Multi-Task Hierarchical Imitation Learning for Home Automation

Roy Fox*[1], Ron Berenstein*[2], Ion Stoica[1], and Ken Goldberg[1,2,3]

***Abstract*— Control policies for home automation robots can be learned from human demonstrations, and hierarchical control has the potential to reduce the required number of demonstrations. When learning multiple policies for related tasks, demonstrations can be reused between the tasks to further reduce the number of demonstrations needed to learn each new policy. We present HIL-MT, a framework for Multi-Task Hierarchical Imitation Learning, involving a human teacher, a networked Toyota HSR robot, and a cloud-based server that stores demonstrations and trains models. In our experiments, HIL-MT learns a policy for clearing a table of dishes from 11.2 demonstrations on average. Learning to set the table requires 19 new demonstrations when training separately, but only 11.6 new demonstrations when also reusing demonstrations of clearing the table. HIL-MT learns policies for building 3- and 4-level pyramids of glass cups from 8.2 and 5 demonstrations, respectively, but reusing the 3-level demonstrations for learning a 4-level policy only requires 2.7 new demonstrations. These results suggest that learning hierarchical policies for structured domestic tasks can reuse existing demonstrations of related tasks to reduce the need for new demonstrations.**

I. Introduction

Advances in imitation learning (IL) from demonstrations provided by human teachers kinesthetically or via teleoperation, have the potential to expand the set of industrial, domestic, and agricultural tasks that robots can perform. IL algorithms take as input a set of demonstrations, and output a learned model representing the control policy. Such a policy maps a sequence of robot sensory observations into a sequence of robot control actions. As providing demonstrations by human control of a physical robot can be costly and time-consuming, we seek to reduce the number of demonstrations needed to learn control policies.

Hierarchical control policies allow the teacher to provide more information with each physical demonstration, by annotating it with the hierarchical structure of the task, namely the recursive decomposition of the task into smaller and smaller sub-tasks [1]–[3]. A short-term policy, called a *skill*, can be trained to perform each sub-task in the hierarchy, and the set of trained skills can then be assembled into a complete control policy for the task.

Rather than viewing the abundance of robot tasks as a challenge, Cloud and Fog Robotics view it as an opportunity [4], [5]. Automation tasks may differ in initial conditions, system dynamics (due to variability in both robotic platforms and their environments), and task success criteria, but many sets of tasks also share similarities. For example, the tasks of clearing and setting a table both require skills for reaching and grasping dishes. Other aspects of the two policies, such as deciding where to place dishes, must differ due to the different success criteria of the tasks. The key insight in multi-task learning is that successful control for one task may be informative for other tasks.

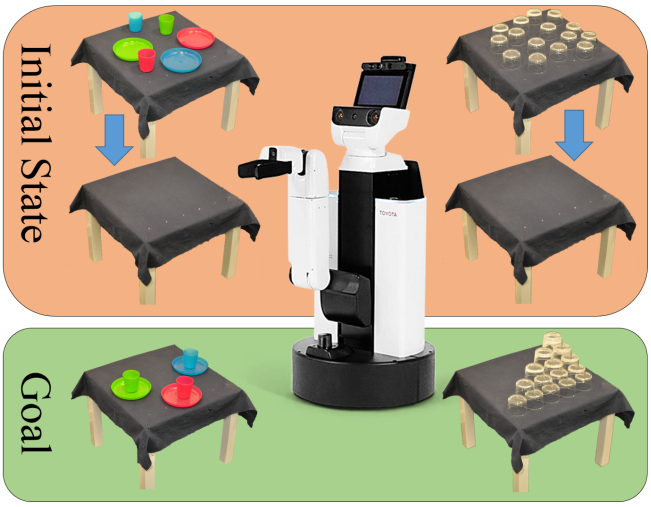

Fig. 1. The Toyota human support robot (HSR) performing two tasks, setting a table and building a pyramid of glass cups. We use the proposed HIL-MT framework to learn an autonomous hierarchical control policy that sets the table (left) from 19 human demonstrations annotated with the task's hierarchical structure. Reusing demonstrations of clearing the table reduces the required number of new demonstrations of setting the table to 11.6. HIL-MT can learn a policy that builds a 5-level pyramid (right) from 35 total demonstrations of 2, 3, and 4 levels, without additional 5-level demonstrations (zero-shot).

We present HIL-MT, a framework for multi-task hierarchical imitation learning, involving: (1) a human teacher, who demonstrates each new task on the robot; (2) a networked Toyota Human Support Robot (HSR), which uploads the demonstrations to a server during the training phase, and queries the server for the trained policy during task execution; and (3) a cloud-based machine-learning-as-a-service (MLaaS) server, which stores the demonstrations of past tasks and trains new hierarchical policies. In our experiments, demonstrations are provided by human teleoperation of the HSR, in collaboration with Python-programmed skills.

We consider three alternative modes of multi-task imitation learning, training each skill of the new policy from: (1) only past demonstrations of related tasks; (2) only demonstrations of the new task; or (3) the joint set of past and new demonstrations. We attempt to use each one of the three multi-task modes to learn each skill, and validate it on a held-out set of demonstrations. The skill is considered

The AUTOLab at UC Berkeley; autolab.berkeley.edu
*Equal contribution
[1]EECS, [2]CITRIS, [3]IEOR, UC Berkeley, CA, USA
Corresponding author: royf@berkeley.edu

successfully trained if it accurately predicts the decisions made by the human teacher in the held out demonstrations.

Which of the three multi-task modes requires the fewest demonstrations to successfully train each skill? We hypothesize that the answer varies among the skills, due to the relative benefits of each mode. We analyze examples of these benefits in Section V, and present experimental results suggesting that simply selecting the same mode for all skills can be less data efficient than per-skill mode selection.

We evaluate HIL-MT on two multi-task domains, using the HSR robot augmented with pre-trained vision modules (detailed in Section V). HIL-MT learns a policy for clearing a table of dishes from 11.2 demonstrations on average. Learning to set the table requires 19 new demonstrations when training separately, but only 11.6 new demonstrations when also reusing demonstrations of clearing the table. For the tasks of building pyramids of glass cups of a given number of levels, HIL-MT learns separate policies for building 3- and 4-level pyramids from 8.2 and 5 demonstrations, respectively. Reusing the 3-level demonstrations reduces the required number of new demonstrations for learning a 4-level policy to 2.7. These results suggest that learning hierarchical policies for structured domestic tasks can reuse existing demonstrations of related tasks to reduce the need for new demonstrations.

The contributions of this work are:

1) HIL-MT, a multi-task hierarchical imitation learning framework, involving a computer-assisted human teacher, a networked robot, and a cloud-based learning server. An implementation of the server and the robot-facing client is available at https://github.com/BerkeleyAutomation/HIL-MT.
2) Experimental data from two multi-task domains on the Toyota HSR robot, clearing and setting a table (40 demonstrations of each task, each demonstration manipulating between 2 and 6 dishes), and building a pyramid of glass cups (a total of 35 demonstrations, arranging a total of 170 cups in 2-, 3-, and 4-level pyramids), in which our framework successfully learned hierarchical policies with high data efficiency, and demonstrated the benefits of multi-task hierarchical learning.

II. Related Work

Imitation learning (IL) has been a popular and successful framework for training control policies from teacher demonstrations [6]–[10] or from annotation of correct control during execution of the learner policy [11]–[13]. The policy is often augmented with a memory state [14] so that it can take actions that depend on hidden state features that were revealed in past observations. In contrast, the internal structure of the teacher policy is rarely considered in IL settings to explicitly inform the demonstrations or the learning process.

Structure in the control model, and hierarchical control in particular, can assist in imitation learning [15], [16] and in multi-task learning [17]–[19]. Many frameworks of hierarchical control [3], [20]–[24] consider tasks as consisting of simpler sub-tasks, or *skills*, that need to be performed in sequence, each skill terminating for the next to begin. Hierarchical control allows the teacher to make the hierarchical structure explicit in demonstrations, to give a more informative teaching signal from which the policy can be learned with higher data efficiency [1]–[3], [25], [26]. While hierarchical control policies can be trained from weaker signals, such as rewards [21], [27]–[29], annotating demonstrations with the entire execution sequence of each skill in the hierarchy facilitates more efficient learning by decomposing the hierarchical imitation learning (HIL) problem into independent IL of individual skills [1], [3], [30]. This is particularly important in robot learning, where annotation of demonstrations can be far less costly and risky than physical demonstrations, which require expensive hardware and risk breakage of the robot and of the objects it manipulates. Compared with reinforcement learning (RL) from reward signals, where even short-horizon skills require months or even years of costly and risky interaction with the environment [31], [32], HIL with annotated demonstrations has the potential to train longer-horizon policies from few hours of safer human demonstrations.

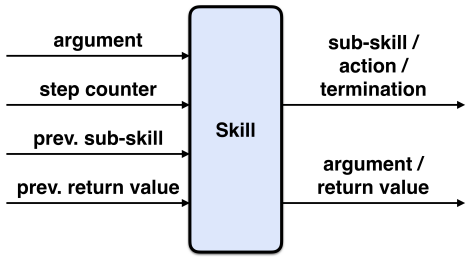

Fig. 2. The input–output interface of a skill in a hierarchical policy. In each step, the skill selects an operation (calling a sub-skill, taking a control action, or terminating to return to the caller skill) and generates either an argument for the sub-skill or action, or a return value for the caller. The input for each skill step is the skill's own argument (generated once by the caller and fixed for the skill's entire execution), a step counter that is incremented by 1 with each step, and the identifier and return value of the previous sub-skill or action (except in the first step).

In RL and IL, hierarchies exhibit a number of benefits, including reduction of the computational complexity of long-horizon planning problems [27], [28], [33], [34], reduction of the sample complexity in robot imitation learning tasks [35], better generalization when the initial state has high variability [2], [36], [37], and the possibility of skill transfer between distinct tasks [25], [38].

Several hierarchical frameworks have been proposed [23], including Hierarchical Abstract Machines [20], the options framework [21], and MAX-Q [22]. In this work, we build on the Parametrized Hierarchical Procedures (PHP) framework [3], which in turn builds on Abstract Hidden Markov Models [27], [35], [39]. We extend the PHP skill interface of [3] by allowing each skill to accept an argument from its caller when execution starts, and pass a return value back to its caller when it terminates (Figure 2).

Earlier works in multi-task learning mostly studied transfer learning, where previously trained models help solve new tasks [40]. More recent works study shared learning methods,

where multiple tasks are solved jointly, either by learning shared representations [41]–[43], shared value functions [44], [45], or shared policies [18], [46], [47], or by distillation of multiple existing policies into one [17], [48], [49]. Particularly relevant to this work are [18], [49], although they only utilize 2-level hierarchical structures. In contrast to the meta-learning approach of [18], which requires data for a large number of tasks, and the distillation approach of [49], which requires pre-trained value networks for each task, our approach can reduce sample complexity using only human demonstrations of two or more related tasks, but requires these demonstrations to be annotated.

III. Problem Statement

A. Multi-Task Imitation Learning

We model the interface between the robot and its environment as a stochastic process of observations o_t in space $\mathcal{O}$ and actions a_t in space $\mathcal{A}$. The observation sequence is generated by a latent Markov process (POMDP) with state s_t in space $\mathcal{S}$, induced by the the initial state distribution $p(s_0)$ and the observation and transition distributions $p(o_t|s_t)$ and $p(s_{t+1}|s_t, a_t)$. The action sequence is generated by a control policy π, which upon seeing o_t updates the controller's internal state and outputs a_t. The interaction terminates in state s_T when the robot selects the termination action $a_T = \varnothing$.

In this work, all tasks involve the Toyota HSR robot, so that the interface $\langle \mathcal{O}, \mathcal{A} \rangle$ is fixed (detailed in Section V). The latent state space $\mathcal{S}$ and its dynamics p will generally change depending on the objects involved in each task. Each task $\mathcal{T}_i = \langle \mathcal{S}_i, p_i, R_i, u_i \rangle$ is defined by: (1) the state space $\mathcal{S}_i$; (2) the state dynamics p_i; (3) a task success indicator $R_i(s_{0:T})$, which is 1 on successful execution and 0 on failure; and (4) a real-vector specification u_i of the task, given as input to the policy. The objective is to learn a policy π_i for task $\mathcal{T}_i$ that achieves high success rate, $\mathbb{E}_{\pi_i}[R_i] \approx 1$, where the expectation is over the initial state and the system dynamics (due to noise in the robot's motion and sensing).

In imitation learning (IL), we assume the availability of a teacher with policy π^*, and use it to train the control policy π. A common IL setting is behavior cloning (BC), where the teacher demonstrates a set of complete traces $\mathcal{D}$ by rolling out π^* and recording observations and actions. Given $\mathcal{D}$, BC uses supervised learning to fit a model π to the training data, such that π would take the same actions as π^* given the same observations.

When learning to perform multiple tasks, applying BC independently to each task would be inefficient, since the demonstrations $\mathcal{D}_i$ of task $\mathcal{T}_i$ can be informative of policy π_j of task $\mathcal{T}_j$. In this work, we consider the setting of *online* multi-task learning, in which learning π_i starts after $\pi_1, \ldots, \pi_{i-1}$ are fully trained, and $\mathcal{D}_1, \ldots, \mathcal{D}_{i-1}$ are available through a cloud-based data-sharing service.

B. Hierarchical Control

We build on the hierarchical control framework of Parametrized Hierarchical Procedures (PHP) [3], in which a hierarchical control policy is represented as a set of skills, where each skill performs a specific behavior by breaking it down into a sequence of simpler motions, and calling a sequence of sub-skills to perform these motions.

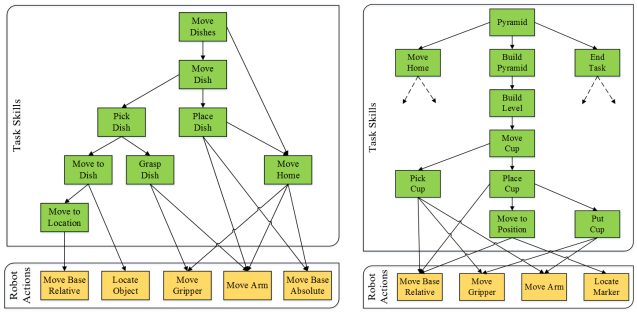

Fig. 3. Hierarchical skill structure of the Clear/Set Table tasks (left) and the Pyramid tasks (right). Each green block is a skill with the interface shown in Figure 2. Yellow blocks are HSR control actions, called with arguments that specify their goals. Execution of the hierarchical policy starts at the root skill, follows an edge down with each call to a sub-skill, and traces the edge back up when the sub-skill terminates.

A hierarchical policy starts by executing a *root* skill. Then the skill takes a sequence of operations, each operation being either: (1) calling another skill as a sub-skill; (2) taking a robot control action; or (3) terminating and returning to the caller (the skill that called this skill). This process is a walk on the controller's call-graph (Figure 3), where calling a sub-skill moves along a directed edge, and termination backtracks through that edge in the opposite direction. When the root terminates, the task is done and its success is evaluated.

Computer programs are executed in the same way, with procedures calling other procedures, and a call-stack keeping track of the active procedures to which we backtrack on termination. Inspired by this analogy, we extend PHP to allow skills to pass real-vector arguments to sub-skills and control actions, and to return real-vector values to the caller upon termination. As in PHP, we also maintain for each skill a step counter, and increase it by 1 with every operation (i.e., calling a sub-skill or taking an action) that the skill makes.

The input–output interface of a skill is summarized in Figure 2. A skill is simply a function from its argument, step counter, and the return value of the previous sub-skill or action; mapping to its next operation (sub-skill, action, or termination) and the argument or return value of that operation. This function can be implemented by code, or by a learned model.

Finally, we note that the robot's sensory observations are not provided directly as input to the skill. Instead, the observation o_{t+1} is treated as the return value of action a_t. This value can then be processed by each skill, and only the relevant information propagated up the hierarchy through return values. This feature is a form of attention that leverages the structure of the hierarchy. It allows keeping high-level skills simple by encapsulating low-level perceptual features in low-level skills, and only returning higher-level features. In processing perception, the levels of the hierarchy therefore perform similarly to the layers of a deep network.

IV. HIL-MT Framework

Our multi-task hierarchical imitation learning framework, HIL-MT, consists of: (1) a human teacher providing annotated demonstrations in collaboration with programmed skills; (2) a networked Toyota HSR robot used for demonstrations; and (3) a cloud-based server that stores training data, trains skills, and serves trained policies to the robot. In this section we describe how demonstrations are generated and how policies are learned.

A. Demonstrating and annotating hierarchical control

Learning a policy for a new task begins with a human expert identifying a decomposition of the task into a hierarchy of skills. We start by specifying a set of skills that break the task down into successively smaller sub-tasks (Figure 3). For each skill, we then specify the metadata needed by the framework (see for example Table I): (1) the arguments that the skill takes when called; (2) the values that the skill returns when it terminates; (3) the learnable model that implements the skill; and (4) the set of sub-skills and actions that the skill can call. The latter forms the edges of a call-graph of skills (Figure 3), called the *sketch* of the hierarchical policy.

A human teacher can now demonstrate the task on the physical robot. The hierarchical decomposition allows the teacher to annotate the demonstration with the hierarchical task structure, which improves the data efficiency of the learning algorithm. An annotated demonstration details the execution of each skill in the hierarchy (see for example Table II): which sub-skills and actions were called, what their arguments were, when the skill terminated, and what the return values were.

Annotated demonstrations can be entirely manual. This is a tedious process, but manageable when only few demonstrations are needed. In this work, we streamlined this process by coding most of the skills in Python, typically in under 10 lines of code per skill. Our Python infrastructure records the execution of each skill, i.e. its inputs and outputs. Skills that involve visual and physical features, such as those mapping from pixel positions to physical locations, are challenging to program. Instead, the infrastructure for these skills pauses the policy execution and waits for a human to teleoperate the robot. It then records the change in the robot's pose, and appended it to the demonstration as a call to the appropriate control actions, with the recorded arguments, e.g. "Move Base Rel (difference in base pose)".

B. Learning Method

HIL-MT is organized around a cloud-based machine-learning-as-a-service (MLaaS) server, that provides an application programming interface (API) for registering new task metadata, uploading annotated demonstrations, learning hierarchical policies, and serving learned policies to the robot for execution. After a task is registered by uploading the metadata of all its skills and linking to potentially related tasks, the networked robot uploads to the server each new demonstration of the task. The server adds the demonstration to the task's database, attempts to learn a hierarchical policy, and notifies the user whether all skills are successfully trained or more demonstrations are needed.

The learning method follows these steps:

1) Load the set of demonstrations of the new task $\mathcal{D}_{new}$ and of all past related tasks $\mathcal{D}_{past}$.
2) For each skill s, extract from these demonstration sets the steps taken by s, respectively $\mathcal{D}^s_{new}$ and $\mathcal{D}^s_{past}$. Each datapoint in these sets is given by an input–output pair (i, o) of the skill step function (Figure 2).
3) Split each set into a training set $\mathcal{T}^s_{new}$ and a validation set $\mathcal{V}^s_{new}$ (and similarly for $past$). This is repeated $k = 20$ times, with $|\mathcal{V}^s_{new}| = \frac{1}{k}|\mathcal{D}^s_{new}|$ for k-fold cross-validation.
4) Apply a learning algorithm $\mathbb{A}$ to learn three skill models: $m_{past} = \mathbb{A}(\mathcal{T}^s_{past})$, $m_{new} = \mathbb{A}(\mathcal{T}^s_{new})$, and $m_{joint} = \mathbb{A}(\mathcal{T}^s_{past} \cup \mathcal{T}^s_{new})$.
5) For each model m_j, compute the prediction of each datapoint $\hat{o} = m(i)$ in the corresponding validation set $\mathcal{V}^s_j$. Accept the first model m_j for which the $\mathrm{RMSE}(o, \hat{o}) < \alpha$, where RMSE is the root mean square error over all k validation sets, and α is the required accuracy.

This process repeats for each new demonstration, until all skills have a successfully validated model, and can be assembled into a hierarchical policy.

The server is implemented with a RESTful API in the flask Python library, using a SQLAlchemy backend over SQLite. The code is available at https://github.com/BerkeleyAutomation/HIL-MT.

V. Experiments

A. Experiment setup

We experiment with HIL-MT in two multi-task object manipulation domains on the Toyota Human Support Robot (HSR): clearing and setting a table, and building a pyramid of glass cups. Each experiment proceeds according to the protocol detailed in Section IV-B. Our hypothesis is that reusing demonstrations of related tasks reduces the required number of new demonstrations in some skills, but hinders learning other skills; therefore per-skill selection of the multi-task mode may help reduce the total number of demonstrations needed for training the hierarchical policy.

We defined an observation–action interface $\langle \mathcal{O}, \mathcal{A} \rangle$ for the HSR, with 4 actions that accept real-vector arguments:

- *Move Arm* (lift, arm flex, arm roll, wrist flex, wrist roll)
- *Move Gripper* (0=open; 1=close)
- *Move Base Abs* (X, Y, heading)
- *Move Base Rel* (ΔX, ΔY, rotation)

Move Base Abs and *Move Base Rel* use the robot's internal motion planner to reach a goal position and heading in the robot's initial and (respectively) current coordinate frame. There is no observation after these four actions. For each domain, we augmented the HSR with an active-perception action; see details for each domain below. The observation returned after this action is a real vector of domain-relevant features, extracted by a pre-trained module from an image captured by the robot's right head RGB camera.

B. Experiment 1: Clearing and setting a table

Domain specification. This domain consists of two tasks, Clear Table and Set Table (Figure 1). In each of the tasks, the robot starts facing a standard IKEA table (height=45cm). The robot's initial position and heading relative to the table is drawn at random from a Gaussian with mean and standard deviation (30±2cm, 0±2cm, 0±2°). On the table are scattered between 2 and 6 dishes, in arbitrary quantity and positions determined by the human experimenter (the same author conducted all of our experiments). The dishes are of two types, plates and cups, and of three colors, blue, green, and red. To the robot's left, in a fixed position relative to the first table, is a second table. In the Clear Table task, we place a large bin on the second table, and the success criterion is to place all dishes in the bin in any order. In the Set Table task, the dishes must be moved in a specific order of colors (blue, green, red), with the plate of that color moved first to a specific position on the left table (respectively (0, 20cm), (30cm, 20cm), and (15cm, 0) in the robot's initial coordinate frame), and the matching cup placed centered atop the plate.

The active-perception action Locate Dish is given the type (plate/cup/any) and color (blue/green/red/any) of an object to locate, and returns the type, color, and pixel position of the nearest matching object, if any were found. We programmed a vision module that: (1) downsizes the captured image to 920x690 with 4x4 bicubic interpolation; (2) feeds the image into the YOLO object detection algorithm [50] to find candidate bounding boxes, and filters them by the detected class (allowed classes: bowl, cup, frisbee, and orange); (3) classifies the dish type of each detected object by the pixel coordinates of its bounding box, using a classifier that we pre-trained on 264 labeled images; (4) extracts the dish color as the maximum channel of the mean RGB color in the bounding box; (5) returns the center of the lowest (i.e., physically nearest) bounding box matching the action's argument.

Policy metadata. We designed a hierarchical decomposition into sub-tasks that is shared by both Clear Table and Set Table policies. Table I details the metadata of each of the 8 skills in the hierarchy. The possible arguments:

- Task ID is 0 for Clear Table and 1 for Set Table.
- Dish # is a counter of the dishes moved so far.
- Dish type and color are described above.
- Segment # is a counter of the motion segments towards the dish; due to large errors in the HSR's motion over large distances, the teacher policy reaches for the dish in two segments: a longer rough approach, and a shorter, more accurate corrective motion.
- Pixel X and Y of the center of the detected bounding box, as described above.

The possible models:

- A quadratic model extracts quadratic features (the product of each pair of input variables) and performs logistic regression on these features to decide on the sub-skill / action / termination; and quadratic regression to decide on the argument / return value.

TABLE I

METADATA FOR SKILLS OF THE CLEAR/SET TABLE POLICIES

Skill name	Arguments	Returns	Model	Sub-skills
Move Dishes	Task ID	–	quadratic	Move Dish Move Home
Move Dish	Task ID Dish #	Success	quadratic or table	Pick Dish Place Dish
Pick Dish	Dish Type Dish Color	Dish Type Dish Color	quadratic	Move to Dish Grasp Dish
Place Dish	Task ID Dish Type Dish Color	–	quadratic	*Move Arm* *Move Base Abs* Move Home
Move to Dish	Dish Type Dish Color Segment #	Dish Type Dish Color	timed quadratic	*Locate Dish* Move to Loc
Grasp Dish	Dish Type Dish Color	Dish Type Dish Color	quadratic	*Move Gripper* *Move Arm*
Move Home	–	–	quadratic	*Move Gripper* *Move Arm* *Move Base Abs*
Move to Loc	Dish Type Dish Color Pixel X Pixel Y	Dish Type Dish Color	quadratic	*Move Base Rel*

TABLE II

ANNOTATED DEMONSTRATION OF SET TABLE, FIRST 11 SKILL STEPS

Active skill (arguments)	Step cnt.	Prev. sub-skill (return values)	Operation (arguments)
Move Dishes (1)	0	–	Move Home ()
Move Home ()	0	–	*Move Gripper* (0)
Move Home ()	1	*Move Gripper* ()	*Move Base Abs* (0cm,0cm,0°)
Move Home ()	2	*Move Base Abs* ()	*Move Arm* (45cm,0°,0°,−90°,−90°)
Move Home ()	3	*Move Arm* ()	∅ ()
Move Dishes (1)	1	Move Home ()	Move Dish (1,0)
Move Dish (1,0)	0	–	Pick Dish (plate,blue)
Pick Dish (plate,blue)	0	–	Move to Dish (plate,blue,0)
Move to Dish (plate,blue,0)	0	–	*Locate Dish* (plate,blue)
Move to Dish (plate,blue,0)	1	*Locate Dish* (plate,blue,589.5,361)	Move to Location (plate,blue,589.5,361)
Move to Location (plate,blue,589.5,361)	0	–	*Move Base Rel* (1.2cm,−30cm,0.1°)
...	...	...	...

- A table model is a simple look-up table for inputs seen in training; the most common output is used for unseen inputs.
- A selection between two models (here, quadratic or table) is performed by internal 20-fold cross-validation on the training set.
- A timed model learns a different model for each value of the step counter, to more easily learn a fixed sequence of sub-skills.

Regressors are optimized using the sklearn Python library. The accuracy required for successful validation of each of the skills is perfect prediction of the operation, and RMSE less than $\epsilon = 10^{-6}$ of the arguments; except for Move to Location which requires 1cm, 1cm, and 1° accuracy respectively in its first, second, and third arguments (interpreted as X, Y translation and rotation when calling Move Base Rel).

Results. Each task was demonstrated 40 times (see for example Table II). The demonstrations were then uploaded to the server in a random order, until validation was successful.

TABLE III

OF DEMONSTRATIONS TO LEARN SET TABLE AFTER CLEAR TABLE

$\mathcal{D}_{clear}$	$\mathcal{D}_{set}$	$\mathcal{D}_{clear} \cup \mathcal{D}_{set}$	Per-skill selection
Failed	19±0.3	Failed	**11.6**±0.25

Clear Table was successfully trained by 11.2 demonstrations on average.

Set Table was then trained with each of the three multi-task modes, and with a per-skill selection of the mode that trains it from the fewest demonstrations (Table III). Training all skills with only past data $\mathcal{D}_{clear}$ failed, because in the Place Dish skill, which puts a dish down in a position that depends on its type and color, the two tasks disagree on the correct position. Training with joint data $\mathcal{D}_{clear} \cup \mathcal{D}_{set}$ also failed, because a successful Place Dish skill for both tasks requires a higher-dimensional model than quadratic. Attempts to learn it with higher-dimensional polynomial regression failed due to insufficient data.

Training Set Table independently with only new data $\mathcal{D}_{set}$ was successful after 19 demonstrations on average. Allowing per-skill mode selection improved this to 11.6 demonstrations, because data from the Clear Table task reduced the new data requirements of the Move to Location skill, which has high sample complexity due to noise in its demonstrations. In detailed analysis of the multi-task learning modes that required the fewest demonstrations for each skill, we found that transfer from only past data was strictly best in Pick Dish, Grasp Dish, and Move Home, but these were also easily learnable in other modes and not a limiting factor.

Successful task execution was also evaluated physically on the HSR robot, and matched the validation results. Each trained policy was executed 10 times. Depending on lighting, object detection fails on about 5% of the calls to the active-perception action. Policies where some skills did not pass validation always failed their execution. Validated policies succeeded in every execution to clear or set all successfully detected dishes.

C. Experiment 2: Pyramid of glass cups

Domain specification. This domain consists of the tasks Pyramid<n>, in which the goal is to build a pyramid of glass cups of height n between 2 and 5 on a table (Figure 1). These tasks are very related to each other, but considered distinct because each has a different success criterion. The same two tables as in the previous domain were used, with the same initial position distribution relative to the robot. To keep the experiment focused on building the pyramid, cups were fed manually into the robot's gripper in a fixed position on one table, and two AR markers were taped to the wall behind the other table were the pyramid is built, 30cm above the table and 50cm apart.

The active-perception action Locate Markers takes no arguments, and returns the pixel positions of 4 corners of each of the two markers (a vector of 16 integers). We programmed a module that: (1) passes the RGB image through a binary threshold at 12.5% intensity; (2) feeds the black–white result into the ar_markers Python library, which returns the 8 pixel positions.

TABLE IV

METADATA FOR SKILLS OF THE PYRAMID POLICIES

Skill name	Arguments	Returns	Model	Sub-skills
Pyramid	Height	–	timed linear	Move Home Build Pyramid End Task
Move Home	–	–	timed linear	*Move Gripper* *Move Arm*
Build Pyramid	Height	–	linear	Build Level
End Task	–	–	timed linear	*Move Arm*
Build Level	Level # Level size	–	linear	Move Cup
Move Cup	Level #	–	timed linear	Pick Cup Place Cup
Pick Cup	–	–	timed linear	*Move Gripper* *Move Arm* *Move Base Rel*
Place Cup	Level #	–	timed linear	Move to Pos Put Cup *Move Base Rel*
Move to Pos	Level # Segment # Direction	–	timed linear	*Locate Markers* *Move Base Rel*
Put Cup	Level #	–	timed linear	*Move Gripper* *Move Arm*

Policy metadata. We designed a hierarchical decomposition into sub-tasks that is shared by all Pyramid<n> tasks. Table IV details the metadata of each of the 10 skills in the hierarchy. The possible arguments:

- Height is the number of levels in the pyramid.
- Level # is the index of the level currently being built.
- Level size is the number of cups in the current level.
- Segment # is the same as for Clear/Set Table.
- Direction is 0 for motion towards the table, 1 for away from the table.

The possible models are similar to the Clear/Set Table policies, except that only linear features of the input are used. The required validation accuracy for generated arguments is $\epsilon = 10^{-6}$, except for the Move to Position skill which requires 1cm accuracy in translation and 1° in rotation.

Results. The tasks Pyramid2, Pyramid3, and Pyramid4 were demonstrated 20, 10, and 5 times, respectively, so that a total of at least 50 cups were placed for each task. The demonstrations were then uploaded to the server in a random order, until validation was successful. Policies were successfully trained with an average of 16.5, 8.2, and 5 demonstrations, respectively.

Training Pyramid4 with access to the demonstrations of Pyramid3 reduced the number of new Pyramid4 demonstrations to 2.7. The challenging skill is Move to Position, which converts marker pixel positions (returned from Locate Markers) to relative physical translation and rotation. Due to human error in demonstrating this skill, it requires many demonstrations on which to average out the error, and reusing past demonstrations reduces the need for new ones. In this case we find that learning all skills from joint data performs as well as per-skill mode selection.

Similarly to Clear/Set Table, successful task execution was also evaluated physically on the HSR robot, and matched the validation results. In addition, we trained the skills from all demonstrations of Pyramid2–4, and tested the hierarchical policy on Pyramid5. Despite not seeing any demonstrations of this task (zero-shot), the policy generalized to successfully build a higher pyramid than seen in training. The reason is that our hierarchical decomposition ensures linear generalization of each skill in the hierarchy, although the entire policy is highly non-linear.

VI. Conclusion

We presented HIL-MT, a multi-task hierarchical imitation learning framework that allows training hierarchical robot control policies from human demonstrations that are annotated with hierarchical skill structure. Reusing demonstrations from related tasks can reduce the need for new demonstrations, but not equally for all skills: some skills benefit from the larger amount of joint data, while others are hurt by conflicting demonstrations. In these cases, per-skill selection of the multi-task mode further reduces the total number of demonstrations.

We experimented with HIL-MT in two multi-task domestic object manipulation domains, in which a Toyota HSR robot clears and sets a table, and builds a pyramid of glass cups. HIL-MT was successful in training interesting tasks with high data efficiency, and even achieved zero-shot generalization to unseen tasks. HIL-MT brings together multiple factors that have the potential to reduce the number of demonstrations: hierarchical imitation learning, annotated demonstrations, multi-task learning, and finally per-skill mode selection.

In future work, we will explore how the hierarchical abstraction of higher-level skills can allow sharing training data and trained skill models between distinct robotic platforms that have different form factors.

Acknowledgements

This research was performed at the AUTOLAB at UC Berkeley in affiliation with the Berkeley AI Research (BAIR) Lab, Berkeley Deep Drive (BDD), the Real-Time Intelligent Secure Execution (RISE) Lab, and the CITRIS "People and Robots" (CPAR) Initiative.

This research was supported in part by: the Scalable Collaborative Human-Robot Learning (SCHooL) Project, NSF National Robotics Initiative Award 1734633, and by the NSF ECDI Secure Fog Robotics Project Award 1838833.

The authors were supported in part by donations from Siemens, Google, Toyota Research Institute, Autodesk, Knapp, Honda, Intel, Comcast, Hewlett-Packard and by equipment grants from PhotoNeo, NVidia, and Intuitive Surgical.

References

[1] S. Reed and N. De Freitas, "Neural programmer-interpreters," 2016.

[2] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese, "Neural task programming: Learning to generalize across hierarchical tasks," *arXiv preprint arXiv:1710.01813*, 2017.

[3] R. Fox, R. Shin, S. Krishnan, K. Goldberg, D. Song, and I. Stoica, "Parametrized hierarchical procedures for neural programming," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJl63fZRb

[4] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[5] A. K. Tanwani, N. Mor, J. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, "A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering," in *Proc. of IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2019, pp. 1–8.

[6] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[7] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[8] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.

[9] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 21, 2017.

[10] Y. Li, J. Song, and S. Ermon, "Infogail: Interpretable imitation learning from visual demonstrations," in *Advances in Neural Information Processing Systems*, 2017, pp. 3815–3825.

[11] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[12] S. Ross and J. A. Bagnell, "Reinforcement and imitation learning via interactive no-regret learning," *arXiv preprint arXiv:1406.5979*, 2014.

[13] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, "Deeply aggrevated: Differentiable imitation learning for sequential prediction," *arXiv preprint arXiv:1703.01030*, 2017.

[14] K. Nguyen, "Imitation learning with recurrent neural networks," *arXiv preprint arXiv:1607.05241*, 2016.

[15] A. L. Friesen and R. P. Rao, "Imitation learning with hierarchical actions," in *Development and Learning (ICDL), 2010 IEEE 9th International Conference on*. IEEE, 2010, pp. 263–268.

[16] H. M. Le, N. Jiang, A. Agarwal, M. Dudík, Y. Yue, and H. Daumé III, "Hierarchical imitation and reinforcement learning," *arXiv preprint arXiv:1803.00590*, 2018.

[17] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust multitask reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4496–4506.

[18] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," *arXiv preprint arXiv:1710.09767*, 2017.

[19] J. Xu, Q. Liu, H. Guo, A. Kageza, S. AlQarni, and S. Wu, "Shared multi-task imitation learning for indoor self-navigation," *arXiv preprint arXiv:1808.04503*, 2018.

[20] R. Parr and S. J. Russell, "Reinforcement learning with hierarchies of machines," in *Advances in neural information processing systems*, 1998, pp. 1043–1049.

[21] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[22] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *JAIR*, vol. 13, no. 1, pp. 227–303, 2000.

[23] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341–379, 2003.

[24] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1470–1477.

[25] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," *arXiv preprint arXiv:1611.01796*, 2016.

[26] V. Murali, L. Qi, S. Chaudhuri, and C. Jermaine, "Neural sketch learning for conditional program generation," *arXiv preprint arXiv:1703.05698*, 2017.

[27] C. Daniel, H. Van Hoof, J. Peters, and G. Neumann, "Probabilistic inference for determining options in reinforcement learning," *Machine Learning*, vol. 104, no. 2-3, pp. 337–357, 2016.

[28] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture." in *AAAI*, 2017, pp. 1726–1734.

[29] R. Fox, S. Krishnan, I. Stoica, and K. Goldberg, "Multi-level discovery of deep options," *arXiv preprint arXiv:1703.08294*, 2017.

[30] C. Li, D. Tarlow, A. L. Gaunt, M. Brockschmidt, and N. Kushman, "Neural program lattices," 2017.

[31] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," in *Advances in neural information processing systems*, 2016, pp. 64–72.

[32] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[33] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," *ICLR*, 2017.

[34] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *JAIR*, vol. 61, pp. 215–289, 2018.

[35] S. Krishnan, R. Fox, I. Stoica, and K. Goldberg, "Ddco: Discovery of deep continuous options for robot learning from demonstrations," in *CoRL*, 2017, pp. 418–437.

[36] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 360–375, 2012.

[37] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto, "Learning and generalization of complex tasks from unstructured demonstrations," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5239–5246.

[38] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *ICRA*. IEEE, 2017, pp. 2169–2176.

[39] H. H. Bui, S. Venkatesh, and G. West, "Policy recognition in the abstract hidden markov model," *Journal of Artificial Intelligence Research*, vol. 17, pp. 451–499, 2002.

[40] A. Lazaric, "Transfer in reinforcement learning: a framework and a survey," in *Reinforcement Learning*. Springer, 2012, pp. 143–173.

[41] D. Borsa, T. Graepel, and J. Shawe-Taylor, "Learning shared representations in multi-task reinforcement learning," *arXiv preprint arXiv:1603.02041*, 2016.

[42] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[43] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 2371–2378.

[44] A. Lazaric and M. Ghavamzadeh, "Bayesian multi-task reinforcement learning," in *ICML-27th International Conference on Machine Learning*. Omnipress, 2010, pp. 599–606.

[45] C. Dimitrakakis and C. A. Rothkopf, "Bayesian multitask inverse reinforcement learning," in *European Workshop on Reinforcement Learning*. Springer, 2011, pp. 273–284.

[46] Z. Yang, K. E. Merrick, H. A. Abbass, and L. Jin, "Multi-task deep reinforcement learning for continuous action control." in *IJCAI*, 2017, pp. 3301–3307.

[47] P. Dewangan, S. Phaniteja, K. M. Krishna, A. Sarkar, and B. Ravindran, "Digrad: Multi-task reinforcement learning with shared actions," *arXiv preprint arXiv:1802.10463*, 2018.

[48] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," *arXiv preprint arXiv:1511.06295*, 2015.

[49] H. Yin and S. J. Pan, "Knowledge transfer for deep reinforcement learning with hierarchical experience replay." in *AAAI*, 2017, pp. 1640–1646.

[50] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.