RAPTOR Plus: Implementing a Robust Systems Architecture for Social Hierarchical Learning

Thaddeus Diamond
diamond@cs.yale.edu

Abstract—Much of contemporary robotics research focuses on methods by which robotic agents can operate in environments that have little, if any, human presence. Human-robot collaboration, where robots perform tasks alongside a human confederate, has been almost wholly ignored by current research. Social Hierarchical Learning is a method by which robots can learn representations for primitive actions, become experts at performing those actions, decompose a complex task into some superset of those skills, and then enter a mixed human-robot environment to begin actively collaborating. The purpose of this independent research project was to aid in the design and implementation of such a framework, decomposing the complex pipeline described into a set of discrete components that can be investigated independently of one another or linked together to form a robust and flexible system. The result is prototype software for several components in the framework and a detailed architecture for implementing the remaining modules.

I. INTRODUCTION

Robotics research has traditionally focused on research involving the completion of complex tasks by autonomous robots in a controlled setting. However, the “trivial nature” of generalizing this model to a real-world, mixed, human-robot environment has increasingly garnered much criticism in recent years. Corporations such as iRobot[7] and Willow Garage[16] are attempting to lead the charge by conducting computationally rigorous, yet highly application-focused research. However, these products fall far short of providing robust robotic systems that can dynamically interact with humans in practical settings. Additionally, Robonaut[4], a robotic system mimicking the upper torso of an astronaut, has been extremely successful in performing tasks alongside humans in space. However, this system—beyond being an extremely expensive, one-of-a-kind prototype developed by NASA—relies heavily on teleoperation to perform tasks that require precision and teamwork, obscuring its direct impact on automated human-robot collaboration.

The unavailability of precise sensory data for the ground truth representations of tasks across several different knowledge domains necessitates that any successful robotic system be able to learn by actively watching experts and non-experts alike perform the task. Learning from Demonstration (LfD)[2] is a common method for doing exactly this. Recent work by Jenkins et al. provides examples for one of several ways to learn robotic action control policies through LfD[6].

Once knowledge about how to roughly perform a specific task has been acquired, a robotic system must then be able to perfect that skill over time. Reinforcement learning—a method by which an agent traverses a state space state by state over time and receives rewards (or penalties) at each iteration—has been extensively used to improve an agent’s action execution[3][15]. Unfortunately, the cost of finding an expert in a given domain who could create a representation for the underlying reward function has been so prohibitive that the use of reinforcement learning has not been used as frequently for real-world applications. Recent work by Tamer and Knox[8] has improved the ease with which simple, non-expert binary feedback can be leveraged to improve a robot’s action policy execution. In addition to allowing the use of non-expert reinforcement, this work greatly reduces the number of trials required to converge an action upon an acceptable policy. However, the research has largely focused on simulated domains such as Tetris. Its applicability and efficiency in real-world domains remains to be investigated.

After a set of action primitives have been learned by a robot, it must decompose a complex collaborative task into a simple, traversable representation. Traditional research has not generally focused on this aspect of robotic learning, as few robots actually have to perform a complex task in a non-serial order (i.e. in parallel with other workers). However, there is a large body of work relating automatically decomposing a complex set of sensor data into a hierarchical task structure using learned state representations[11]. The problem of segmenting efficiently is further investigated in the face of highly dynamic environments with several tasks being performed at once[5]. As we will discuss further in section VI, our system will use a hierarchical task structure in favor of other representations.

Finally, once a set of primitives is learned at an expert level and a task has been reduced to a hierarchical sequencing of some subset of those primitives and the “unknown” task, each component of it can be manually designated as “human,” “robot,” or “ambivalent.” An investigation of the methods for creating a process by which this assignment can be automated is left for further research. As soon as this categorization has been completed, humans and robotic agents can begin collaborating on the complex task structure in parallel, with the robot adjusting to the human’s relative skill-set in real time.

It is with this learning pipeline in mind, and by leveraging the research of eminent researchers in the previously mentioned fields that we introduce Social Hierarchical Learning (SHL). Our system is capable of, given a set of primitive actions, augmenting the competency with which those skills can be performed, constructing a hierarchical task structure for a complex collaborative assignment, and responding in
real-time to the needs of humans the deployed robots are collaborating with.

Section II discusses in brief research in related fields. In section IV we discuss the common representation for action primitives throughout the entire SHL pipeline. Section V details RAPTOR, the component of our system that is responsible for simultaneously classifying the primitives that compose a complex task and improving the expertise with which the individual skills can be performed. Section VI discusses preliminary work we are doing in constructing a hierarchical task structure for a set of time-ordered primitive actions. Finally, in section VII, we detail some of the steps required for completion of the SHL pipeline, and in section VIII we conclude.

II. BACKGROUND AND RELATED WORK

In this section we introduce terminology and research that is relevant to individual components of the SHL pipeline.

A. Portable Skill Representation

How a robotic system might represent primitive actions—the ones that will compose the complex task such a system will collaborate with a human on—can vary. Traditional robotic systems often use Markov Decision Processes (MDPs) in combination with Q-Learning to accomplish this goal. The result of such an effort is both a Q-Table, representing the known states in a given environment and the corresponding rewards for transitions between them, and a policy indicating all possible executions of a skill. Such a policy, in the simplest case, can be represented as a greedy traversal of the Markov Decision Process (MDP) until the goal state is reached.

Because a Q-Table can be viewed as an arbitrary n-dimensional representation of a given state space, it can be leveraged to classify the observed execution of known policies. For example, consider a state space representing positions of an agent’s right hand, right elbow, and right shoulder in three-dimensions. A robotic system with a pre-defined dictionary of gestures could use standard statistical methods to decompose sensor data into timeframes labeled according to which of the robot’s policies most closely matched during that interval. In fact, that is exactly what our system does. Such a modality for representing action primitives can be generalized beyond our initial demo using ASL to domains such as tool manipulation and collaborative problem solving requiring an external perspective (e.g., tasks used as samples by Trafton et al.).

The notion of skill transfer, repurposing learned data to be applicable in novel contexts, is not wholly original. Konidaris et al. [9] have successfully leveraged existing learned skills to perform manipulation of simple tools. However, our major contribution is that the skill transfer resides not in transferring knowledge across domains, but rather enhancing the utility of training data through reuse and reinforcement during object recognition.

B. Q-Learning in Robotics

Markov Decision Processes provide a convenient and efficient mechanism to create flexible, arbitrarily complex options: closed-loop policies describing action sequences [13]. The temporal abstraction and generality of representation make this a favorable method of internally representing knowledge about actionable skills. When designing for a non-expert audience, it is often favorable to trade complexity (and, on occasion, optimal resource allocation) for simplicity. This accessibility contributes to Q-Learning [15] being one of the most widely utilized reinforcement learning methods within robotics. After having acquired a set of primitive actions during the first phase of SHL, it is completely natural that our robot would then use Q-Learning to augment the level of expertise with which each skill can be performed.

However, generic Q-Learning problems scale exponentially with dimensionality, therefore necessitating that we employ methods for reducing the number of trials it takes to converge on an acceptable policy. One example heuristic uses human feedback as both an immediate and anticipatory reward signal, leveraging human foresight to effectively reduce the search depth required to learn acceptable paths through state space [8]. Another effective heuristic involves leveraging data obtained through observing demonstrations of skills to favorably influence transition probabilities within the MDP, achieving accelerated convergence to an acceptable policy [2]. Our algorithm, RAPTOR, extends this capability by performing a skill classification step as well (combining phases two and three of the SHL pipeline).

C. Hierarchical Skill Architectures

A hierarchical skill architecture is useful for collaborative task execution. To maximally benefit the collaborative aspects of complex task execution, all agents involved must have reasonable approximations of the intent and plan of each other worker. Humans are naturally receptive to skill scaffolding, expressing a complicated task in terms of actions, sub-actions, and temporal sequencing (an in-order traversal of the hierarchy’s leaves). As the manner in which humans interact with this system is crucial to its long-term success, a hierarchical structure was chosen to support a robust and fluid interaction that human collaborators would be innately comfortable with.

Konidaris et. al. (2011 have successfully shown that hierarchical learning can be used to enable skill transfer between environments, identifying irrelevant dimensions of state space and decoupling them from the option representation. By extending the usefulness of a skill laboriously acquired through trial and error, the ratio of the cost of the learned action to its overall utility is reduced substantially. Automatically recognizing instances where already-known actions can be transferred and applied would constitute a significant advance for collaborative robotics. This is highlighted by the increasing incorporation of “demonstration as reward signal”, which is becoming more prevalent as the associated computational costs and challenges diminish.
III. DEFINITION OF ROBOTIC SYSTEM OPERATION

The agent and operating environment are represented by a continuous-state, continuous-time Semi-Markov decision process, defined by the tuple \((S, A, P, \Lambda, \gamma, D, R)\), where \(S\) represents a continuous set of states, \(A\) is a set of actions, \(P\) is a set of policies referred to as primitive actions, \(\Lambda\) is a set of \(\epsilon\)-neighborhoods defined for readings from each input sensor comprising the state descriptor, \(\gamma \in [0, 1)\) as a discount factor, \(D\) is the initial-state distribution, and \(R: S \times A \rightarrow \mathbb{R}\) is an unknown state-action reward function. This data structure is representative of those commonly utilized within skill training and execution systems. We further define the state vector \(\vec{v} \in S\) as \(\vec{v} \in \mathbb{R}^k\) for \(k\) sensor inputs. Our environment definition does not assume a uniform sensor refresh rate, as states may be sampled with stale data from a subset of its input sensors.

IV. PRIMITIVE SKILL ACQUISITION

Before the robot can become an expert at performing a set of low-level action primitives, it must first acquire some “rough approximation” of how to perform those skills. Then, via the RAPTOR algorithm, the system can begin improving its internal representation and categorizing a complex task.

For now, we assume the agent holds these primitives a priori. We denote the set of these skills as \(P\), each describing a policy \(P_\pi\) for traversing the agent-environment SMDP. Each primitive action contains its own exploration function, \(P_e\), to guide its traversal while solving for known branches of the problem-space MDP. Additionally, each primitive \(P \in P\) contains metadata indicating the estimated time required for completion of the primitive, \(P_t\). Completion of a primitive action is defined as traversing state space from an unknown initial state to within the \(\epsilon\)-neighborhood of a known goal state.

Each policy \(P_\pi\) can be described as a series of decision rules to be applied when appearing in familiar situations. These decisions may range from completely deterministic to fully stochastic, and are influenced by the exploration function \(P_e\). Within each primitive, we introduce the notion of reward layers, which are used to influence exploration transition probabilities but not spread during reward signal propagation.

V. RAPTOR: PRIMITIVE ACTION RECOGNITION AND REINFORCEMENT

In this section we present Reinforcing Action Primitives through Observation and Reinforcement (heretofore referred to as RAPTOR), an algorithm for simultaneously categorizing a complex task into a set of learned primitives and improving a robot’s level of expertise in performing those primitives. The key insight we make here is that once we have categorized a primitive task inside a larger demonstration, we can use that portion of the input stream as expert training data, and improve our SMDP representation. The algorithm leverages particularly strong demonstrations to propagate what is perceived as the most beneficial reward signals while processing the identification step, contributing towards converging on an optimal execution policy. RAPTOR performs opportunistic inverse reinforcement learning [12] over multiple policies through continuous observation.

A. RAPTOR Algorithm Specification

Initialize \(P_{\text{hit}} = \{\emptyset\}\) for each \(P \in P\)

While (True):

1) Poll each sensor to generate a composite state descriptor vector, \(\vec{v} \in \mathbb{R}^k\).

2) For each \(P \in P\):

   a) If the observed input state possesses non-zero action-reward values through incoming transitions originating at the immediately prior observed world state, it is added to \(P_{\text{hit}}\). Previously
unobserved states within the \( \epsilon \)-neighborhood of known states are given copies of incoming and outgoing action-reward transitions of these ‘nearby’ states, with reward values weighted inversely proportional to distance in \( \mathbb{R}^k \).

b) If the duration \( d \) represented between \( P_{\text{hit}}[0] \) and the current time is greater than a defined tolerance constant \( t \cdot P_t \); erase states off the front of \( P_{\text{hit}} \) until the \( d \leq t \cdot P_t \) seconds.

c) If either the duration \( d \) represented within \( P_{\text{hit}} \) is too short and fails to satisfy \( d \geq 1/t \cdot P_t \) or if the density of \( P_{\text{hit}} \) is less than a predefined accuracy constant \( a \), continue to the next primitive \( P \in \mathbb{P} \) at step (2).

d) For a previously specified neighborhood-distance relaxation constant \( d_g \in [1, \infty) \), if the currently observed state is within the \((d_g \cdot \epsilon)\)-neighborhood of a known expert-trained goal state:

   Compute a confidence \( C_P \in [0, 1] \) representing likelihood that \( P_{\text{hit}} \) is an example of \( P \) as dictated by:

   \[
   C_P = \alpha_1 \cdot \text{pathlen}_a + \alpha_2 \cdot \text{pathlen}_o + \alpha_3 \cdot \bar{R}_{\text{optimist}} + \alpha_4 \cdot \bar{R}_{\text{actual}}
   \]

   subject to the constraint

   \[
   \sum_{i=0}^{4} \alpha_i = 1.0
   \]

   i) Generate an ‘optimistic’ policy \( \pi_a \). This is accomplished by adding states from \( P_{\text{hit}} \) into a set \( W_p \) with probability \( p_{wp} \). For each state ("waypoint") in \( W_p \), modify all incoming action-reward transitions by adding a reward bonus \( w \) to bias towards generating policies favoring transitions through waypointed states. We utilize these policies of idealized traversal for reward propagation, encouraging the agent to traverse similar paths during action execution, influencing the agent’s behaviors through observations of its peers.

   ii) Compute a path through MDP-space according to the optimistic policy, beginning at state \( P_{\text{hit}}[0] \), terminating when within a reduced \( \epsilon \)-neighborhood of an expert-trained goal state in \( P \) or when no outgoing transitions remain. We define \( \bar{R}_{\text{optimist}} \) as the mean transition reward sustained throughout the execution of \( \pi_a \).

   iii) Repeat (ii), following a policy closely resembling the observed data, \( \pi_o \). We define \( \bar{R}_{\text{actual}} \) as the mean transition reward sustained throughout the execution of \( \pi_o \).

   iv) Compute the number of states traversed following policy \( \pi_a \) to completion:

   \[
   \text{pathlen}_a = 1 - \left( \text{sampling rate} \right)
   \]

   \[
   \left( \text{states traversed via } \pi_a - P_t \right) / P_t
   \]

   v) Compute the number of states traversed following policy \( \pi_o \) to completion:

   \[
   \text{pathlen}_o = 1 - \left( \text{states traversed via } \pi_o - P_t \right) / P_t
   \]

   e) Apply a \((C_P \text{ score, label})\) tuple to each frame of input data represented within \( P_{\text{hit}} \).

   f) Given a high-confidence threshold \( h_c \), if \( C_P \geq h_c \):

      i) Permanently apply additional reward coefficient \( \beta_{\text{reward}} \in [1, \infty) \) to boost values of all action-reward transitions utilized in the traversal through \( \pi_a \).

      ii) Set \( P_{\text{hit}} = \{ s_i|\{s_i \in P_{\text{hit}}\} \cap \{i > |P_{\text{hit}}|/2\} \} \)

B. Details and Implementation

We introduce the concept of probabilistically "waypointing" states along observed paths to generate a hybrid policy allowing for increased noise and error tolerance. Waypointed states have temporary, artificially high incoming transition rewards, biasing exploration functions to choose traversals incorporating them. We combine the use of waypointed states with greedy, non-exploratory policies to create an optimistic assessment of how a particular observed MDP traversal can be combined with what is already known about a particular skill. This step is especially important when considering the consequences of implicitly attempting to declare which skill is being demonstrated, as the learner must be able to check if the observed decision sequence is at all relatable to its internal model of each primitive’s policy. In lieu of waypointing, the confidence value is calculated by increasing \( \alpha_1 \) and \( \alpha_4 \) while decreasing \( \alpha_2 \) and \( \alpha_3 \) to zero. Additionally, algorithm steps 2(d)i and 2(d)ii are not executed, as these constitute the waypoint generation process.

One of the most troublesome concerns within any system dealing with high dimensional MDPs is maintaining sparsity of space representation. As the initialized state space increases, the process of intelligently connecting previously unseen states becomes computationally infeasible while simultaneously maintaining the responsiveness required for real-world systems. Within the context of our algorithm, it is suggested that a garbage collection mechanism be implemented that prunes low-reward states, prioritizing the removal of those with high connectivity.

C. Preliminary Results

To show the effectiveness and viability of RAPTOR, we tested its ability to classify a set of seven American Sign Language gestures: coat, drive, in, out, school, shoes, and store. Due to the inherent similarity of movement within our
gestures and low fidelity of our input signals, the chosen gestures had significant overlap in six-dimensional space. In particular, the gestures for ‘shoes’, ‘school’, and ‘in’ are extremely similar after being converted into our six-dimensional reference frame. In all cases we trained our agent using a single, expert-provided example of unaltered, Kinect-recorded data consisting of centroid coordinates \((x,y,z)\) for each hand in a frame of reference centered on the location of the demonstrator’s head. We presented our system with sequences of naturally captured, noised, and lossy data. As expected, our algorithm performed without error when tested on its training data. As increasingly lengthy gesture compositions were provided, each observed gesture became increasingly robust to both noisy data and outliers. We generate noisy data by multiplying each value within the incoming state descriptor vector by a randomly selected value \(r \in [1 - n/2, 1 + n/2]\), where \(n\) designates the severity of the noise. We simulate lossy data by replacing large contiguous blocks of valid frames, approximately 10-30% of total sensor input, with a single repeated stale input frame. This simulates the failure of the sensor being polled by RAPTOR. The algorithm’s mechanism for estimating the proper connection of previously unseen states successfully accommodated for the noise and absent states in the test data.

The complex scenario shown in figure 4 and figure 3 included the placement of some of the most similar gestures in close temporal proximity. We introduced additional noisy data, composited randomly from familiar states in each of the seven gestures’ training data, to demonstrate our algorithm’s ability to discern when there is an unrecognized action taking place. The introduction of waypointing, shown in Figure 3, boosted overall confidence values for correct classifications and smoothed the transitions between non-confident and confident classifications. One particular example of this can be seen near frame 410, as without waypointing the gesture for ‘school’ is confused with the end of the ‘shoes’ gesture preceding it.

The successful application of our algorithm to this representative example of non-trivial action classification serves as a proof-of-concept that can be applied to benefit any generic hierarchical learning framework.

VI. HIERARCHICAL PLANNING AND HUMAN-GUIDED TASK ASSIGNMENT

Once the complex task structure has been decomposed into a time-delineated sequence of events, it must be converted to a hierarchical representation. This is advantageous because it allows entire subsections of the task to be categorized as appropriate for human agents, robotic agents or both. In addition, designating separate branches of the tree as parallelizable will allow for concurrency and faster task completion.

Training data for creating a hierarchy is provided by having several humans perform the task in one of two ways. First, the human is asked to perform the task twice. Often, this will result in the human naturally performing sub-groupings in parallel, rather than performing each task to completion. This parallelization reveals several possible branches that could compose some arbitrary task tree. Secondly, the human is asked to pause for a moment in between portions of the task that he or she considers separate subtasks. In the case that all humans carry out the task to completion serially in the first method, this provides an explicit demarcation for possible branches of the tree. There are relatively trivial ways to assign a set of time-order primitives to a hierarchy. However, instead of simply dividing the work into a first half and second half or using other heuristics to provide a division of labor, our hierarchical categorization system aims to provide an automated way to discover the underlying segmentation function for any domain. Therefore, when new tasks composed of similar, yet never before seen, primitives are introduced, the segmentation function learned from training data will be able to compose an accurate hierarchy without requiring new samples.
Although not yet finished, we are currently developing a mechanism for automating the learning of this segmentation function based on training data. Two contiguous actions are placed into separate branches if the sum of some set of features, each multiplied by its individual discount factor, is greater than some predefined threshold. The initial features we have decided to use when comparing to actions include:

1) Are the “tools-in-use” common to both primitives?

2) Historically, have the two primitives been grouped together or separately?

3) What is the distance between the feature set of each skill?

4) What is the distance between the geographic centers of each action’s execution path?

5) Are there any danger factors that would necessitate the new primitive be labeled as “human-only”?

6) Apply a bonus weight based on a trend toward $n$ primitives per grouping, where $n$ is inversely correlated with the total number of agents.

For now, we manually delegate the role assignment (human/robot/both) using an Android application that we have developed in our lab. However, future research should investigate the plausibility of automating such decisions.

**VII. RELATED AND FUTURE WORK**

It is important to differentiate our contributions from existing work in the fields of LfD and automated learning of hierarchical control structures. Although leveraging the insights of those bodies of work for the overall SHL framework, our ultimate goal is to build a unified pipeline that begins with few assumptions about the environment of the system’s agents, and culminates in humans and robots working together and responding to each other in real time.

Much of the focus of this paper has been on preliminary work to convert a set of rough approximations for low-level primitives into a complete representation of a hierarchical action structure for a complex task. However, research in the future will be directed more at closing the loop and involving humans in the system. For example, little work has been done in allowing humans to provide information that will allow the branches of the hierarchical task tree to be parallelized. Creating interfaces that are intuitive and leveraging existing modalities for role assignment by human operators will be crucial.

Recent funding has provided us with the opportunity to acquire materials for the creation of a smart bench. The workstation we have designed includes three Kinect sensors, a dedicated Linux workstation with 16GB of memory, modular framing systems designed by 80/20 and two KUKA youBot arms. This setup will provide excellent test data and, ultimately, demonstrations of the power of SHL for quickly accomplishing “four-armed” tasks such as soldering and construction.

One area of research that we have not yet had the time to investigate involves how the robot should handle reactive cues during the period of time during which the agent is actively collaborating with the human. These signals will be presented in many ways including rewards, punishments, and general feedback. In addition, the manner in which they are communicated may be aurally, visually, or through a computer-based control system. Regardless of the method of communication, such feedback must be processed with extremely high priority to ensure the safety of the human and the successful completion of the task.
VIII. Conclusions

We have presented herein a framework for converting a set of poor representations of primitive actions into a thorough understanding of a complex task structure. The complex task can then be executed by humans and robots collaborating in a dynamic environment. By utilizing human involvement and feedback throughout the course of the collaborative process, robots can operate alongside humans on complex tasks such as cooking and construction, where a deep understanding of the underlying task structure and awareness of the current progress are essential.

For several years, research in robotics has focused on answering questions of engineering, design, and planning in order to operate in real-world environments. However, much of this research has, unfortunately, been based off of assumptions that certain properties of the real world can be largely ignored, or that no human agents are a part of the robot’s environment and therefore the largely stochastic behavior associated with those human agents can be ignored. Rodney Brooks, a pioneer in Human-Robot Interaction and the co-founder of both iRobot and Heartland Robotics, attempted to address this issue of usability in his recent keynote at HRI 2012[1]. As such, the success of the SHL framework resides in its fundamental assumption that the only environment a robot should operate in is one that contains both humans and robots and where the robots must respond directly to the needs and relative skills of the human agents, not vice versa. As Brooks opined, “User experience is critical to robot success. It’s about the user, not the robot.”[10]

IX. Acknowledgments

Large sections of this paper have been adapted from a working draft of a conference submission, grants, and presentations on which Bradley Hayes, Professor Brian Scassellati and I have co-authored. Thanks are due to Brad for his careful thought and effort on leading the current research and development of the SHL prototype system and to Sam Spaulding and Caroline Jaffe for their help as well. This work is being funded in part by an ONR grant, reference number C148807.

REFERENCES