at a task state $g_t \in \mathcal{G}$ for a few time-steps until z_t indicates that the underlying sub-task has been successfully accomplished and the system then switches to a different task state. If we only provide g_t and o_t to

```
Input: A POMDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, \gamma)$ formulation of the manipulation task, set $\mathcal{G}$ of task states, a learned task transition function $\hat{T}^g_\eta$, and a learned policy $\pi_\theta$ ;
Output: A sequence of actions $(a_0, a_1, \dots, a_t)$ to be executed by the robot;
for $i := 1; i \leq max\_nb\_trajectories; i \leftarrow i + 1$ do
    Set $o_0$ to the initial configuration of the robot and $g_0$ to initial task state $g^0$; $w[i] \leftarrow 1$; $r_0 \leftarrow 0$ ;    // $w[i]$ is the probability of sampled trajectory $i$, and $r_t$ is the reward received at time-step $t$.
    for $t := 0; (t < max\_horizon \wedge r_t < 1); t \leftarrow t + 1$ do
        $s_t \leftarrow \langle o_t, g_t \rangle$; Sample $a_t = (j, \Delta p^j) \sim \pi_\theta(s_t, .)$ ;    // Sample an action $a_t$ with probability $\pi_\theta(s_t, a_t)$.
        $z_t[t \bmod N] = \log \pi_\theta(s_t, a_t)$ ;    // Insert the log-probability of the sampled action into memory vector $z_t$.
        Sample $g_{t+1} \sim \hat{T}^g_\eta(g_t, o_t, z_t, .)$ ;    // Sample next task state with probability $\hat{T}^g_\eta(g_t, o_t, z_t, g_{t+1})$.
        Sample $o_{t+1} \sim \hat{T}^o(o_t, a_t, .)$ ;    // Sample the next observation conditioned on the simulated action $a_t$ and previous observation $o_t$, and using transition function $\hat{T}^o(o_t, a_t, o_{t+1})$.
        $w[i] \leftarrow w[i]\pi_\theta(a_t, s_t)\hat{T}^g_\eta(g_t, o_t, z_t, g_{t+1})\hat{T}^o(o_t, a_t, o_{t+1})$ ;    // Update the probability of the trajectory.
        if $g_{t+1} \neq g^*$ then $r_{t+1} \leftarrow 0$; else $r_{t+1} \leftarrow 1$ ;    // Successful trajectory obtained.
    end
    $\mathcal{T}[i] \leftarrow (a_0, a_1, \dots, a_t)$; $h[i] \leftarrow t$ ;    // Save the length of trajectory $i$.
end
Select $\tau \leftarrow \mathcal{T}[i]$ by sampling index $i$ from distribution $P(i) = \frac{\gamma^{h[i]} w[i]}{\sum_j \gamma^{h[j]} w[j]}$ where $i \in \{1, \dots, max\_nb\_trajectories\}$;
```

Algorithm 1: Inference

the transition model without memory z_t, then the number of task states needed to explain the binary reward received at the end of a trajectory increases significantly.

We will show in Section 4.4 a simple algorithm that learns the number of task states, a transition model $\hat{T}^g_\eta$ and a policy π_θ. The proposed algorithm iterates between learning these models and planning to actively sample new trajectories from the learned models. Therefore, we start by first explaining in the following section the planning procedure.

4.3. Inference

Algorithm 1 receives as inputs a POMDP model $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, \gamma)$ without a reward function, a finite-state task machine, defined by a set $\mathcal{G}$ and learned transition function $\hat{T}^g_\eta$, in addition to a learned policy π_θ. The algorithm returns a sequence of actions to be executed on the robot.

The algorithm samples in simulation a large number of trajectories of states s_t and actions a_t. It then computes a probability distribution on the sampled trajectories that are predicted to be successful at solving the task according to the learned task machine. The probability of a sampled trajectory that is predicted to be successful is proportional to its length, so that shorter trajectories are preferred. At the end, the algorithm returns one trajectory sampled from this distribution.

Specifically, we start by sampling an action $a_t = (i, \Delta p^i_t)$ for time-step t from distribution $\pi_\theta(s_t, .)$. The log-probability of the sampled action is inserted into the N-sized memory vector z_t, which keeps track of the last N selected actions. To simulate next state s_{t+1}, which is defined as $s_{t+1} = \langle o_{t+1}, g_{t+1} \rangle$, we first sample a next task state g_{t+1} with probability $\hat{T}^g_\eta(g_t, o_t, z_t, g_{t+1})$. We then sample next physical state o_{t+1}, which is defined as $o_{t+1} = (c_{t+1}, \langle p^1_{t+1}, l^1_{t+1} \rangle, \dots, \langle p^n_{t+1}, l^n_{t+1} \rangle)$ wherein n is the total number of objects that are present in the scene. Semantic labels do not change over time, i.e $l^i_{t+1} = l^i_t$. To obtain o_{t+1}, we first sample a new 6D pose $p^i_{t+1} \sim \mathcal{N}(p^i_t + \Delta p^i_t, \Sigma)$ where i is the index of the manipulated object. The new configuration c_{t+1} of the robot and the 6D poses of the other objects relative to the end-effector are all computed from p^i_{t+1}. The reward r_{t+1} is defined as 1 if the task state g_{t+1} is the terminal success state g^*, and 0 otherwise.

4.4. Learning

Algorithm 2 provides detailed steps of the proposed approach for learning the finite-state task machine $(\mathcal{G}, \hat{T}^g_\eta)$ and policy π_θ used by the planning algorithm explained above. In a nutshell, the robot explores its environment by manipulating randomly selected objects with random movements. The process is fully self-supervised, but the robot receives a reward of 1 when a trajectory of states and actions ends up with successfully performing the required task. The robot receives a reward of 0 for all other time-steps. The extreme sparsity of the reward signal makes this process particularly long. After accidentally discovering a successful sequence τ of actions that solve the task, the robot enters a second phase of learning. In the second phase, the robot tries to locally improve trajectory τ by exploring new actions that are not too distant from the actions in τ. At the same time, the algorithm searches for the most compact task machine $(\mathcal{G}, \hat{T}^g_\eta)$. Initially, $|\mathcal{G}|$ is set to be equal to the number of objects that were manipulated in the first successful trajectory τ, wherein each task state g^i corresponds to manipulating a specific object i. The robot then experiments with skipping various objects, and eliminating their corresponding task states from $\mathcal{G}$. Since the outcome of executing a trajectory is not deterministic, we formulate the problem of learning the states of $\mathcal{G}$ as a multi-armed bandit problem and utilize the *Upper Confidence Bound* (UCB) (Auer, 2003) technique to solve it efficiently. This process is explained in the following.

Set of task states $\mathcal{G}$ is initially set as $\{g^0, g^1, \ldots, g^n, g^*\}$, wherein n is the maximum number of objects, and g^* is an abstract terminal state. Initial policy π_θ is set such that only object i can be manipulated in task state g^i. When the task machine transitions into state g^i, it remains there with a large probability $1-\epsilon$, and switches to any other state (including the terminal state) with probability ϵ. These probabilities, used for initial exploration, are updated later in the learning process. The robot executes initial policy π_θ using initial task transition function $\hat{T}^g_\eta$ until a successful trajectory τ is encountered. Before starting the improvement phase, policy π_θ and transition $\hat{T}^g_\eta$ are trained to "imitate" trajectory τ. To this end, the neural networks corresponding to π_θ and $\hat{T}^g_\eta$ are trained to maximize the likelihood of trajectory τ.

The second phase consists in improving successful trajectory τ by exploring new actions that reduce the size of task states $\mathcal{G}$, and also shorten the overall length of the trajectory. The robot performs experiments of the type "what if I re-execute all the actions of trajectory τ except those related to object i?". Since the outcome of the open-loop execution of the actions is stochastic, one cannot immediately conclude that a task state cannot be skipped based on a single failed outcome. At a high-level, this is an $(n+1)$-armed bandit problem, where the $(n+1)$ arms correspond to the options of skipping one of the n current states of the task machine, or not skipping any state $(i = 0)$. We use the UCB technique in Algorithm 2 where $V[i]$ is the empirical average reward of the experiments that skip task state g^i, and $C[i]$ is the number of such experiments. Before experimenting with skipping a task state g^j, the task-machine is temporarily modified by changing its transition probabilities. Probabilities of transitioning to skipped state g^j from any state are all set to 0. Probabilities of transitioning from skipped state g^j to any other state g^k are re-distributed among all other states g^i after multiplying them with the probabilities of transitioning from g^i to skipped state g^j. Using the modified task machine T^g, and current policy π_θ, the robot samples and executes a new sequence of actions and obtains a new trajectory $\tau = (o_0, a_0, r_0, \ldots, o_h, a_h, r_h)$. The empirical value of the option of skipping task state g^j is updated based on the received terminal reward r_h. If the new trajectory is unsuccessful (i.e., $r_h = 0$), then one cannot yet conclude that task state g^j is unnecessary because the failure could be due to a small noise in the execution or perception. The confidence in the utility of task state g^j, given by the empirical average $V[j]$ and counter $C[j]$, is however decreased in this case. If the new trajectory is successful (i.e., $r_h = 1$), then one can conclude immediately that task state g^j can be eliminated from set $\mathcal{G}$. In this case, the neural network that corresponds to transition function $\hat{T}^g_\eta$ is re-trained to maximize the likelihood of

```
Input: List of objects $i \in \{1,\dots,n\}$;
Output: Set $\mathcal{G}$ of task states, task transition function $\hat{T}^g_\eta$, and policy $\pi_\theta$;
Initialize policy network $\pi_\theta$ with random weights $\theta$ ;   // Uniform initial exploration policy
$\mathcal{G} \leftarrow \{g^0, g^1, \dots, g^n, g^*\}$ ;   // Initial task states: one state per object, an initial sate $g^0$, and a terminal state $g^*$
For all $i \in \{1,\dots,n\}$, set $\pi_\theta(\langle ., g^j \rangle, \langle i, . \rangle) \leftarrow 0$ for all $j \neq i$ ;   // Only object $i$ can be manipulated in task state $g^i$
For all $i \in \{1,\dots,n\}$, set $\hat{T}^g_\eta(g^i, ., ., g^i) \leftarrow 1-\epsilon$ and $\hat{T}^g_\eta(g^i, ., ., g^j) \leftarrow \frac{\epsilon}{n}$ for all $j \neq i$ ;   // Initial task state transition probabilities
  do not depend on observations or memory variables.
repeat
    Set $o_0$ to the initial configuration of the robot and $g_0$ to initial task state $g^0$;
    Sample a trajectory $\tau = (o_0, a_0, o_1, a_1 \dots, o_h, a_h, r_h)$ by executing policy $\pi_\theta$ with task transition function $\hat{T}^g_\eta$;
    /* Stop when the first successful trajectory is obtained   */
until $r_h = 1$;
$\theta \leftarrow \arg\max_\theta \sum_{t=0}^{h} \log \pi_\theta(s_t, a_t)$ ;   // Train the policy network by maximizing the likelihood of the actions $a_t$ in the first
  successful trajectory $\tau$
$\eta \leftarrow \arg\max_\eta \sum_{t=0}^{h} \log \hat{T}^g_\eta(g_t, o_t, z_t, g_{t+1})$ ;   // Train the task transition function by maximizing the likelihood of the task
  states $g_t$ in the first successful trajectory $\tau$
For all $i \in \{0,\dots,n\}$, initialize $V[i] \leftarrow 0$ and $C[i] \leftarrow 0$; /* $V[i]$ is the expected return of policies that skip task state $g^i$, $C[i]$
    counts trajectories sampled from such policies. $i = 0$ indicates that no task state is skipped.   */
repeat
    $T^g \leftarrow \hat{T}^g_\eta$ ;   // Copying the transition function of the current task-state machine
    $j \leftarrow \arg\max_{i\in\{0,\dots,n\}} V[i] + c\sqrt{\frac{\ln(\sum_{k=0}^{n} C[k])}{C[i]+1}}$; $C[j] \leftarrow C[j]+1$ ;   // UCB for selecting a task state to skip
    if $j \neq 0$ then
        for $i := 1; i \leq n; i \leftarrow i+1$ do
            $T^g(g^i, ., ., g^j) = 0$ ;   // To skip task state $g^j$ in the current experiment
            For all $g^k \in \mathcal{G} - \{g^j\}$: $T^g(g^i, ., ., g^k) \leftarrow T^g(g^i, ., ., g^k) + T^g(g^i, ., ., g^j) T^g(g^j, ., ., g^k)$ ;   // Transitions into state $g^j$
              are rerouted toward following states $g^k$ (including the terminal state $g^*$).
        end
    end
    Set $o_0$ to the initial configuration of the robot and $g_0$ to initial task state $g^0$;
    Sample a trajectory $\tau = (o_0, a_0, o_1, a_1 \dots, o_h, a_h, r_h)$ using $\pi_\theta$ and $T^g$ ;   // A trajectory that skips state $g^j$.
    $V[j] \leftarrow \frac{(C[j]-1)V[j]+r_h}{C[j]}$;
    if $r_h = 1$ then
        $\mathcal{G} \leftarrow \mathcal{G} - \{g^j\}$ ;   // The trajectory was successful, so $g^j$ is unnecessary and can be removed
        $\eta \leftarrow \arg\max_\eta \sum_{t=0}^{h} \log \hat{T}^g_\eta(g_t, o_t, z_t, g_{t+1})$ ;   // Train the task transition function by maximizing the likelihood of
          the task states $g_t$ in the latest successful trajectory $\tau$
        $\theta \leftarrow \theta + \alpha \sum_{t=0}^{h} \gamma^{h-t} r_h \nabla_\theta \log \pi_\theta(s_t, a_t)$ ;   // Policy gradient using the actions $a_t$ of trajectory $\tau$.
    end
until $timeout$;
```

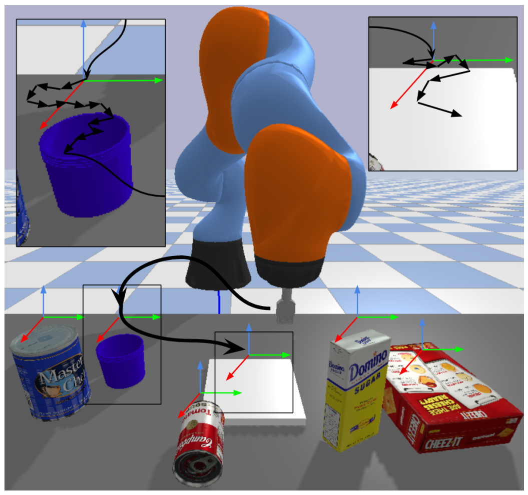
Algorithm 2: Learning

the task states in the new trajectory τ. Parameters θ of the policy network π_θ are also updated based on the reward received at the end of τ and the length h of τ, by using the policy gradient approach. This process is repeated. The task machine converges to a compact set of states after cutting off all unnecessary intermediate states. The policy, steered by the task machine, also converges to choosing the shortest sequence of moves for manipulating each object thanks to the policy gradient updates.

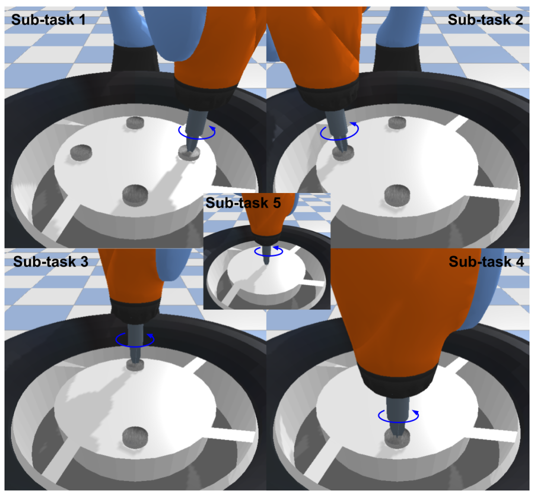
5. EXPERIMENTS

More details and videos of the experiments are available at https://rb.gy/z5a3hc.

Tasks. We thoroughly evaluated the proposed framework and algorithm on two long-horizon manipulation tasks illustrated in Figure 2, using the realistic *Pybullet* simulator of a *Kuka LBR* robot. The first one is a painting experiment. A reward is received at the end of a trajectory if the robot first successfully dips a brush attached to its end-effector in an object labeled as “paint bucket”, and then successfully paints a straight stroke on another object labeled as “canvas”. The entire simulation is physically realistic, except for the fluid (paint liquid) simulation which is simplified. The robot needs to avoid collisions while manipulating the objects. The dipping maneuver is considered as effective if and only if the brush touches the bottom of the paint bucket for more



(*a*)



(*b*)

Figure 2: Tasks considered in the experiments. (a) *Learning to paint*. (b) *Learning to remove a tire.*

than three time-steps, which results in loading the paintbrush with sufficient paint. The painting is successful if the loaded brush is moved along the surface of the canvas in a straight line. In addition to the objects labeled as "paint bucket" and "canvas", there are four other *irrelevant* objects on the table that have other labels. The robot does not know anything about the task, and does not know which types of objects should be manipulated, in which order or how to manipulate them. The four *distracting* objects make the learning more challenging because the robot will explore all of them before eventually learning a task state machine that indicates the types of objects that are relevant and the manipulation order, in addition to a policy for the low-level motor primitives.

The painting task involves only two sub-tasks (loading the brush, and stroking). To test the proposed algorithm on problems with more sub-tasks, we designed a second task where the robot learns to remove a wheel. A wrench is already attached to the end-effector. The task consists in placing the wrench on every lug nut to loosen it before moving to the center of the wheel to pull it. The robot transits to the next sub-task by loosening a lug nut: the end-effector should rotate counterclockwise more than $30°$ along the z-axis on a lug nut. The wheel can be taken off only after all lug nuts are loose and the end-effector is placed no more than 1cm from the center of the wheel. Here again, a reward of $+1$ is given when the task is successfully finished, all other states have a reward of 0. Results are averaged over five different positions of the nut lugs. We consider two types of wheels, those with two lug nuts (three sub-tasks in total) and three distracting objects (rubber and two fixed pieces), and those with four lug nuts (five sub-tasks) and one distracting object (rubber).

Compared Methods. We compare the proposed algorithm with the model-free RL algorithms Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Advantage Actor Critic (A2C) (Mnih et al., 2016), both with an LSTM unit. We also compare with the model-based approach of (Oh et al., 2015) where a neural network is trained to predict terminal rewards from full trajectories of data, using LSTM in one variant and GRU (Wahlström et al., 2015) in another. We also test PPO and A2C in an *assisted setup*, where extra intermediate rewards of $+1$ are given to the robot after finishing each sub-task. All compared methods use the same actions and observations defined in Section 4.1.

Results. The results of the experiments are averaged over 50 independent test episodes and five different initial positions of the objects in the scene. Figure 3 (top) shows the task success rates in

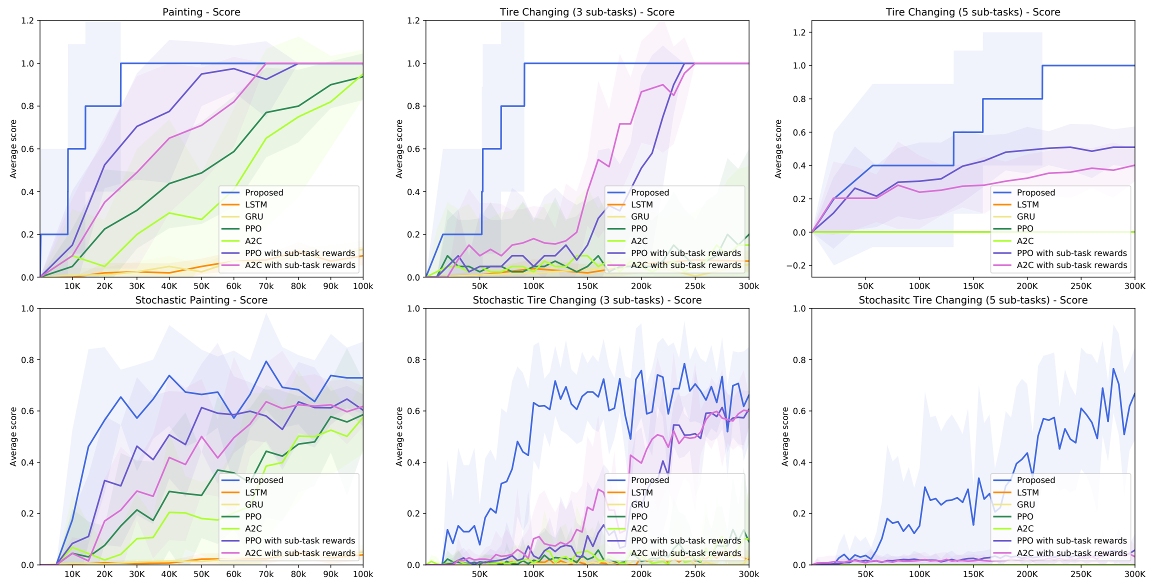


Figure 3: Average reward per test episode as a function of the number of time-steps in training.

a setting where the robot's end-effector moves deterministically. Our algorithm converges to 100% success rate after a relatively short initial exploration phase. Figure 3 (bottom) shows the average value functions of the learned policies when a Gaussian noise of standard deviation $0.5cm$ is added to the motions of the end-effector and the poses of the objects. Other algorithms fail to learn the same tasks, mainly because of the sparsity of reward signals. When the robot receives intermediate sub-task rewards in the assisted setting for PPO and A2C, we notice that it learns to perform two of these tasks, although it still needed significantly more training data. Notice also how the planners using LSTM and GRU needed more training data, which indicates that our finite-state machine is potentially a better memory structure for long-horizon manipulation planning.

6. CONCLUSION

The ability to plan over long horizons to reach distant goals is important in robotic manipulation. This problem is extremely challenging when there are no intermediate reward signals to guide the planning, and when certain attributes of the manipulated objects cannot be easily obtained using existing perception tools, due to occlusions and noise in the sensory input. This work presents a new solution to this problem, where a low-level control policy is learned simultaneously with a finite-state task machine. The learned task machine splits the main task into individual sub-tasks, and signals to the low-level policy the stage that the robot has reached in performing the task. We demonstrated through several experiments that the task machine is a better memory structure for manipulation problems than some existing tools such as LSTM and GRU. The immediate next step in this work is to demonstrate the learned policies on a real manipulator. We will then investigate new approaches to further improve data efficiency through skill reusability and transfer learning. Another interesting future direction is to consider images directly as states instead of the 6D poses, and to compare to closely related end-to-end techniques such as (Nair and Finn, 2019).

References

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6077–6086, 2018.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5055–5065. Curran Associates Inc., 2017. ISBN 9781510860964.

Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, 3(null):397–422, March 2003. ISSN 1532-4435.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. AAAI'17, page 1726–1734. AAAI Press, 2017.

Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. Closing the learning-planning loop with predictive state representations. *Int. J. Rob. Res.*, 30(7):954–966, June 2011. ISSN 0278-3649.

A. Boularias and B. Chaib-Draa. Predictive representations for policy gradient in pomdps. In *ICML 2009*, pages 65–72, New York, NY, USA, June 2009. Max-Planck-Gesellschaft, ACM Press.

Krzysztof Choromanski, Carlton Downey, and Byron Boots. Initialization matters: Orthogonal predictive state recurrent neural networks. In *Proceedings of the Sixth International Conference on Learning Representations (ICLR)*, 2018.

Carlton Downey, Ahmed Hefny, Boyue Li, Byron Boots, and Geoffrey J. Gordon. Predictive state recurrent neural networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017.

Yan Duan, Marcin Andrychowicz, Bradly C. Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *NIPS*, pages 1087–1098, 2017.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SJx63jRqFm.

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

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519, 2016.

Roy Fox, Richard Shin, Sanjay Krishnan, Ken Goldberg, Dawn Song, and Ion Stoica. Parametrized hierarchical procedures for neural programming. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJl63fZRb.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels, 2018.

De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. *CoRR*, abs/1807.03480, 2018. URL http://arxiv.org/abs/1807.03480.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. volume 80 of *Proceedings of Machine Learning Research*, pages 2107–2116, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Wenhao Jiang, Lin Ma, Yu-Gang Jiang, Wei Liu, and Tong Zhang. Recurrent fusion network for image captioning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 499–515, 2018.

Leslie Pack Kaelbling. Learning to achieve goals. In *IN PROC. OF IJCAI-93*, pages 1094–1098. Morgan Kaufmann, 1993.

Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Proceedings of the 1st AAAI Conference on Bridging the Gap Between Task and Motion Planning*, AAAIWS'10-01, page 33–42. AAAI Press, 2010.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. volume 87 of *Proceedings of Machine Learning Research*, pages 651–673. PMLR, 29–31 Oct 2018.

Aryeh Kontorovich, Boaz Nadler, and Roi Weiss. On learning parametric-output hmms. volume 28 of *Proceedings of Machine Learning Research*, pages 702–710, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

James J. Kuffner and Steven M. Lavalle. Rrt-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 995–1001, 2000.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 3307–3317, 2018.

Suraj Nair and Chelsea Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. *CoRR*, abs/1909.05829, 2019. URL http://arxiv.org/abs/1909.05829.

Suraj Nair, Mohammad Babaeizadeh, Chelsea Finn, Sergey Levine, and Vikash Kumar. TRASS: time reversal as self-supervision. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, pages 115–121. IEEE, 2020. doi: 10.1109/ICRA40945.2020.9196862. URL https://doi.org/10.1109/ICRA40945.2020.9196862.

Alexander Neitz, Giambattista Parascandolo, Stefan Bauer, and Bernhard Schölkopf. Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. In *Advances in Neural Information Processing Systems*, pages 9816–9826, 2018.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L. Lewis, and Satinder P. Singh. Action-conditional video prediction using deep networks in atari games. In *NIPS*, pages 2863–2871, 2015.

Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Probabilistic movement primitives. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 2616–2624, Red Hook, NY, USA, 2013. Curran Associates Inc.

Lawrence R. Rabiner. Readings in speech recognition. chapter A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pages 267–296. Morgan Kaufmann Publishers Inc., 1990. ISBN 1-55860-124-4. URL http://dl.acm.org/citation.cfm?id=108235.108253.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Satinder Singh, Michael R. James, and Matthew R. Rudary. Predictive state representations: A new theory for modeling dynamical systems. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI '04, pages 512–519, Arlington, Virginia, United States, 2004. AUAI Press. ISBN 0-9749039-0-6. URL http://dl.acm.org/citation.cfm?id=1036843.1036905.

Marc Toussaint, Kelsey R. Allen, Kevin A. Smith, and Joshua B. Tenenbaum. Differentiable physics and stable modes for tool-use and manipulation planning. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 6231–6235. ijcai.org, 2019. doi: 10.24963/ijcai.2019/869. URL https://doi.org/10.24963/ijcai.2019/869.

Trieu H Trinh, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. Learning longer-term dependencies in rnns with auxiliary losses. *arXiv preprint arXiv:1803.00144*, 2018.

Niklas Wahlström, Thomas B. Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *CoRR*, abs/1502.02251, 2015.

Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. *CoRR*, abs/1710.01813, 2017. URL http://dblp.uni-trier.de/db/journals/corr/corr1710.html#abs-1710-01813.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France, 07–09 Jul 2015. PMLR.