Accelerate Then Imitate: Learning from Task and Motion Planing

Michael McDonald
Dylan Hadfield-Menell, Ed.

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-96
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-96.html

May 14, 2021

Accelerate, Then Imitate:
Learning from Task and Motion Planning

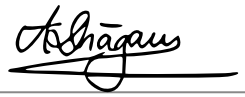
by Michael James McDonald

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor Anca Dragan
Research Advisor

(Date)

* * * * * * *

Dylan Hadfield-Menell
Second Reader

05/14/2021

Accelerate, Then Imitate:
Learning From Task and Motion Planning

Acknowledgments

No words can express my gratitude to (soon-to-be Professor) Dylan Hadfield-Menell for his mentorship over these past few years. His guidance and support (along with his unending patience) have made me both a better researcher and a better person, and this report would never have been possible without him.

I would like to deeply thank Professor Anca Dragan, for accepting me into InterACT Lab and providing the opportunity to see this work come to fruition. I would like to thank Professor Pieter Abbeel, for accepting me all those years ago as an undergraduate into the Robot Learning Lab and for the resulting opportunities I have been privileged to explore. And I would like to thank Thanard Kurutach for his invaluable insight and advice over the course of this project.

And finally, I would like to thank my family, for their unending support. In particular, thank you to my parents Catherine Coombs and James McDonald, for their unconditional encouragement in whatever I choose to do. Thank you to my twin brother Steven, for putting up with me for these past 24 years. And thank you to my grandfather Clyde Coombs and late grandmother Ann Coombs, who always made sure my brother and I had the opportunities and resources to pursue our dreams, and without whom I would not be here today.

Abstract

Accelerate, Then Imitate:
Learning From Task and Motion Planning

by

Michael James McDonald

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Anca Dragan, Chair

The capabilities of both imitation and reinforcement learning for robotics have burgeoned with the advent of deep learning, but these methods still struggle to extend to tasks with long time horizons. Hierarchical policy learning and goal-conditioning in policies have offered great promise in overcoming this limitation, but still cannot match the horizons or reliability of classical planners. Task and motion planning remains the gold standard for high-precision, multi-step tasks but suffers from computational burden and difficulties in planning directly from sensor data - limitations that neural networks do not have. In this work, we propose an asynchronous training method to integrate imitation learning into task and motion planning. Our method trains goal-conditioned hierarchical policies to emulate the planner, and in turn uses those policies to accelerate the planner and generation of training data. In robotic manipulation tasks, the partially trained policies achieve a 2x reduction in the combined time for motion plan refinement and simulated execution. For 7 DOF robotic pick-place tasks, our method produces end-to-end policies capable of placing four objects with an 86% success rate. And for 2d navigational pick-place tasks with high-dimension goals, our method can place five objects with an 88% success rate when working from state observations or an 83% success rate for three objects when using camera images.

Accelerate, Then Imitate: Learning From Task and Motion Planning

1 Main Work

1.1 Introduction

One of the success stories of the last 5 years has been the application of (deep) *policy learning* to robotics. These approaches have demonstrated an impressive ability to solve robot control tasks from perceptual input [1, 2, 3]. But that success has largely been limited to short-horizon tasks - e.g. a policy to grasp particular types of objects from a location. Learning policies that can reason over extended sequences of actions and adapt to novel tasks remains a challenge [4, 5].

When posed with long-horizon problems in robotics, a common solution is *task and motion planning* (TAMP). TAMP is a hierarchical approach to planning that integrates a search over abstract actions with motion planning to determine feasible trajectories. TAMP solvers can solve problem that require tens or hundreds of abstract actions, each consisting of complex geometric constraints in high-dimension configuration spaces [6, 7]. And under mild assumptions, they can be guaranteed to return a trajectory for any problem expressible in the structure of the search space [8]. But while these approaches are quite general, they are often prohibitively slow to run and rely on an ability to effectively track world state [9].

In this work, we propose a method that uses the output of TAMP solvers as the target for deep imitation learning. Similar design patterns have been effectively applied to learn motor control policies [10], visuomotor skills [1], and solve classical planning problems [11]. The hope is to learn policies that emulate the planner's capabilities without its computational overhead or need to model the environment. To do so, we identify and address the following three roadblocks: 1) the computational cost of generating supervision, 2) the challenge of learning the interplay between the abstract actions and low-level controls, and 3) the training-to-test distribution shift.

The first roadblock is that TAMP solvers run substantially slower than the planners used for supervision in the above. To compensate, we leverage our learned policies to accelerate the planner. We train a *motion policy* to imitate the output of a trajectory optimizer, and then initialize trajectory optimization around the output of the policy. As the policy improves, the computational expense of refining each motion plan decreases. To further amplify data generation, our algorithm decomposes across parallelized nodes. The distributed structure allows us to scale our system to utilize all available hardware.

The second roadblock is that directly imitating task and motion planner output is hard. The mapping from abstract goals to motor controls is complex and can require sharp transitions in control space for relatively minor changes in observation. To address this challenge, we leverage the domain-specific TAMP hierarchy in the architecture of the final policy. We train a *task policy* to output which action to take (i.e., grasp or place) and the associated parameters (i.e., the target object or location). The motion policy then executes conditioned on these outputs. Together the two policies form a single policy that maps observations and goals to low-level controls.

The final roadblock is distribution shift. As the time horizons increase, the policies become more likely to encounter observations not covered in their supervision. To account for this shift, we use our policy structure to adopt an active learning (AL) procedure that can be viewed as a hierarchical variant of Dataset Aggregation (DAgger) [12]. At the motion level, we sample rollouts of the motion policy and then augment trajectory optimization with a penalty for deviating from those rollouts. The penalty encourages the optimizer to provide controls similar to those from the policy, akin to DAgger at the motion level. The task planner then samples states where the task policy violates pre- or post-

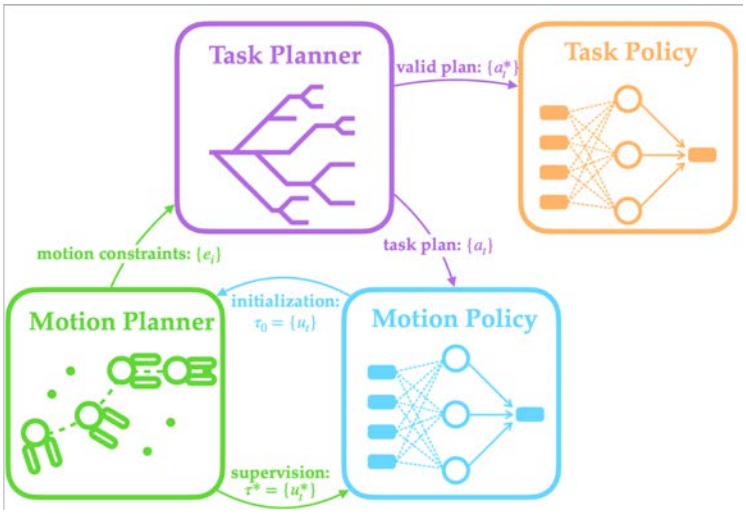


Figure 1: A system diagram of our core approach. In standard TAMP, the task planner and motion planner directly communicate. In our approach, the motion planner generates supervision for a motion policy π_{motion}. This policy can then be rolled out to initialize trajectory optimization and accelerate TAMP. We leverage the fast planner this produces to generate sufficient supervision to train a task policy π_{task} to imitate successful task plans. By combining π_{motion} and π_{task} we obtain a unified policy that can be executed online to achieve goals over long horizons.

conditions. The choice of these states focuses demonstrations to regions where the task policy and expert disagree, akin to DAgger in the abstract space.

The contributions of this work are as follows: 1) we provide a method to accelerate TAMP via learning a motion policy to map parameterized actions to controls and using that policy to amortize trajectory optimization; 2) we place this accelerated TAMP method inside a parallelized imitation learning framework to simultaneously train the motion policy and a goal-conditioned task policy that maps observations to a parameterized action; 3) we extend this framework to allow for active learning via the structure of the TAMP problem; and 4) we evaluate this system in a suite of pick-place domains and show that it is capable of learning to accomplish goals over (comparatively) long horizons with both high-dimension sensor data and high-dimension action spaces (e.g. 7-DOF joint control). Our approach uses established machine learning (and in particular, only feed-forward neural networks) and builds on standard robotics tools. We believe it has the promise to bridge the divide between classical planning and policy learning for robot control.

1.2 Preliminaries

We begin by briefly going over the components of our method.

1.2.1 Task and Motion Planning

Task and motion planning (TAMP) is a robot planning approach designed to account for complex goals over long horizons. TAMP breaks the problem into two components: an abstract, symbolic representation of actions (e.g., grasp, place) and a geometric representation of the world. Task planning operates on a logical representation of the world. It identifies sequences of *actions* that accomplish a goal, e.g., pick(obj_1), place(obj_1, $targ_1$), etc. Each action encodes a motion problem that must be solved or refined to obtain a trajectory that satisfies constrains and, thus, can be executed.

In this work, we will use a slight modification of the formalization introduced in [13]. A task and motion planning problem is a tuple $\langle X, F, G, U, A, f, x_0 \rangle$:

X: the space of valid world configurations, $x \in X$.

F: a set of *fluents*, binary functions of the world state that characterize the task space $f : X \rightarrow \{0, 1\}$. E.g., at($obj_3$, $targ_2$) or holding(obj_1).

G: a set of fluents $\{g_i\}$ whose conjuction defines the set of goal states.

U: the control space of the robot, $u \in U$.

A: the set of *action schemas*. Each action a has four components.

1. $a.params$: the parameters of the action (e.g., which object to grasp)
2. $a.pre$: a set of parameter-dependent fluents that defines the states when this action can be taken
3. $a.mid$: a set of fluents that constrains the allowable controls for this action
4. $a.post$: a set of fluents that will be true after the action is executed

f: the world dynamics; $f(x_t, u) = x_{t+1}$

x_0: the initial world configuration $x_0 \in X$.

A solution to a TAMP problem is a pair of sequences: $(\vec{a}, \vec{\tau})$. $\vec{a}$ is a sequence $\{a_i\}$ of high-level actions and $\vec{\tau}$ is a sequence $\{\tau_i\}$ of motion trajectories. A plan is valid if 1) each action's preconditions $a_i.pre$ are satisfied in the state it is executed in; 2) each trajectory τ_i satisfies the corresponding mid-conditions $a_i.mid$; 3) each trajectory's final state satisfies the corresponding post-conditions $a_i.post$; and 4) the final state satisfies the fluents that define the goal.

1.2.2 Imitation Learning

Imitation learning (IL) concerns observing demonstrations to acquire new skills. Methods for collecting demonstrations can be either passive or active. Passive approaches assume access to static data and are, as a result, simpler. Behavioral cloning (BC), the standard passive approach, trains a policy π via supervised learning on a fixed dataset and ignores the influence of the policy on the environment dynamics. For such an approach, Ross and Bagnell [14] showed that the expected cost under the 0-1 loss may scale quadratically with the time horizon. In contrast, active approaches interact with an expert that can respond to the learner's queries. A prevalent example is Dataset Aggregation (DAgger), as proposed by Ross et al. [12]. DAgger iteratively aggregates the target dataset with the demonstrator's behavior at states encountered by a mixed-policy, invoking the current learned policy π^i with probability β and querying the expert with probability $1-\beta$. The resulting demonstrations align with the state distribution encountered by the final policy. While pure DAgger is computationally infeasible to couple with a TAMP solver, we adapt a similar approach to target demonstrations.

1.2.3 Hierarchical Policy Learning

A common approach to training hierarchical policies is the options framework proposed by Sutton et al. [15], which assumes a-priori the existence of a set of options $\mathcal{O}$. An option in this context is a tuple $< \pi^{lo}, \beta, \mathcal{I} >$, where $\pi^{lo} : \mathcal{S} \rightarrow \mathcal{A}$ is a policy, $\beta : \mathcal{S} \rightarrow [0, 1]$ is a termination condition, and $\mathcal{I} \subset \mathcal{S}$ is the set of valid states from which π^{lo} may be invoked. Under such a formulation, a high-level policy π^{hi} is trained to select from available options. The policy executes the corresponding option until termination, and then repeats the process.

The TAMP formalization lends itself naturally to such a framework, with intuitive correspondence from $\mathcal{O}$ to $\mathcal{A}$, mapping task planning to the selection of $o \in \mathcal{O}$ and motion planning to execution of the respective π^{lo}. We leverage this abstraction to decouple training of the high-level policy π_{task} and low-level policy π_{motion}. Rather than train separate networks, the termination conditions are implicitly captured in the training for π_{task}. To provide additional guidance for π_{motion}, we augment π_{task} in a manner similar to that of Van et al. [16], where the authors train both a policy to predict the next $a \in A$ and a separate policy to predict a onehot-encoding of $a.params$. In this work, we condense these into a single policy that predicts both the action $a \in A$ and the encodings.

1.2.4 Learning for Long Horizon Planning and TAMP

Learning behaviors for planning has attracted significant interest [17, 18, 19, 20, 21]. In particular, using hierarchical structures to separately learn policies for sub-goal prediction and goal-oriented

control has shown promise in extending the horizons existing policy-learning methods can handle [22, 23, 24]. Still, these approaches have yet to achieve the planning horizon and control complexity of TAMP solvers.

Various works have proposed learning-based methods for improving efficiency in the different components of TAMP. Most of these have focused on the high-level search. Kim et al. [25] used a score-space representation to guide the search via transferring knowledge from previous plans. Wells et al. [26] learned a classifier to assess motion feasibility and incorporated it as a heuristic into the search. Kim et al. [27] used an actor-critic method trained form past experience for selecting continuous parameters during the search to improve planning efficiency.

For improving the efficiency of motion planning, Ichnowski et al. [28] used neural networks to predict an approximate trajectory to transition from a start frame to the goal, and showed using these approximate solutions to warm-start trajectory optimization provided significant reductions in overall planning time.

The above approaches provide ways to improve TAMP via learning, but do not attempt to acquire pure policy-based systems.

To train policies that behave like a TAMP solver, Paxton et al. [29] combined Monte-Carlo Tree Search (MCTS) in an abstract space with deep q-learning in a control space. They adopted a hierarchy similar to ours, with both a high-level policy to select options and a set of associated low-level controllers. Their approach did not need an existing TAMP solver but did not cover parameterized actions or tasks with multiple sub-goals.

Kase et al. [30] used static datasets to train hierarchical policies for performing TAMP from visual inputs. Their approach trained a high-level model to infer symbolic states, and low-level controllers to convert those symbolic states into executable actions. Since they did not attempt to train a model for predicting abstract actions, the system required a symbolic planner to be invoked at the start of each task.

For learning from TAMP solutions, Driess et al. [31] proposed a hierarchical policy structure similar to ours. They trained their policy to predict discrete action parameterizations and the associated continuous values, and then passed those to a learned controller. Unlike ours, their design explicitly accounted for joint reasoning over full action sequences. From an initial observation, their policy would predict a feasible sequence of discrete actions and continuous parameterizations. The action sequence then remained fixed, and they would update only the continuous values at action transitions. Their approach also differed from ours in that they did not incorporate policies into the data generation process. They instead used model predictive control (MPC) to build transitions around a set of pre-computed solutions, and incorporated the Hessian of the MPC cost into the training loss of their network.

1.3 Method

1.3.1 Overview

The output of our system is a modular TAMP policy that emulates (and accelerates) the behavior of an existing TAMP system. To accomplish this, we decompose the system into four types of nodes: (1) policy training, (2) task planning, (3) motion planning, (4) structured policy rollouts. The core motivation for our design is parallelization: each component can operate separate from the others in its own process or machine, and pass information via shared data structures. Arbitrary copies of the last three types may execute simultaneously, providing for efficient online data generation. This modularity allows for optimal resource allocation per component and provides flexibility to substitute alternate algorithms into our structure as needed (even online).

Algorithm 1 Policy Optimization Node

Require: Shared dataset D
Require: Shared memory ϕ // Policy parameters
Require: Imitation procedure $TRAIN()$
1: **while** not terminated **do**
2: $\tau = (o_0, u_0, r_0), \ldots, (o_T, u_T, r_T) \leftarrow D$
3: $\hat{\phi} \leftarrow TRAIN(\tau, \phi)$
4: Publish $\hat{\phi}$ to ϕ

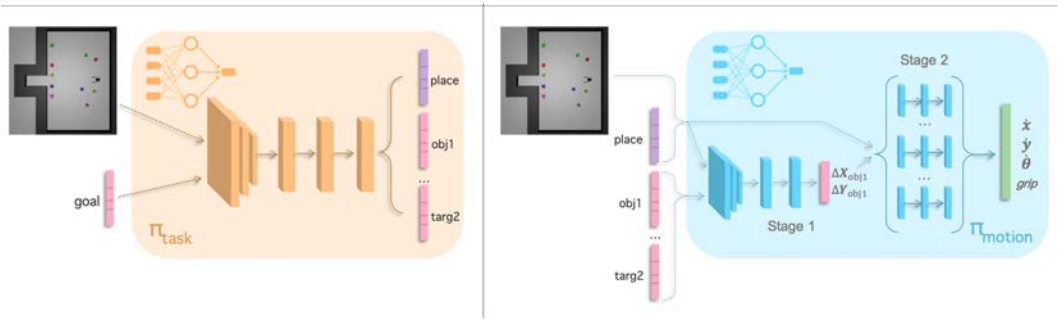


Figure 2: Policy Design. Left: π_{task} takes in the observations and the goal and produces one-hot encodings of the action parameterization. Right: π_{motion} converts the discrete parameterization to relevant continuous values, in this case the displacement from object 1 to target 2 (stage 1). The continuous values then combine with other observations and the action type $place$ selects which control network to use (stage 2). In practice, stage 2 may alternatively contain a single control network conditioned on the action.

1.3.2 Hierarchical Task and Motion Policies

The learned TAMP policy consists of two components. The first is a high-level task policy π_{task} to select a parameterization for the next action a. The second is a low-level motion policy π_{motion} to select the next continuous control u conditioned on the ouput of π_{task}. Each sub-policy is parameterized as a neural network, and trained in isolation within its respective node. No back-propagation occurs between π_{task} and π_{motion}. For this work we found standard supervised learning sufficient for good performance, however the modularity of the system permits the substitution of more complex procedures into the node (e.g. generative adversarial imitation learning [32] or inverse reinforcement learning [33]). Specific network architectures and hyperparameters are included in the appendix.

Task Policy

To emulate task planning, π_{task} maps from a continuous observation space into the discrete space specified by $\langle A, F, G \rangle$. Per figure 2, the output of π_{task} consists of a sequence of vectors. The first provides a one-hot encoding of the choice of action-type (e.g. place). Each subsequent vector provides a one-hot encoding for the choice of action parameters (e.g. $obj1$). As detailed in algorithm 1, training data for this policy accumulates in a shared dataset D_{task}.

Motion Policy

The motion policy π_{motion} consists of two stages, as shown in figure 2. The first is an attention module. It converts the continuous observations and the outputs of π_{task} into a continuous representation. For example, the onehot-encoding of $place(obj1, targ2)$ would become the euclidean displacement of $obj1$ from $targ2$. If the policy has access to state information, this can be defined explicitly with an affine transformation. In more general observation spaces, a separate regression model is trained.

Algorithm 2 Task Planning Node

Require: Shared queues Q_{task}, Q_{motion}
Require: Problem distribution P_{prob}; $p_{new} \in [0, 1]$
while not terminated **do**
  if flip(p_{new}) **then**
    $(x_0, \Phi, \text{goal}, \tau) \sim P_{prob}$
  else
    $(x_0, \Phi, \text{goal}, \tau) \leftarrow \text{pop}(Q_{task})$
  $\vec{a} \leftarrow \text{task\_plan}(\Phi, g)$
  push(Q_{motion}, (x_0, Φ, goal, $\vec{a}$)

The encoded abstract action, the continuous parameterization, and the original continuous observation pass to the second stage, which predicts the next optimal control. This stage may consist of either separate control networks, of which one is selected per the choice of action, or a unified network that is conditioned on the action. In either case, the training data for this policy accumulates in a shared dataset D_{motion} as detailed in algorithm 1.

1.3.3 Parallel Data Generation

The task planning and motion planning nodes follow a formulation inspired by that of Srivastava et al. [8], modified to allow for guidance from the current policies. These nodes generate the supervision data for the policy training, incorporating the current policies to guide planning.

Algorithm 3 Motion Planning Node

Require: Shared queues Q_{motion}, Q_{task}
Require: Expert datasets D_{motion}, D_{task}
Require: Motion policy π_{motion}
while not terminated **do**
  $(x, \Phi, \text{goal}, \vec{a}) \leftarrow Q_{motion}$
  $\tau \leftarrow []$
  for $a_i \in \vec{a}$ **do**
    $\tau^i \leftarrow \text{rollout}\big(x, \text{goal}, \pi_{motion}(*|a_i)\big)$
    $\tau^i, \text{success} \leftarrow \text{motion\_plan}\big(x, a_i, \tau^i\big)$
    if success **then**
      $x \leftarrow \tau^i_{end}$; $\tau \leftarrow \tau \cup \tau^i$
      $D_{motion} \leftarrow D_{motion} \cup \{(a_i, \tau^i)\}$
    else
      ## compute missing geometric facts $\{e_i\}$
      push$(Q_{task}, (x, \Phi \cup \{e_i\}, goal))$
      break
  ## successful task plan, send to task policy
  $D_{task} \leftarrow D_{task} \cup \{(\tau_i, a_i, \text{goal})\}$

Task Planning

We outline our procedure for task planning in algorithm 2. The system maintains a shared priority queue Q_{task} of encountered problems (with random probability choosing a new problem from a predefined distribution P_{prob}). The task node runs a black-box task planner from a state configuration x_0 and a set of discovered geometric facts Θ, and generates a valid sequence $\vec{a}$ of action schemas such that $goal \subset \vec{a}(\Theta)$. Given the decentralization of the planning, a valid $\vec{\tau}$ is also tracked (with $\vec{\tau}_{end} = x_0$), which transitions some originally sampled $(\hat{x_0}, \hat{\Theta}, \hat{goal}, \emptyset) \sim P_{prob}$ to the current problem $(x_0, \Theta, goal, \vec{\tau})$. Intuitively, $\vec{\tau}$ represents a reduction from an initial problem to a simpler one requiring a shorter plan.

Motion Planning

We outline our procedure for motion planning in algorithm 3. The proposed $\vec{a}$ passes to the motion planning node via another priority queue. The node then refines the elements a^i into valid trajectory segments τ^i and discovered geometric facts $\{e_i\}$. The process terminates when either all a^i have successfully refined or the motion planner fails to return a trajectory.

Refinement occurs in two stages: (1) motion policy rollout and (2) fixing these rollouts via trajectory optimization. For action a^i, the motion policy $\pi_{motion}(*|a^i)$ is rolled out from the current state τ^{i-1}_{end} to obtain $\hat{\tau}^i$. Trajectory optimization is then initialized around $\hat{\tau}^i$. As π_{motion} improves, the optimizer requires less effort to find solutions.

To reduce distribution shift within the system, we augment with the optimization with an additional cost term $||\hat{\tau}^i - \tau_{opt}||^2_A$. Here $|| * ||_A$ is the velocity norm outlined by Dragan [34]. This cost encourages the optimizer to produce controls similar to those sampled from $\pi_{motion}(*|a^i)$.

1.3.4 Structured Policy Supervision

The final component of our system, the structured rollout node, identifies states where the task policy generates invalid actions. We adopt a procedure that enforces the structure of the action schemas A onto the policy rollouts. Note this structure is enforced only in training and not during evaluation of the policies.

At each timestep t, the task policy π_{task} proposes a parameterized action $\hat{a}_t$. We then check the post-conditions of the last action a_{t-1} and the pre-conditions of $\hat{a}_t$. If both are satisfied, this is a valid high-level action sequence and we set $a_t = \hat{a}_t$. If the post-conditions of a_{t-1} are not met (and $\hat{a}_t \neq a_{t-1}$), we reject $\hat{a}_t$ and leave a_{t-1} unchanged. In this case, we generate a *negative* training example for the task policy: we populate a dataset D_{neg} and train the network to minimize the likelihood of those (invalid) high-level actions. We also generate negative examples when a_{t-1}'s post-conditions are satisfied and $\hat{a}_t = a_{t-1}$. This discourages redundant actions that do not change the high-level state.

If the pre-conditions of $\hat{a}_t$ are not met, we need to select an alternative action for a_t. We do this by sampling from a softmax distribution over the raw logits of the network with temperature parameter

λ. We continue sampling until 1) either a valid action is found or 2) we hit a termination condition. For each rejected candidate $\hat{a}_t$, we generate a negative training example.

If we rejected any candidate $\hat{a}_t$ at timestep t or terminate without having reached the goal, a new problem instance is added to the shared queue Q_{task}. The problem instance contains four values: a state vector x, a symbolic description Φ of the world in state x, the trajectory τ up to the timestep of x, and the current goal $goal$. If $\hat{a}_t$ was rejected due to pre-condition violations, we set x to the current state. If the rollout failed to reach the goal or $\hat{a}_t$ was rejected due to post-condition violations, we set x to the state of the most recent action transition. The assumption in this case is that π_{motion} has failed to execute properly, and the system should query from the last timestep when both π_{task} and π_{motion} were trusted. Our full approach is outlined in algorithm 4.

Algorithm 4 Structured Rollout Node

Require: Datasets D_{task}, D_{neg}
Require: Problem distribution P_{prob}
Require: Shared queue Q_{task}
Require: Motion policy π_{motion}; Task policy π_{task}
Require: Temperature λ; coefficient η
Require: Environment dynamics $f : X \times U \to X$

while not terminated **do**
  $x, \Phi, goal \sim P_{prob}$
  $a \leftarrow \pi_{task}(x, \lambda)$; $x^{prev} \leftarrow x$; $\tau \leftarrow []$; $\vec{a} \leftarrow []$
  while not goal(x) and not timeout **do**
    $\hat{a} \leftarrow \pi_{task}(x, \lambda)$
    if $\hat{a} \neq a$ and $a.post(x)$ **then**
      $x^{prev} \leftarrow x$
      push$(Q_{task}, (x, \Phi, \tau, goal))$
      while not $\hat{a}.pre(x)$ and not max_iter **do**
        $D_{neg} \leftarrow D_{neg} \cup \{(\{x\}, \{\hat{a}\}, \text{goal})\}$
        $\lambda \leftarrow \eta \cdot \lambda$
        $\hat{a} \leftarrow \pi_{task}(x, \lambda)$
      $a \leftarrow \hat{a}$
    else if $\hat{a} \neq a$ and not $a.post(x)$ **then**
      push$(Q_{task}, (x^{prev}, \Phi, \tau, goal))$
      $D_{neg} \leftarrow D_{neg} \cup \{(\{x\}, \{\hat{a}\}, \text{goal})\}$
    else if $\hat{a} = a$ and $a.post(x)$ **then**
      push$(Q_{task}, (x, \Phi, \tau, goal))$
      $D_{neg} \leftarrow D_{neg} \cup \{(\{x\}, \{a\}, \text{goal})\}$
    $u \leftarrow \pi_{motion}(x|a)$
    $x \leftarrow f(x, u)$
    $\Phi \leftarrow$ update(x, Φ)
    $\vec{a}$.append(a); τ.append(x)
  if goal(x) **then**
    $D_{task} \leftarrow D_{task} \cup \{(\tau, \vec{a}, \text{goal})\}$
  else
    push$(Q_{task}, (x, \Phi, \tau, goal))$

The benefit of these structured rollouts is two-fold. First, at each timestep t we can guarantee the sequence of abstract actions up to t forms a valid plan. This enables the system to isolate specific points of failure: either the high-level has provided a bad transition or the low-level has failed to properly execute. Second, we isolate configurations where the task policy must disagree with the task planner, without the cost of invoking the task planner. This enables efficient feedback from the task policy into the training.

1.4 Evaluation

In this section we evaluate the components of our system as well as the performance of our trained policies. These evaluations include: (1) the acceleration of task and motion planning over the course of training in our system; (2) comparisons of our approach to reinforcement learning and learning from offline data; (3) the performance gains of our policy structure over a flat policy; (4) the performance gains of using the policies to guide data generation; (5) the ability of our system to train from visual inputs; and (6) the ability of our system to train on problems with four and five objects.

1.4.1 Setup

For our experiments, we used Fast-Forward [35] as our task planner and a variant of the trajectory optimizer used by Hadfield-Menell et al. [36] as our motion planner. All experiments were run for five random seeds, with plots showing 500-sample rolling averages and standard deviation. For resource allocation, unless stated otherwise, evaluations of our method used 2 processes for task planning, 18 processes for motion planning, and, when applicable, 10 processes for supervised exploration. Observations in all domains, unless otherwise stated, contained a one-hot encoding of the goal, displacement vectors from the robot gripper to the center of each object, and displacement vectors from the center of each object to its respective goal location. Furthermore, each neural network was trained inside its own process. Further details are in the appendix.

We provide experimental results from two simulated environments, shown in figure 3. The first is a 2D pick-place simulator implemented in DMControl [37]. The robot is modelled as a cylinder with

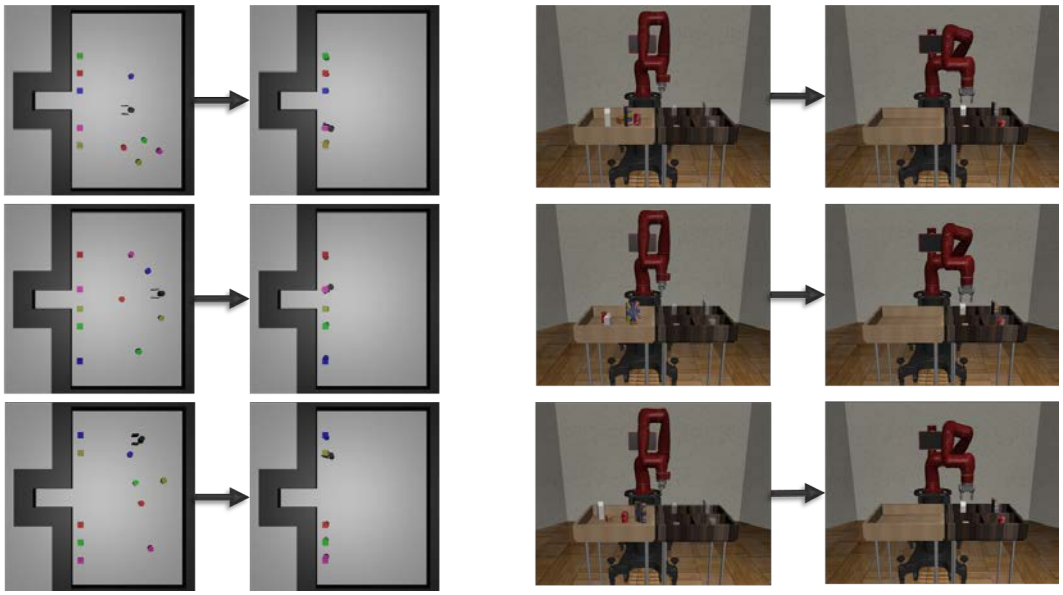


Figure 3: Left: Our 2D pick-place domain; the goal is for the robot to move objects from the right onto the corresponding targets on the left. Right: The Robosuite pick-place domain; the goal is for the robot to grasp each object and move it to its corresponding bin

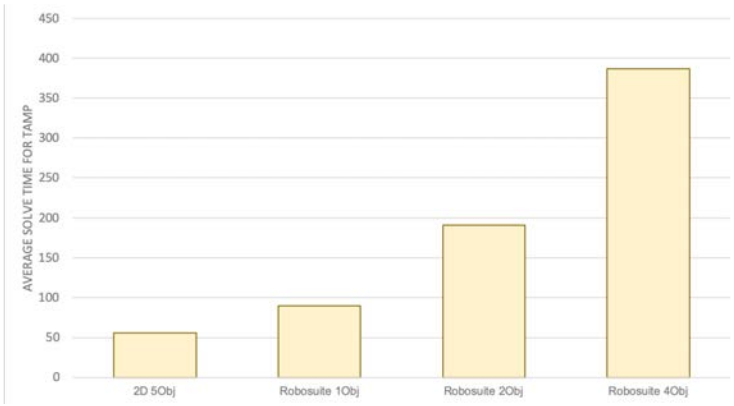


Figure 4: Average time in seconds for the default task and motion planner to generate a trajectory and begin execution. By comparison, the policies we train can begin execution immediately.

an attached actuated gripper. The controls are the x, y, and rotational velocity of the robot plus the gripper value (for 4 DOF). Objects are generated at random positions in the world, with a randomly assigned (unique) target location chosen from a possible set of 8. A goal is considered achieved when each object overlaps with its respective target (i.e. $||\text{pos}_{obj} - \text{pos}_{targ}||_2 \leq \text{radius}_{obj}$). The task domain contains three actions: moveto-and-grasp, transfer, and place-and-retreat. The observation space is augmented with the robot's current directional and rotational velocity, as well as LIDAR-style sensor. The LIDAR sensor performs ray checking along 39 directions evenly spaced around the body of the robot and each pointing away from the robot's center of mass.

The second environment is the Pick-and-Place environment provided in Robosuite [38], using a Sawyer robot. The controls are the joint velocities of the robot plus the gripper value (for 8 DOF). In this environment, four objects of different shapes/sizes are initialized at random locations on the left, and the goal is to place each object in the corresponding container on the right. The containers for each object are the same in every run. The task domain contains four actions: moveto, grasp, moveholding, and putdown. The observation space is augmented with the current joint angles of the

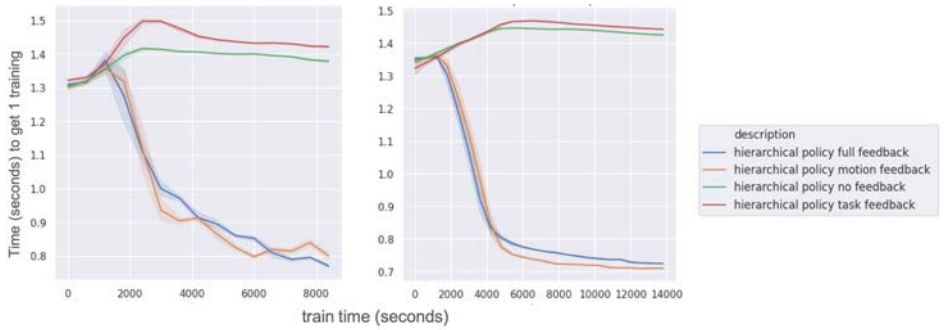


Figure 5: Average duration (in seconds) for each motion node to generate one timestep of supervision over the course of training. Initializing trajectory optimization around π_{motion} yields approximately a 2x speed up in plan refinement, effectively doubling the rate at which supervision is generated. We show results in the Robotsuite domain with only the cereal object (left) and the cereal and milk objects (right).

robot as well as the translation and rotational (recorded in axis-angle format) displacements from the gripper to each object.

The first domain has a relatively simple control space but a complex task space, with $\frac{T!}{(T-O)!}$ possible goal configurations. Here T denotes the number of potential target locations and O the number of objects; in the five-object variant this yields 6720 unique possible goals. In contrast, the second domain has a simple task space, as the goals are fixed, but a complicated control space.

1.4.2 Acceleration of Task and Motion Planning

The primary advantage of learned policies is the ability to instantaneously execute within an environment. As shown in figure 4, task and motion planning requires over six minutes on average for our hardest problems before it can generate the first action to take. Each neural network requires negligible compute for each forward inference, hence no such delay exists when acquiring the first action from the policy.

The first question we answer is if our learned policies actually accelerate the TAMP process. Motion planning in the 2D pick-place domain incurs minimal overhead, hence we restricted our focus for these experiments to the more challenging Robosuite domain. The system was evaluated with no feedback, only task-planner feedback, only motion-planner feedback, and full feedback to compare how the expense of plan refinement changes over the course of training. As seen in figure 5, our system was able to reduce the expense to refine and execute a plan by almost a factor of two as compared to those variants without feedback, doubling the rate at which supervision data is generated.

1.4.3 Comparisons to Other Training Methods

To illustrate the need for learning from TAMP on these domains, we compare against both flat reinforcement learning (RL) and existing results in learning from human demonstrations.

For the 2D pick-place domain, we used the implementation of proximal policy optimization (PPO) [39] included in the Stable Baselines package [40]. We ran with the default hyperpameters and parallelized training over 60 vCPUs. As seen in figure 6, PPO rarely achieved the goal even in the 1 object variant, and never in the 2 object variant. As shown by figure 7, it was only able to move objects modestly towards their goals. The challenges to flat RL here include: (1) proper placement involves deceleration and direction reversal that create sharp discontinuities in the control space, (2) the objects require precise placement relative to the size of the work-space (and are easily knocked off-target by small motions), and (3) the presence of multiple sub-goals inhibits reward shaping. A particular difficulty is the use of the gripper: the policy learns how to push each object towards its goal, but never discovers the ability to grasp objects and obtain more stable control.

For the Robosuite domain (with a single object), we refer to existing benchmarks that show state-of-the-art RL has yet solve this domain. Fan et al. [41] were unable to scale PPO and deep deterministic

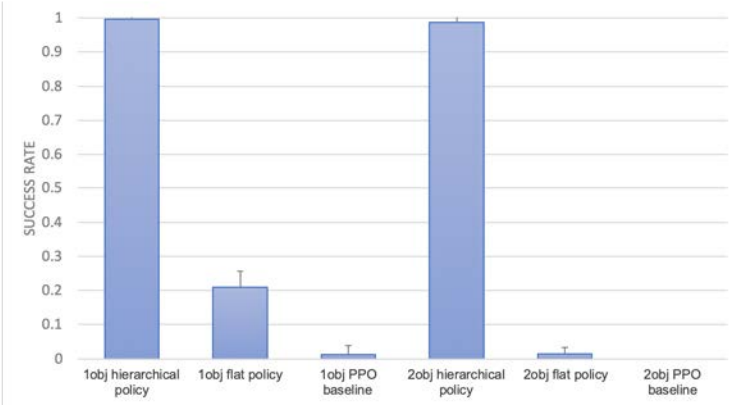


Figure 6: Binary success rate at the end of training for the 2d pick-place domain with 1 or 2 objects. Our approach can obtain hierarchical policies with $> 99\%$ success rates. Flat policies trained in the same manner succeed only 1 out of 5 times for a single object and almost never for two objects. Policies trained on the same problem via PPO almost never reach the goal.

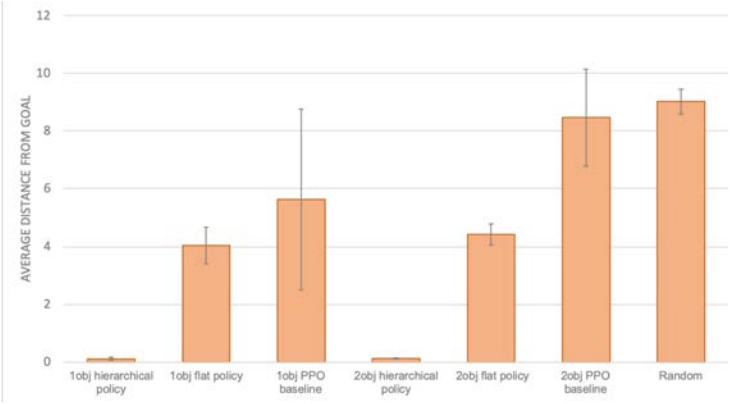


Figure 7: Average remaining distance from each object to its goal at the end of a rollout from a fully-trained policy for the 2d pick-place domain. Smaller distances indicate better performance. Hierarchical policies place objects on average one-tenth the distance of flat policies trained in the same manner. Standard RL makes progress towards the single object variant but lacks precision in localizing the goal and does not make progress on the two object variant. Random denotes the average distance when sampling a new random configuration.

policy gradients (DDPG) beyond moving a single object. For the one object variant, the official benchmarks for Soft-Actor Critic (SAC) achieve a normalized return below 50 for 500 timesteps whereas our average returns for the same time horizon ranged from 320 (with flat policies and no feedback) to over 390 (with hierarchical policies and combined motion and task feedback) [38].

To show the difficulty of learning from offline demonstrations on the Robosuite task, we again refer to existing results. In a single object variant of this task, Mandlekar et al. [42] were able to achieve success rates on the order of 45% using offline human demonstrations from state data, whereas several variants of our approach achieved success rates over 95% as shown in figure 9.

1.4.4 The Benefit of Hierarchy

The next question we answer is whether the hierarchical policy structure provides benefit. We compared the performance of training with our proposed hierarchy against training with a flat policy that mapped directly from the observation and goal to controls. Observation and control spaces, along with any hyperparameters unrelated to policy structure, were kept fixed for all evaluations on a task.

For the Robosuite domain with a single object, as seen in figure 9, hierarchy provided no noticeable benefit in the absence of feedback but provided significant benefit in the presence of either motion-

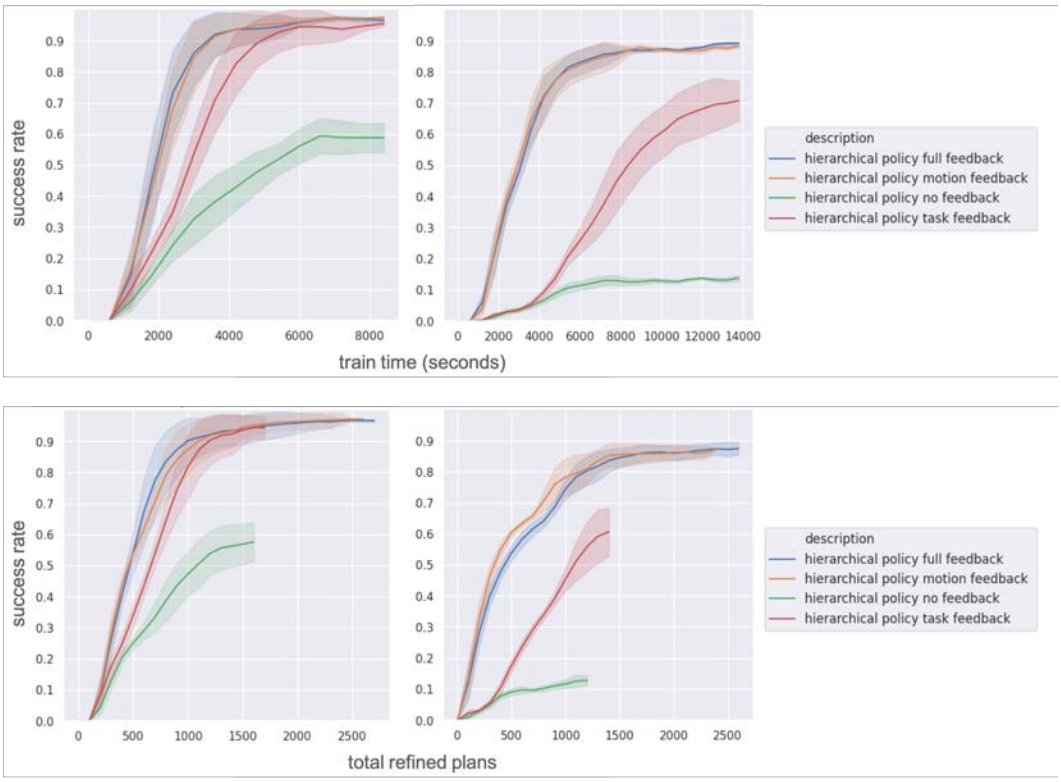


Figure 8: Binary success rates over the course of training. Top: as a function of training time. Bottom: as a function of planned trajectories used for supervision. Left: Evaluation in the Robosuite domain with only the cereal object. Right: Evaluation in the Robosuite domain with the cereal and the milk objects. Feedback from π_{motion} into the planner offered the greatest performance increase. Feedback from π_{task} improved success rates over training without feedback, but offered no clear benefit in the presence of motion feedback. The bottom row indicates differences in performance cannot be attributed solely to the rate of data generation.

planner or task-planner feedback. By contrast, as seen in figure 10 the two object variant improved considerably using the hierarchical policy as compared to the flat - even when the hierarchical policy trained with no feedback and the flat policy did. A direct comparison on task-planner feedback was infeasible due to the dependency of our structured approach on π_{task}. Instead, we ran a variant in which random configurations from rollouts of the flat policy were passed back to the task planner. These variants are marked in figure 9 as $random\ task\ feedback$.

In the 2D pick-place domain, hierarchy is critical. Figure 6 shows the fully-trained flat policy significantly under-performed its hierarchical counterpart when given a single object. With two objects, the flat policy never achieved the goal. To shed additional light on these numbers, figure 7 shows that the hierarchical policy was able to place the objects precisely on the goal. This represents distances approximately 10x closer than the flat policy. While this domain seems simpler than the Robosuite domain, it contains several pitfalls uniquely challenging to the flat policy: (1) action transitions involve deceleration and direction reversals that create sharp discontinuities in the control space, which the flat policy does not capture, (2) there are far more possible goals, which increases dependence on the one-hot goal vector when choosing controls, and (3) the objects require precise placement relative to the size of the work-space.

1.4.5 The Benefit of Feedback

The third question we answer is whether the two forms of feedback provided in the system (i.e. initializing the motion planner at π_{motion} and using structured rollouts to generate task problems) benefit training. For the Robosuite domain, we compare four variants: without feedback (essentially

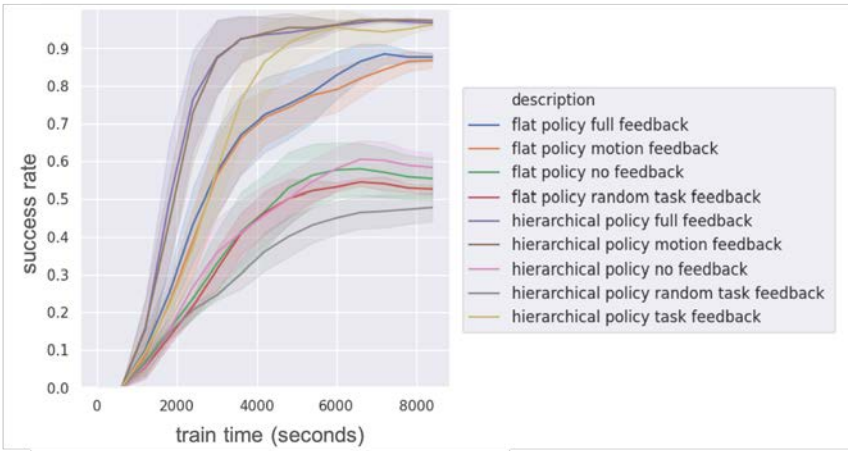


Figure 9: Full ablation study for Robosuite domain with only cereal object. Binary success rate over the course of training as a function of training time. With feedback present, variants with hierarchy outperformed those without. Without feedback, hierarchical and flat variants showed no difference. Motion feedback improved performance in all cases. Task feedback improved performance in isolation, but had no obvious impact in the presence of motion feedback.

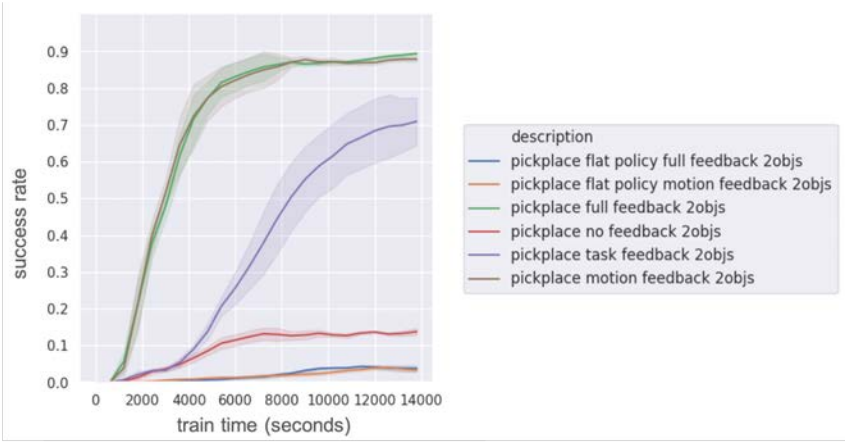


Figure 10: Partial ablation study for Robosuite domain with both cereal and milk objects. Binary success rate over the course of training as a function of training time. Hierarchical policy universally outperformed flat policies. Motion feedback improved performance for the hierarchical policy but not the flat. As in the one-object variant, task feedback improved performance for the hierarchical policy in isolation but had no obvious impact in the presence of motion feedback.

behavioral cloning), with only feedback between the motion planner and policy, with only feedback between the task node and the structured rollout node, and with full task and motion feedback.

Motion feedback had a sizable impact within the Robosuite domain. As shown in figure 8, the feedback into the motion planner provided significant performance gains with both one and two objects present. Notably, the TAMP acceleration alone does not explain this gap in performance. As shown in figure 8, the variants with motion planner-policy feedback outperformed those without as a function of the data generated for supervision.

The results in figures 9 and 10 show that task-planner feedback (via the structured rollout node) offers benefit in isolation, but not in the presence of motion-planner feedback. To evaluate the influence of our structured problem selection, we provide a baseline (without motion feedback) where random states from the policy rollouts were fed back to the task planning node. These variants are marked in figure 9 as $random\ task\ feedback$, and performed no better than the variants with

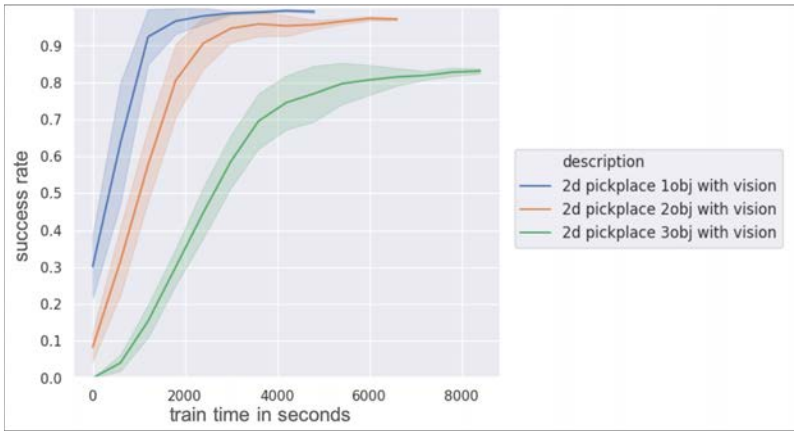


Figure 11: Binary success rate over the course of training from vision for 2d pick-place domain with increasing number of objects. The policies successfully placed each object on the correct target over 99% of the time for a single object present, 97% of the time for two objects present, and 83% of the time for three objects present.

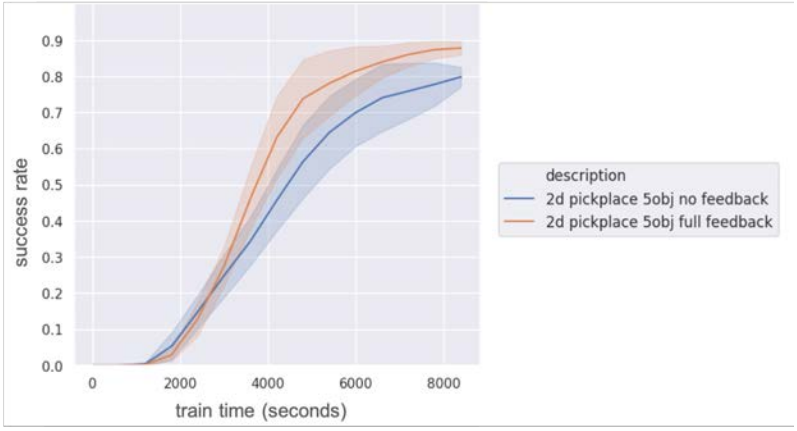


Figure 12: Binary success rate over the course of training for 2d pick-place domain with five objects. By the end of training, the policies are able to transfer all five objects to the correct targets over 88% of the time

no feedback at all. This indicates the value of task feedback depends on informed problem selection, and supports the use of the structured rollouts to guide selection.

For the 2d pick-place domain, we focus on a direct comparison of full feedback vs. no feedback. As seen in figure 12, feedback offered only modest benefit to training in the hardest variant of this domain. Taken together, these results suggest the benefit of feedback depends more on the dimension of the control space than that of the task space.

1.4.6 Vision-Based Learning

Now we examine the ability of our system to train from camera inputs. For these experiments, stage 1 of the motion policy is a convolutional neural network that takes both an image and the onehot-encoded action parameterization. Each image was 112x112x3 pixels in size, acquired from a fixed camera centered over the scene. As seen in figure 11, the system suffered only modest decreases in performance as the number of objects increased. With one and two objects present, the policies placed all objects on the correct targets over 97% of the time. With three objects present, the policies were still able to place all three objects on the correct targets over 83% of the time.

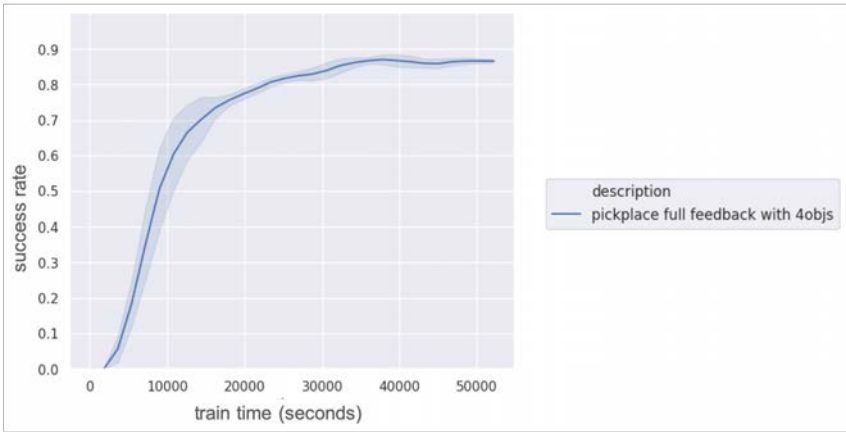


Figure 13: Binary success rate over the course of training for Robosuite pick-place domain with all four objects. By the end of training, the policies are able to successfully transfer all four objects to their target containers over 86% of the time

1.4.7 Handling of Long Horizons

We finish with a demonstration of our method's ability to scale to problems that require over a dozen high-level actions to solve. We plot the results in figure 13. Our final policies successfully complete the full Robosuite pick-place task with 4 objects 86% of the time. Similarly, our method trained policies capable of completing five object pick-place problems over 88% of the time, as seen in 12.

1.5 Discussion and Future Work

In this work we propose a method for training goal-conditioned hierarchical policies via imitation of task and motion planning, and demonstrate the ability of the resulting policies to solve complex long-horizon tasks with sensory observations and with high dimensional control spaces. Our method is able to create a symbiotic relation between learned policies and the TAMP solver, allowing the policies to accelerate the plan refinement process and in turn accelerating the generation of training supervision. The parallelized nature of our algorithms inherently allow them to scale with available hardware, making the system particularly well-suited to industrial environments where many robots operate simultaneously. And we provide ablation studies to validate both our policy structure and our methods for handling distribution shift for the learned policies - results which can help guide extensions and generalizations of this approach to new planners and environments.

There are several limitations that should be addressed in future work. First, our evaluation tasks had a (relatively) simple structure in the high-level space. We would like to evaluate the system in scenarios that require complex geometric reasoning, such as navigation around obstacles or the use of tools. Second, our tasks do not impose large deviations between train and test time. We hope to evaluate the system's ability to generalize by training in environments with more variability. Finally, we have not yet attempted to extend these results to physical robots. To do so, we hope to integrate sim-to-real transfer into our training.

2 Asynchronous Intervention in Learning from Demonstrations

2.1 Introduction

Advancements in robotics over the last ten years have rapidly pushed automated systems into wide swathes of life, from hospitals [43, 44] to factories and warehouses [45, 46] to our homes and cars [47, 48, 49, 50, 51]. The surfeit of scenarios and novel environments robots will encounter brings rise to a need for systems that can learn and adapt to new tasks, but also make it wholly infeasible for these systems to learn from scratch [52, 53, 54]. Learning from Demonstrations (LfD) [55] has become the go-to approach for robots to quickly learn to perform new tasks, covering methods from simple imitation learning [12] to inverse reinforcement learning [33].

Learning from Demonstrations contains a core challenge: it is difficult for humans to choose the most informative demonstrations for the robot [56, 57]. Most prior work has assumed the existence of human demonstrators, and focused on expanding the expressiblity of the robot, from the questions it asks [58, 59] to how it demonstrates failure [60]. But what if the demonstrators cannot understand the robot? The use of classical control systems, such as task and motion planners, to provide demonstrations to robots has gained increasing attention in recent years [31, 30]. These systems are computationally expensive, struggle to operate from sensor data, and typically require partial-to-full replanning when the environment shifts unexpectedly - hence the desire for robots to learn reactive policies. Using them to demonstrate tasks makes it possible to scale LfD to scenarios where humans demonstration would break-down, e.g. due to lack of physical access or when there are many more learners than humans present. But while they are reliable and can be freely queried for demonstrations, the same limitations that make them undesirable for practical tasks inhibit their ability to be good demonstrators. As an added challenge, unlike humans they cannot easily answer questions or form explanations of why a robot has failed. So what role should the human take to intermediate between the demonstrators and the learners?

Our key insight is that human instincts to anticipatorily correct mistakes can be leveraged to direct demonstration queries to regions where the robot will fail. We assume that the robot has access to black box demonstrators (which in this case are task and motion planners) that are expensive to query, and that a human can observe but not physically intervene in the training. The contributions of this report are as follows: (1) a proposed algorithm for efficient asynchronous human guidance of demonstrations from black-box demonstrators, (2) a straightforward user-interface for robustly locating the states to query, and (3) experimental evaluation of this system with both trained and untrained users.

2.2 Preliminaries

2.2.1 Active Learning

A primary challenge to learning from demonstrations is disagreement between those states encountered by the learner and those encountered by the demonstrator. The simplest form of imitation learning, behavioral cloning, obtains a policy via supervised learning on a dataset of pre-collected demonstrations. As shown by Ross and Bagnell [14], such an approach may have errors in the learned policy under the 0-1 loss scale quadratically with the duration of execution. A common remedy to this challenge is to adopt an iterative training procedure. Dataset Aggregation (DAgger) is the archtypical such method. DAgger iteratively aggregates the target dataset with the demonstrator's behavior on states encountered by a mixed-policy, invoking the current learned policy π^i with probability β and querying the expert with probability $1 - \beta$. Under such an approach, demonstrations align with the distribution induced by π. Unfortunately, the expense of replanning at every timesteps makes DAgger intractable when the demonstrator is something like a taks and motion planner.

This gives rise to the need for active learning (AL), where the learner can make targeted queries to the demonstrator[61]. The motivation of AL is to maximize the information conveyed via each demonstration; queries range from simply choosing which states to ask for demonstrations [62] to more complex questions[58] that provide greater context on the demonstrations. Unfortunately most approaches still suffer from requiring large number of queries [62]. This motivates a desire for queries a human can answer efficiently yet convey meaningful information.

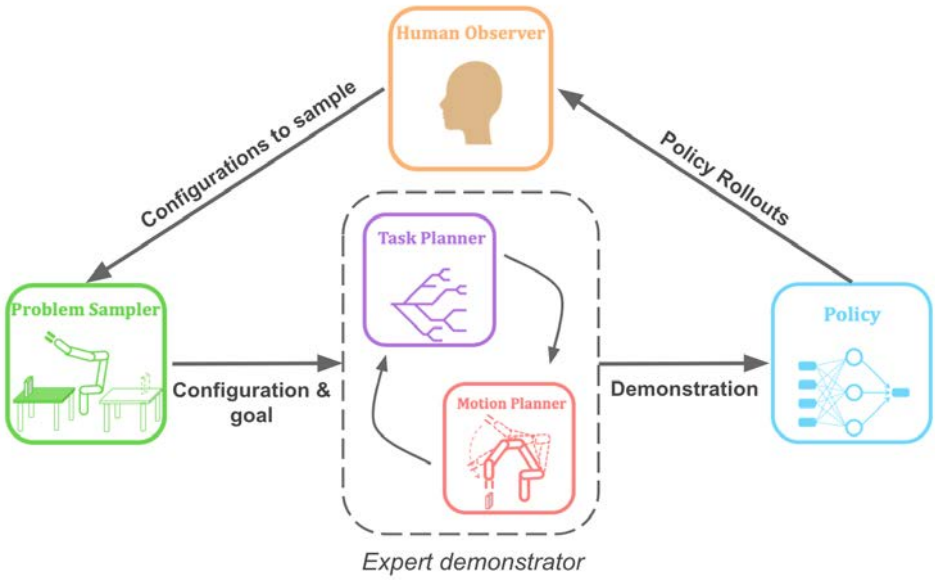


Figure 14: Overview of the proposed method. Starting from an initial problem distribution, the expert demonstrator provides trajectories to the policies, which train via direct supervision. Simultaneously, the human observes past trajectories of the robot and identifies when the robot should have queried for help, passing this to the problem distribution.

2.2.2 Humans Anticipate Mistakes

Contrary to other works that seek to maximize the expressibility of queries posed to the human [60, 58], our approach seeks to maximize the number of queries answered by relying on innate human abilities to anticipate mistakes.

When presented with adults who were about to retrieve an object from an incorrect location (but had not yet done so), infants as young as 18- and 24- months old consistently indicated the true location via pointing [63]. Such anticipatory corrective motion at a young age reflects an instinctive cognitive model of forward action prediction geared specifically to help others avoid errors.

Leveraging a human's instincts relies on an assumption that humans will attribute the same cognitive model to a robot that they attribute to other humans. One important facet of this is the degree of anthropomorphism humans attribute to the robot. While one may expect humanoid robots to fare better in this regard than non-humanoid robots, prior work has shown that in fact non-humanoid robots may attain a higher degree of anthropomorphism than their humanoid counterparts [64]. This has the particular upside that humans may be more likely to extend their pre-existing cognitive models to robots designed for functional purposes. Additional prior work has in fact shown that humans are more willing to donate time to robots framed in functional terms (e.g. specifying height and weight) than when framed with anthropomorphic terms (e.g. a name and backstory) [65]. Taken together, this supports the notion that humans will extend their cognitive models for anticipatory help to robots with goal-oriented tasks.

2.3 Methods

2.3.1 Problem Setup

For this work we model the environment as a Markov Decision Process (MDP), consisting of the state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and reward function $r : \mathcal{S} \to \mathbb{R}$, which in this case is a binary signal that the task has been completed. During training, we assume access to a pre-defined initial problem distribution P that samples configurations from $\mathcal{S}$ and may be freely queried to reset the environment configuration. The expert demonstrator consists of a parallelized version of the task and motion planning algorithm outlined by Srivastava et al. [8].

2.3.2 Flow of Information

The system performs training in an asynchronous manner outlined in figure 14, where multiple copies of each component may be run in parallel (e.g. when training many robots at once).

In the first component outlined in algorithm 5, the problem distribution samples world configurations to pass to the demonstrator. At the start these come from a pre-defined distribution P over states; as training progresses the system prioritizes pre-mistake configurations identified by the human. This component also trains a classifier C_θ over the human state-label pairs that assigns a probability $p \in [0, 1]$ as to whether or not a mistake will occur in the next several timesteps. Concretely, a value of 1 is assigned in the training data to the step T labelled by a human as anticipating a mistake. Succeeding timesteps are given a value of 0, while preceding steps $t < T$ are assigned a value of e^{t-T}; this is meant to reflect an exponential distribution. While training, the classifier is used to label the N states of a policy rollouts with the highest probability of mistakes occurring. These states are then passed to the demonstrator, with priority beneath those directly labelled by the human but above those from the initial sampler. The classifier is trained identically to the policy, as per algorithm 7.

Algorithm 5 Problem Sampling

Require: Shared queues Q_{label}, Q_{prob}, Q_{roll}
Require: Shared data D_{class}
Require: Initial problem distribution P
Require: Classifier C_θ with shared memory θ
while not terminated **do**
  if Q_{label} not empty **then**
    $\{x_0\}, t, \text{label} \leftarrow \text{pop}(Q_{label})$
    append(D_{class}, (x, t, label))
  else
    ## Samples P if C_θ is untrained
    $\{x_i\}_0^N \sim C_\phi(P, Q_{roll})$
  for $x \in \{x_i\}$ **do**
    push(Q_{prob}, x)

Algorithm 6 Planning

Require: Shared queue Q_{prob}
Require: Shared dataset D_{train}
Require: Expert Demonstrator $PLAN$
while not terminated **do**
  $x_0 \leftarrow \text{pop}(Q_{prob})$
  $\tau \leftarrow PLAN(x_0)$
  append(D_{train}, τ)

Algorithm 7 Policy Training

Require: Shared dataset D_{train}
Require: Shared queue Q_{roll}
Require: Imitation learning procedure $TRAIN()$
Initialize policy π_ϕ
while not terminated **do**
  $\{\tau_i\}_0^N \sim D_{train}$
  $TRAIN(\{\tau_i\}_0^N, \phi)$
  ## Collects rollouts
  $\{\hat{\tau}_i\}_0^M \leftarrow \pi_\phi$
  push($Q_{roll}, \{\hat{\tau}_i\}_0^M$)

The next component shown in algorithm 6 simply takes these initial states, and plans and executes trajectories to accomplish the goal. These trajectories are then passed to the policies for supervised training as per algorithm 7; while the system makes no assumption on the imitation learning procedure used, in this work direct supervised learning was applied to the demonstrations.

In the final component, the human observes rollouts of the partially trained policies and identifies states that happen near-to but before a mistake occurs, as per 8. This creates a form of asynchronous intervention - the human does not directly step in when the robot should ask for help, but instead identifies when the robot *should* have asked for help. This places less pressure on the human and allows greater flexibility for the interface of interaction.

Algorithm 8 Human Labelling

Require: Shared queues Q_{label}, Q_{roll}
while not terminated **do**
  $\tau \leftarrow \text{pop}(Q_{roll})$
  $\{(x_i, t_i, \text{label}_i)\}_0^N \leftarrow \text{LABEL}(\tau)$
  push($Q_{label}, \{(x_i, t_i, \text{label}_i)\}_0^N$)

2.3.3 User Interface

The interface shown to the human is outlined in figure 15. At run-time, the human views video segments from trajectories sampled from the partially trained policy. There are five possible inputs, which are entered as keyboard commands. The right arrow key $\rightarrow$ tells the system to move the video forward in time, showing later timesteps in the trajectory. The left arrow key $\leftarrow$ tells the system to move the video backward in time, showing earlier timesteps in the trajectory. The space bar tells the system the current video segment contains a mistake; i.e. the first timestep is before the mistake but

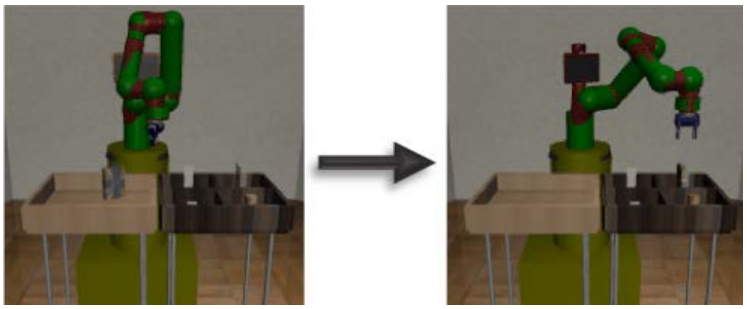


Figure 16: The task used for evaluation. The policy outputs joint velocities to grasp the object and then place it in the bin.

the final is after. The user may also press u, to indicate uncertainty for the current segment (at which point a new trajectory is loaded) or q to exit the program.

Upon labelling a mistake, the initial timestep t of the current video is used to record a tuple (x_t, during). In addition, T preceding timesteps $\{t_i\}_1^T$ spaced evenly (for this work, 6 timesteps apart) are also used to record tuples (x_{t_i}, after). All tuples then pass through to the problem distribution. The intent of this approach is to model the human label not as a hard filter on the timestep to sample, but rather as a distribution over timesteps at which failure may be anticipated. This allows for greater robustness to mistakes in the labelling and provides denser supervision to the system.

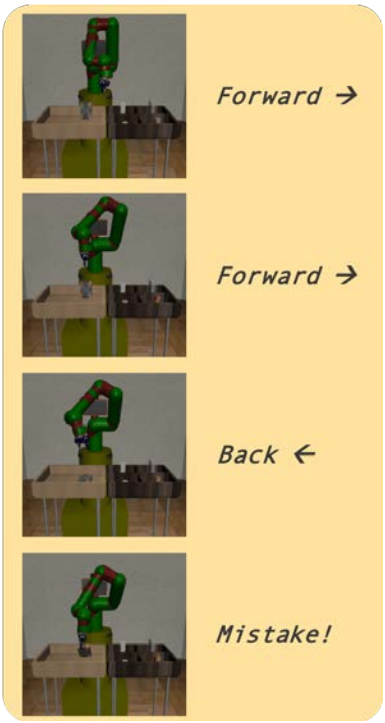


Figure 15: Representation of user interface. (1) No mistake has happened, user should go forward. (2) still no mistake. (3) Mistake has already occurred, go back. (4) Label mistake.

2.4 Evaluation

We evaluate our approach when the policy trains using data labelled offline, with data labelled online with training, and when a classifier is additionally trained alongside online labelling. For the offline variant we provide details of a small user study where the system was provided to untrained users; for the online variants trials were conducted by the author.

All experiments were conducted using the Pick-And-Place task from Robosuite [38] with the Sawyer robot and only the cereal object; in this task the object is randomly initialized on the table at left and the robot must transfer it to the corresponding bin on right, as shown in figure 16. The robot observed state information about both joints and object and target placement. The actions consisted of velocities on the robot's 7 joints plus a signal to open or close the gripper. This task was chosen due its difficulty, as it requires precise orientation of the gripper to grasp the object without knocking it over and the robot design makes it impossible to grasp the object once knocked over. The planning domain had four actions: move-to-grasp, grasp, move-to-putdown, and putdown.

All plots reflect 500-point rolling averages, with error bands showing standard deviation. Data was aggregated across multiple random seeds for each training condition when feasible.

2.4.1 Setup

Our expert demonstrators consist of an integrated task and motion planner using the FastForward planner for task planning and a variant of the trajectory optimization procedure outlined by Hadfield-

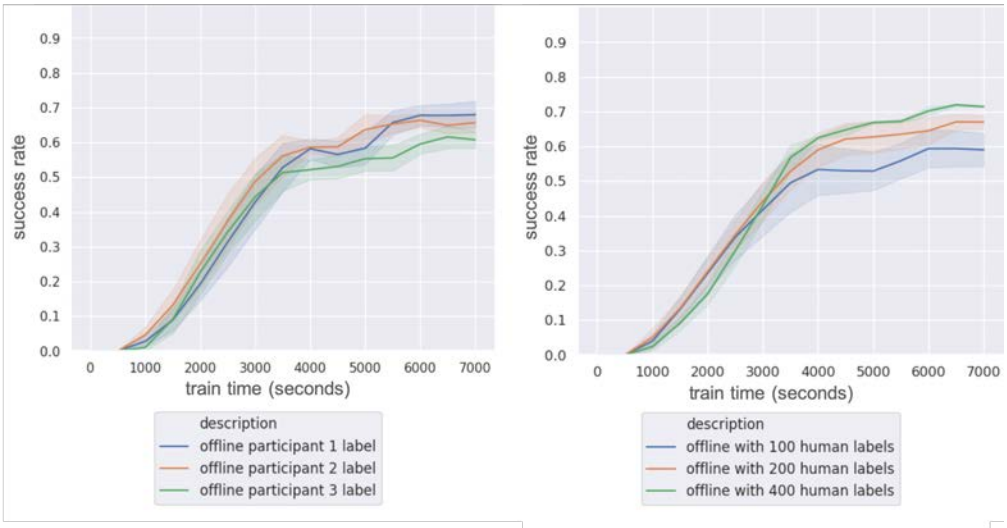


Figure 17: Training from offline data with three random seeds. Left: Comparison of results from each participant using 400 labelled examples. Right: Comparison of results with increasing number of labelled examples.

Menell et al. [36] for motion planning. All policies and the classifier consist of fully-connected neural networks of two hidden layers with 64 units each. RELU activations are used on the hidden layers, while linear activations are used on the outputs. Training was conducted using 18 processes for planning and 12 processes for policy rollouts, along with a dedicated process for training each neural network.

2.4.2 Offline Interaction

User Study Setup

For the first set of trials, where labelling happened offline from training, a small user study was conducted to examine the ability of non-experienced end users to shape the problem distribution. Three participants (1 Female and 2 Male) were recruited. Each participant was asked to download a labelling script and a set of introductory videos. The participants first viewed seven examples of the robot successfully completing the task, in order to familiarize themselves with the objective. They were then asked to review sixteen examples of trajectories where the robot had made a mistake, accompanied by the author's explanation of what mistake had occurred. Finally, the participants viewed sixteen example video segments that had been labelled (forward, backward, mistake) by the author. The purpose of this setup was to familiarize the participants to the task, the nature of failures that could occur, and the interface.

In these trials, a policy had been trained ahead of time without human intervention, and videos of trajectories collected with the learned policy saved alongside the labelling script. The participants ran the script on their local machines. They were shown videos segments containing 12 policy steps, initially at the beginning of a randomly sampled trajectory. The participants could use the arrows keys to shift the videos forward or back by 6 steps, and the space bar to indicate a mistake occurred. Up to six states were sub-sampled per trajectory. A new video was shown to the user upon pressing the space bar, shifting the time frame outside the trajectory, or pressing the u key to indicate uncertainty. The script saved both the states and labels to a file that was then sent to the authors. Training was then restarted, using these states to sample the initial demonstrations. For these variants no classifier was trained.

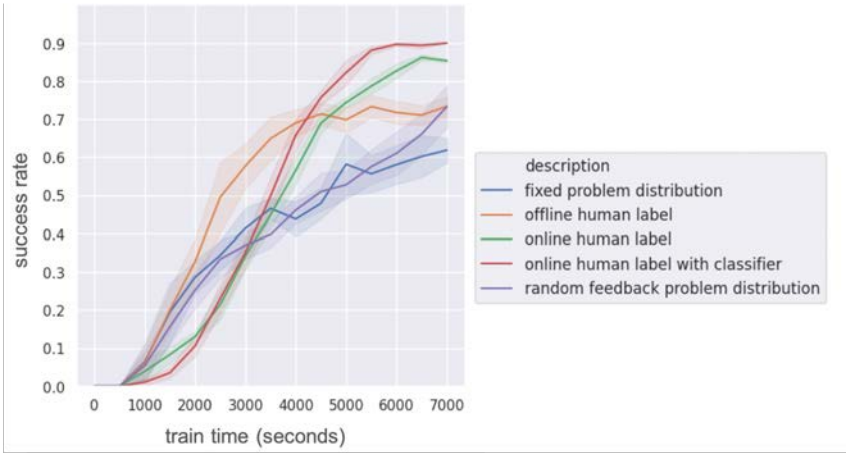


Figure 18: Comparison of five variants of the system; all variants except those run online were run with five random seeds.

Analysis

To compare how training differed between participants, each set of labeled states was truncated to 400 examples. As seen in figure 17, the system varied slightly in its ability to train from different participants, although not significantly so.

To compare how much data is needed to improve performance, training (for 3 random seeds) was also run using 100, 200, and 400 randomly chosen labels across participants. There was a clear improvement in performance as the number of available examples increased, as shown in figure 17.

For comparison against the other variants, training was conducted using a mixture of the participants' labelled data. As seen in figure 18, the offline data allowed the system to outperform the variants without human interaction but did not match the performance of those variants with online interaction.

2.4.3 Online Interaction

For the next trial, a user ran the labelling interface while training occurred, but the system trained no classifier from the data. For this evaluation, only the author performed labelling due to the time consumption involved (requiring two hours). Only a single trial was run for the same reason.

As shown in figure 18, the use of online labelling outperformed both the baseline variants without interaction and the offline variants. While promising, these results are confounded by the shift in labellers and hence further trials are needed to disambiguate the influence of task familiariity from online interaction.

2.4.4 Online with Trained Classifier

For the final trial, a user ran the labelling interface while training occurred, and the system trained the classifier simultaneously. For this evaluation, only the author performed labelling due to the time consumption involved (requiring two hours). Only a single trial was run for the same reason.

As shown in figure 18, the use of online labelling with the classifier outperformed all other variants. While promising, these results again are confounded by the shift in labellers and hence further trials are needed to disambiguate the influence of task familiarity from online interaction. They do however support the claim that the classifier provides benefit beyond the human labels.

2.5 Discussion and Future Work

While promising, this work is only preliminary and leaves unexplored many questions about how human mistake anticipation may shape learning. In particular, a greater user study for both offline and online variants would help disambiguate potential confounds in the evaluation. Due to the

COVID19 pandemic, users ran labelling on their local machines; given greater control over the users' environments, it would also be possible to measure and quantize the rate at which labelling occurs and measure training as a function of that rate. It would also to be of interest to measure the users' perception of the robot in both anthropomorphic and functional aspects over the course of training and identify any shifts that occur.

This work also took a simplest-first approach, in particular examining only imitation learning. Of interest would be in extending this to more complex learning procedures such as inverse reinforcement learning, and in using the trained classifiers to generalize to training in previously unseen tasks without human interaction.

References

[1] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016. URL http://jmlr.org/papers/v17/15-522.html.

[2] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization, 2017.

[3] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang. Solving rubik's cube with a robot hand, 2019.

[4] T. Yu, P. Abbeel, S. Levine, and C. Finn. One-shot hierarchical imitation learning of compound visuomotor tasks, 2018.

[5] A. C. Li, C. Florensa, I. Clavera, and P. Abbeel. Sub-policy adaptation for hierarchical reinforcement learning, 2020.

[6] L. P. Kaelbling and T. Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477, 2011. doi:10.1109/ICRA.2011.5980391.

[7] T. Ren, G. Chalvatzaki, and J. Peters. Extended task and motion planning of long-horizon robot manipulation, 2021.

[8] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 639–646, 2014. doi:10.1109/ICRA.2014.6906922.

[9] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning, 2020.

[10] S. Levine and V. Koltun. Guided policy search. In S. Dasgupta and D. McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1–9, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/levine13.html.

[11] E. Groshev, A. Tamar, S. Srivastava, and P. Abbeel. Learning generalized reactive policies using deep neural networks. *CoRR*, abs/1708.07280, 2017. URL http://arxiv.org/abs/1708.07280.

[12] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. JMLR Workshop and Conference Proceedings. URL http://proceedings.mlr.press/v15/ross11a.html.

[13] 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. pages 447–454, 05 2016. doi:10.1109/ICRA.2016.7487165.

[14] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

[15] R. Sutton, D. Precup, and S. Singh. Intra-option learning about temporally abstract actions. pages 556–564, 01 1998.

[16] H. M. Van, O. S. Oguz, Z. Zhou, and M. Toussaint. Guided sequential manipulation planning using a hierarchical policy. RSS Workshop on Learning (in) Task and Motion Planning, 2020.

[17] A. Tamar, S. Levine, and P. Abbeel. Value iteration networks. *CoRR*, abs/1602.02867, 2016. URL http://arxiv.org/abs/1602.02867.

[18] M. Garnelo, K. Arulkumaran, and M. Shanahan. Towards deep symbolic reinforcement learning, 2016.

[19] B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. *CoRR*, abs/1709.05448, 2017. URL http://arxiv.org/abs/1709.05448.

[20] M. Okada, L. Rigazio, and T. Aoshima. Path integral networks: End-to-end differentiable optimal control, 2017.

[21] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4732–4741. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/srinivas18b.html.

[22] S. Nasiriany, V. H. Pong, S. Lin, and S. Levine. Planning with goal-conditioned policies, 2019.

[23] 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, 2021.

[24] S. Christen, L. Jendele, E. Aksan, and O. Hilliges. Learning functionally decomposed hierarchies for continuous control tasks with path planning, 2020.

[25] B. Kim, L. P. Kaelbling, and T. Lozano-Pérez. Learning to guide task and motion planning using score-space representation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2810–2817, 2017. doi:10.1109/ICRA.2017.7989327.

[26] A. Wells, N. Dantam, A. Shrivastava, and L. Kavraki. Learning feasibility for task and motion planning in tabletop environments. *IEEE Robotics and Automation Letters*, PP:1–1, 01 2019. doi:10.1109/LRA.2019.2894861.

[27] B. Kim, L. P. Kaelbling, and T. Lozano-Perez. Adversarial actor-critic method for task and motion planning problems using planning experience. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019. URL http://lis.csail.mit.edu/pubs/kim-aaai19.pdf.

[28] J. Ichnowski, Y. Avigal, V. Satish, and K. Goldberg. Deep learning can accelerate grasp-optimized motion planning. *Science Robotics*, 5(48), 2020. doi:10.1126/scirobotics.abd7710. URL https://robotics.sciencemag.org/content/5/48/eabd7710.

[29] C. Paxton, V. Raman, G. D. Hager, and M. Kobilarov. Combining neural networks and tree search for task and motion planning in challenging environments. *CoRR*, abs/1703.07887, 2017. URL http://arxiv.org/abs/1703.07887.

[30] K. Kase, C. Paxton, H. Mazhar, T. Ogata, and D. Fox. Transferable task execution from pixels through deep planning domain learning, 2020.

[31] D. Driess, J.-S. Ha, R. Tedrake, and M. Toussaint. Learning geometric reasoning and control for long-horizon tasks from visual input. 2021.

[32] J. Ho and S. Ermon. Generative adversarial imitation learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/cc7e2b878868cbae992d1fb743995d8f-Paper.pdf.

[33] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 1, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138385. doi:10.1145/1015330.1015430. URL https://doi.org/10.1145/1015330.1015430.

[34] A. Dragan. Legible robot motion planning. 2015.

[35] J. Hoffmann. Ff: The fast-forward planning system. *AI Magazine*, 22:57–62, 09 2001.

[36] D. Hadfield-Menell, C. Lin, R. Chitnis, S. Russell, and P. Abbeel. Sequential quadratic programming for task plan optimization. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 5040–5047. IEEE Press, 2016. doi:10.1109/IROS.2016.7759740. URL https://doi.org/10.1109/IROS.2016.7759740.

[37] Y. Tassa, S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, and N. Heess. dm_control: Software and tasks for continuous control, 2020.

[38] Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[40] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[41] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning*, 2018.

[42] A. Mandlekar, F. Ramos, B. Boots, F. Li, A. Garg, and D. Fox. IRIS: implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. *CoRR*, abs/1911.05321, 2019. URL http://arxiv.org/abs/1911.05321.

[43] J. Mišeikis, P. Caroni, P. Duchamp, A. Gasser, R. Marko, N. Mišeikienė, F. Zwilling, C. de Castelbajac, L. Eicher, M. Früh, and H. Früh. Lio-a personal robot assistant for human-robot interaction and care applications. *IEEE Robotics and Automation Letters*, 5(4):5339–5346, 2020. doi:10.1109/LRA.2020.3007462.

[44] R. Shah and S. Nagaraja. Privacy with surgical robotics: Challenges in applying contextual privacy theory, 2019.

[45] e. Ackerman. Full page reload, Jan 2020.

[46] F. Xue, H. Tang, Q. Su, and T. Li. Task allocation of intelligent warehouse picking system based on multi-robot coalition. *KSII Trans. Internet Inf. Syst.*, 13:3566–3582, 2019.

[47] A. Gupta, A. Murali, D. Gandhi, and L. Pinto. Robot learning in homes: Improving generalization and reducing dataset bias, 2018.

[48] G. Kazhoyan, S. Stelter, F. K. Kenfack, S. Koralewski, and M. Beetz. The robot household marathon experiment, 2020.

[49] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020. ISSN 2169-3536. doi:10.1109/access.2020.2983149.

[50] Y. Huang and Y. Chen. Autonomous driving with deep learning: A survey of state-of-art technologies, 2020.

[51] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. Paixão, F. Mutz, L. Veronese, T. Oliveira-Santos, and A. F. D. Souza. Self-driving cars: A survey, 2019.

[52] G. Dulac-Arnold, D. Mankowitz, and T. Hester. Challenges of real-world reinforcement learning, 2019.

[53] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, Nov 2017. ISSN 1053-5888. doi:10.1109/msp.2017.2743240.

[54] L. Choshen, L. Fox, Z. Aizenbud, and O. Abend. On the weaknesses of reinforcement learning for neural machine translation, 2020.

[55] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009. ISSN 0921-8890. doi:https://doi.org/10.1016/j.robot.2008.10.024.

[56] Y. Cui and S. Niekum. Active reward learning from critiques. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6907–6914, 2018.

[57] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety, 2016.

[58] M. Cakmak and A. L. Thomaz. Designing robot learners that ask good questions. In *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 17–24, 2012. doi:10.1145/2157689.2157693.

[59] R. Cohn, E. Durfee, and S. Singh. Comparing action-query strategies in semi-autonomous agents. volume 2, pages 1287–1288, 01 2011.

[60] M. Kwon, S. H. Huang, and A. D. Dragan. Expressing robot incapability. *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, Feb 2018. doi:10.1145/3171221.3171276. URL http://dx.doi.org/10.1145/3171221.3171276.

[61] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang. A survey of deep active learning, 2020.

[62] K. Brantley, A. Sharaf, and H. D. I. au2. Active imitation learning with noisy guidance, 2020.

[63] B. Knudsen and U. Liszkowski. Eighteen- and 24-month-old infants correct others in anticipation of action mistakes. *Developmental Science*, 15(1):113–122, 2012. doi:https://doi.org/10.1111/j.1467-7687.2011.01098.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-7687.2011.01098.x.

[64] C. R. Crowell, J. C. Deska, M. Villano, J. Zenk, and J. T. Roddy Jr. Anthropomorphism of robots: Study of appearance and agency. *JMIR Hum Factors*, 6(2):e12629, May 2019. ISSN 2292-9495. doi:10.2196/12629. URL http://humanfactors.jmir.org/2019/2/e12629/.

[65] L. Onnasch and E. Roesler. Anthropomorphizing robots: The effect of framing in human-robot collaboration. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 63 (1):1311–1315, 2019. doi:10.1177/1071181319631209. URL https://doi.org/10.1177/1071181319631209.

Appendix A: Experiment Settings

Task Policy Details

In both domains, the task policy is a single neural network with RELU activations on the hidden layers and a softmax activation on the final layer. The output consists of multiple heads, each of which represent a one-hot encoded vector. The first head is always dedicated to the choice of action (e.g. $place$) and the remaining heads specify the selected parameterization (e.g. $obj1$, $targ2$). Each head is trained with a cross-entropy loss, and the overall loss is the sum of these individual losses. The learning rate was set to 2×10^{-4}. All networks had L2 regularization.

For the 2d pick-place domain, the task policy produces a three-dimensional one-hot vector for the choice of action schema (moveto-grasp, transfer, or place), a one-hot vector for choice of object (same dimension as the number of objects), and an eight-dimensional one-hot vector for choice of end target.

For the Robosuite domain, the task policy produces a four-dimensional one-hot vector for the choice of action schema (moveto, grasp, moveholding, or putdown) and a one-hot vector for choice of object (same dimension as the number of objects).

Experiment	# conv layers	filter size	# filters	# fc layers	fc dim	reg
Robosuite Pick-Place 1 Obj	0	n/a	n/a	2	64	10^{-4}
Robosuite Pick-Place 2 Obj	0	n/a	n/a	2	64	10^{-4}
Robosuite Pick-Place 4 Obj	0	n/a	n/a	2	64	10^{-4}
2d Pick-Place 1 Obj	0	n/a	n/a	2	96	10^{-4}
2d Pick-Place 2 Obj	0	n/a	n/a	2	96	10^{-4}
2d Pick-Place 5 Obj	0	n/a	n/a	2	96	10^{-4}
2d Pick-Place 1 Obj Vision	2	5	32	2	96	10^{-4}
2d Pick-Place 2 Obj Vision	2	5	32	2	96	10^{-4}
2d Pick-Place 3 Obj Vision	2	5	32	2	96	10^{-4}

Motion Policy Details

In both domains, stage 2 of the motion policy involves a fully-connected neural network with RELU activations on the hidden layers and no output activation. The network is trained via the standard L2 loss with learning rate was set to 2×10^{-4}. All networks had L2 regularization.

For the 2d pick-place domain, stage 2 of the motion policy is a single neural network that outputs velocity controls for the robot's x, y, and rotational position and the next gripper command.

For the Robosuite domain, stage 2 of the motion policy is a set of four neural networks (one for each action) that outputs velocity controls for the robot's 7 joints and the next gripper command.

In most experiments, stage 1 of the motion policy uses the output of the task policy to select relevant indices of the state vector to pass through to stage 2. For the 2d pickplace vision experiments, stage 1 of the motion policy is a convolutional network with the same architecture as the task policy, but the output is a single vector with linear activation and trained via L2 loss.

Experiment	separate option networks	# fc layers	fc dim	reg
Robosuite Pick-Place 1 Obj	Yes	2	64	10^{-6}
Robosuite Pick-Place 2 Obj	Yes	2	64	10^{-6}
Robosuite Pick-Place 4 Obj	Yes	2	64	10^{-6}
2d Pick-Place 1 Obj	No	2	64	10^{-4}
2d Pick-Place 2 Obj	No	2	64	10^{-4}
2d Pick-Place 5 Obj	No	2	64	10^{-4}
2d Pick-Place 1 Obj Vision	No	2	64	10^{-4}
2d Pick-Place 2 Obj Vision	No	2	64	10^{-4}
2d Pick-Place 3 Obj Vision	No	2	64	10^{-4}