Perturbation Training for Human-Robot Teams

by

Ramya Ramakrishnan

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

© Massachusetts Institute of Technology 2015. All rights reserved.
Perturbation Training for Human-Robot Teams

by

Ramya Ramakrishnan

Submitted to the Department of Electrical Engineering and Computer Science on May 20, 2015, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Abstract

Today, robots are often deployed to work separately from people. Combining the strengths of humans and robots, however, can potentially lead to a stronger joint team. To have fluid human-robot collaboration, these teams must train to achieve high team performance and flexibility on new tasks. This requires a computational model that supports the human in learning and adapting to new situations.

In this work, we design and evaluate a computational learning model that enables a human-robot team to co-develop joint strategies for performing novel tasks requiring coordination. The joint strategies are learned through “perturbation training,” a human team-training strategy that requires practicing variations of a given task to help the team generalize to new variants of that task. Our Adaptive Perturbation Training (AdaPT) algorithm is a hybrid of transfer learning and reinforcement learning techniques and extends the Policy Reuse in Q-Learning (PRQL) algorithm to learn more quickly in new task variants. We empirically validate this advantage of AdaPT over PRQL through computational simulations.

We then augment our algorithm AdaPT with a co-learning framework and a computational bi-directional communication protocol so that the robot can work with a person in live interactions. These three features constitute our human-robot perturbation training model. We conducted human subject experiments to show proof-of-concept that our model enables a robot to draw from its library of prior experiences in a way that leads to high team performance. We compare our algorithm with a standard reinforcement learning algorithm Q-learning and find that AdaPT-trained teams achieved significantly higher reward on novel test tasks than Q-learning teams. This indicates that the robot’s algorithm, rather than just the human’s experience of perturbations, is key to achieving high team performance. We also show that our algorithm does not sacrifice performance on the base task after training on perturbations. Finally, we demonstrate that human-robot training in a simulation environment using AdaPT produced effective team performance with an embodied robot partner.

Thesis Supervisor: Julie A. Shah
Title: Associate Professor
Acknowledgments

First, I would like to thank my advisor, Professor Julie Shah, who has been the most supportive and caring supervisor I could have ever asked for. She has been so involved in every step of the process and works so hard to meet with all of us for the incredibly many deadlines we have. Her positive attitude and smile at every meeting encourages me to continue working hard while staying happy and optimistic. She serves as an amazing role model for me, motivating me and inspiring me to conduct impactful research. I thank her for always supporting me and for giving me the opportunities to become a better researcher.

I want to thank my labmates who I enjoy seeing every morning when I come into work. I’d like to thank Vaibhav for all the enjoyable discussions we have had and for being so helpful and supportive. I want to thank Abhi for being an amazing social co-chair with me and for all the story-telling and interesting discussions. I’d like to thank Been for all the wonderful dinner and shopping bonding outings and for being an amazing role model. Thanks also to Stefanos for being such an appreciative and helpful mentor and to Chongjie for his help and insightful advice. I want to thank Pem, Matthew, Joe, Ankit, Claudia, Heitor, Jorge, Alex, Keren, Kyle, and Ryan for being wonderful labmates and for making every day an enjoyable one.

I would like to thank Mandy, Danielle, Amy, Jennifer, and Mengfei for being an amazing support group and for organizing wonderful girls nights and outings. I thank Mandy also for being a great support in my classes and in difficult times of my research. I’d like to thank Karthick for his crazy stories and for encouraging me to always have fun. Both my first-year roommate Jess and my current roommate Yu-Chien have been a great support for me at home. I would like to thank my dance and singing friends, including Natya, Swara, and the Bollywollies. I am grateful to all my friends at MIT. It has truly been a pleasure getting to know everyone the past two years.

I would like to thank Karthik, who has been there for me every step of the way in the last two years, and without whom I could not have gotten through my Master’s.
I thank him for always listening to me and for being my rock of support during every stressful time at MIT. Last but not least, I want to thank my mother, father, and brother Jagdish who have loved, supported, and encouraged me throughout my life. They have taught me to appreciate everything in life and to work hard to achieve my goals. I wouldn’t be the person I am today without them.
# Contents

1 Introduction ........................................ 15  
  1.1 Motivation ..................................... 15  
  1.2 Human-Robot Perturbation Training .......... 16  

2 Related Work ....................................... 19  
  2.1 Learning Algorithm ............................ 19  
    2.1.1 Learning with Humans .................... 19  
    2.1.2 Reinforcement and Transfer Learning .... 21  
  2.2 Multi-Agent Communication .................... 24  

3 Adaptive Perturbation Training (AdaPT) Algorithm 27  
  3.1 Modifications to Prior Work .................. 27  
  3.2 AdaPT Algorithm ................................ 29  
  3.3 Computational Results ......................... 32  
    3.3.1 Fire Extinguishing Task .................. 32  
    3.3.2 GridWorld Task ........................... 36  

4 Human-Robot Perturbation Training Model .......... 41  
  4.1 Co-Learning Framework ........................ 41  
  4.2 Human-Robot Communication Protocol .......... 42  

5 Human Subject Experiments ......................... 47  
  5.1 Aim of Experiments ............................ 47  
  5.2 Hypotheses .................................... 48
List of Figures

3-1 This chart shows the pictorial difference between AdaPT and PRQL. While PRQL learns a new value function that is guided by prior knowledge, AdaPT directly adapts the previous value functions to learn well in new tasks. ................................................. 29

3-2 The Adaptive Perturbation Training (AdaPT) algorithm takes as input a new task, a library of value functions from previously learned tasks, and other parameters. It adapts all value functions simultaneously and finally returns the function with the highest weight to execute in the new task. .......................................................... 30

3-3 Update-QValues is a subfunction of AdaPT that executes one episode of the task (from initial to goal state) and updates all Q-value functions to adapt to the new task. ............................................. 31

3-4 This chart compares the performance of AdaPT and PRQL on the fire task as the number of iterations increase. After every 1000 iterations, the task is executed and the average accumulated reward received is shown over time. .................................................. 35

3-5 This chart compares the performance of AdaPT, PRQL, and Q-learning on the fire task when PRQL is seeded with different priors. The x-axis displays the wind and dryness values of the test tasks (e.g. (7,3) represents a task with wind=7 and dryness=3). The y-axis shows average accumulated reward received on an execution of the task after learning on the test task for 1500 iterations. Less negative reward denotes better performance on the task. ................................. 35
3-6 This figure shows a pictoral representation of the 10x10 gridworld task domain. The yellow tokens and black pits are placed in fixed locations on the grid. The four green locations were the four training task goal locations. The team was then tested on 100 random test goal locations.

3-7 This chart compares the performance of AdaPT and PRQL on the gridworld task as the number of iterations increase. After every 1000 iterations, the task is executed and the average accumulated reward received is shown over time.

4-1 The Human-Robot-Communicate algorithm provides a computational framework for the robot to make decisions when communicating with a person.

5-1 This figure shows the structure of the robot’s learning for the conditions used in human subject experiments. The perturbation AdaPT condition stores a library and uses the AdaPT algorithm. The perturbation Q-learning condition updates one value function for all perturbed tasks. In the procedural Q-learning condition, one value function is updated based on repeated experience with a task.

5-2 This picture shows the setup for the embodied robot experiments. Here, a participant is working with a PR2 robot to make strategic decisions on how best to extinguish a set of fires.

5-3 This chart compares the performance of the three different conditions used in human subject experiments with simulated robots: perturbation AdaPT, perturbation Q-learning, and procedural Q-learning. The y-axis shows average accumulated reward received on the execution of the task with the human subject after the robot simulated on the test task for 1500 iterations. The x-axis displays the wind and dryness values of the test tasks (e.g. (2,9) represents a task with wind=2 and dryness=9). Less negative reward denotes better performance on the task.
This chart compares the performance of teams who tested with a simulated robot vs. an embodied robot. The y-axis shows average accumulated reward received on the execution of the task with the human subject after the robot simulated on the test task for 1500 iterations. The x-axis displays the wind and dryness values of the test tasks (e.g. (2,9) represents a task with wind=2 and dryness=9). Less negative reward denotes better performance on the task.
List of Tables

3.1 This table compares AdaPT, PRQL, and Q-learning on the collaborative grid world task given limited simulation time. We include the standard PRQL approach with an uninformative prior and also PRQL with the best and worst priors from the training tasks. We report the mean and standard error for each, averaged over 100 randomly selected test tasks.
Chapter 1

Introduction

1.1 Motivation

Currently, most robots work in isolation, often performing repetitive tasks. By deploying robots separately from people, we are not able to combine the strengths of humans and robots to form a stronger joint team. Integrating robots into collaborative tasks with people can potentially lead to more efficient performance on tasks.

To achieve high team performance, teams often require rigorous training, particularly in areas such as disaster response, military, and manufacturing. As tasks become increasingly complex and require cooperation within a team of individuals, training helps a team learn to coordinate their roles effectively and perform well under novel task variants [19]. Today, some team-based tasks are performed by humans only, and others by teams consisting solely of robots. In this work, we envision that new types of human-robot training procedures would enable robots and people to work collaboratively and to accomplish tasks more efficiently together than they do separately.

Three training approaches have been widely studied in human teams: procedural, cross and perturbation training [19]. The most simple and common approach is procedural training, in which a team practices a procedure repeatedly in order to become proficient at performing a given task. This approach results in high performance on the trained task, but does not generalize to tasks involving unexpected variations or
disturbances, which almost always occur in real-life scenarios. The cross training approach aims to improve team adaptivity through practice by requiring team members to switch roles with one another. Through this process, teammates can build a shared mental model of the task and collaboration strategy, and can thus better anticipate each other's actions. This training method has been shown to benefit human-robot teams, but cannot be scaled as teams increase in size and diversity [32]. If the roles required for a task are vastly different from one another, or if the size of a team becomes particularly large, learning the roles of other team members becomes difficult and impractical [19]. Individual performance can also decline in such cases, as members must train for a greater number of roles.

The third approach, perturbation training, addresses the limitations of both procedural and cross training [19]. In this method, the team experiences slight disturbances during the training process that are intended to help them learn to coordinate effectively under new variants of a given task. A model that supports robot learning through perturbation training must allow the robot to draw from a library of previous experiences in a way that is compatible with the process of a human partner learning through perturbations. This poses a challenge, as evidenced by prior studies of human teamwork that demonstrated that human teams with members possessing accurate but dissimilar mental models performed worse at a given task than teams with less-accurate but similar mental models among members [28]. Our aim in this work is to develop a model that enables a human-robot team to co-learn a joint strategy through training on perturbations, and to demonstrate proof-of-concept that our approach supports a human and a robot in synthesizing prior knowledge in a way that results in effective team performance.

1.2 Human-Robot Perturbation Training

In this work, we design and evaluate the Adaptive Perturbation Training (AdaPT) algorithm for learning a joint strategy through practice on several variations of a task. The algorithm is a hybrid of transfer and reinforcement learning techniques
and modifies the Policy Reuse in Q-Learning algorithm (PRQL) [17] to learn more quickly in new task variants. A robot using the AdaPT algorithm learns by modifying knowledge from previously learned tasks to perform well on new task variants. We assess task similarity based on simulation of accumulated reward, a metric easily observable and understandable to humans. This measure of similarity enables the robot to effectively match a new task with the most relevant previously learned strategy.

We ran simulations over two domains comparing our algorithm to PRQL over time to evaluate how quickly the two algorithms learned a new task. On both domains, we found that AdaPT learned much more quickly, but PRQL was able to learn well given enough time. We then compared the performance of the algorithms under limited simulation time, because in real-world tasks the robot will have to make decisions quickly when working with people. We compared AdaPT, PRQL with different priors, and Q-learning from scratch. Because standard PRQL starts with an uninformative prior, we included comparisons to PRQL initialized with the best and worst prior from training. We found that given limited simulation time, AdaPT achieved significantly higher performance on new task variants than PRQL with an uninformative prior and Q-learning from scratch. While PRQL performed well with an intelligent prior, the technique does not provide a mechanism for choosing this prior. We found that a key strength of AdaPT is that it provides an automated framework for selecting and utilizing an appropriate prior to quicken learning.

We then extended our algorithm so that the robot can work with people in live interactions by including a co-learning framework and a two-way communication protocol. These three features form our human-robot perturbation training model. The co-learning framework allows the robot to learn with the person in a collaborative way by simulating in between each human interaction. The robot refines its strategies while learning with the person similar to how people work collaboratively with one another. The communication protocol provides a mechanism for the robot to interact with the person and coordinate decision making. Through this computational framework, the robot can suggest actions to the person and determine whether or not to accept the person’s suggestions.
To evaluate whether our model allows the robot to work effectively with people, we conducted and report on human subject experiments wherein 36 teams of two (one person and one robot) co-trained under perturbations in a simulated environment. We analyzed the performance of two models for perturbation training and a model for procedural training. We included an alternative perturbation training algorithm, standard Q-learning with no library, to determine whether a human’s experience of perturbations is enough to achieve high team performance or if it is important for the robot to have the right model to support flexibility in new situations. We found in our experiments that there was a significant difference between the two algorithms, which suggests that the robot’s algorithm is important in achieving high team performance. We also included a comparison of perturbation training using AdaPT to procedural training, in which team members practice repeatedly on one task. We found that perturbation AdaPT teams did not sacrifice performance on the base task, on average, when compared to procedurally trained teams.

Finally, we conducted embodied robot experiments to determine the effect of introducing a real robot into the task. All participants trained in simulation but half tested with an embodied robot, the PR2. We found that human-robot training in a simulation environment using AdaPT resulted in effective team performance with the PR2. This is a promising result, as this means people may be able to train and learn strategies in the task with low-cost simulated robots and then work effectively with embodied robots.

In the next chapter, we will discuss prior work in human-robot collaboration and related techniques. In Chapter 3, we will introduce our Adaptive Perturbation Training (AdaPT) algorithm and present computational results across two domains. Chapter 4 presents the human-robot perturbation training model, which in addition to the algorithm, includes a co-learning framework and a communication protocol. Chapter 5 discusses human-subject experiments, including the hypotheses, the setup for both the simulated and embodied robot experiments, and the results. Chapter 6 will finally end with conclusions and future work.
Chapter 2

Related Work

We first present an overview of previous literature on technical approaches that are related to our computational model for perturbation training. Our model includes a human-robot learning algorithm, a co-learning framework, and a communication protocol. We review prior work in these areas here.

2.1 Learning Algorithm

2.1.1 Learning with Humans

Substantial works have focused on the training of robots to perform complex tasks when given sets of human demonstrations. In [2], humans physically guide the robot through a sparse set of consecutive keyframes, which are connected to perform a skill. However, this requires several demonstrations, which can be difficult for a human to provide. Alexandrova et al. aim to reduce the number of demonstrations by allowing humans to demonstrate the task once and then having them provide more information through an interactive visualization tool [3]. To take better advantage of each human teaching instance, [12] develops a framework to teach multiple robots at the same time. Further leveraging demonstration data from humans, [31] automatically detects repeated structure in the data by segmenting motions and constructing a finite state automaton that represents the task. This technique provides high-level task
knowledge and a set of reusable, flexible skills, such as assembling one leg of a table. These approaches support natural human interaction and help robots learn effectively based on a few human demonstrations. They further enable robots to accommodate variations in the motion- or action-level execution of complex tasks.

While Learning from Demonstration (LfD) supports the robot in learning, it is limited in that it requires demonstrations of the task. Other works learn from human input by incorporating feedback or reinforcement to improve robot decision-making. The TAMER framework, developed by [24], models a human’s reinforcement function, which the robot then uses to select actions that will likely result in the most positive human reinforcement. In [20], rather than using human feedback to shape rewards or Q-value functions, human input is directly used as policy labels to infer what the human thinks is the optimal policy. Another work [11] develops a system for turn-taking interactions between a human and a robot that is used to collaboratively solve the Towers of Hanoi problem. An interactive robot task training framework was developed by [38], where the robot would observe a human performing a task, interrupt when more clarification was needed, and incorporate feedback to improve execution of the task.

Relatively few works address interdependent learning between a human and a robot. One work [22] develops a representation of structured tasks that the robot uses to infer current human actions as well as the execution and timing of future actions. They show that a robot is able to robustly infer human actions and work effectively on a collaborative assembly task. A similar work to ours, [33], incorporates “cheap talk,” non-binding, costless communication, to better interact with a human partner. Examples of “cheap talk” include “I’ve changed my mind” and “That was not fair!” This work, however, considers competitive games rather than purely collaborative tasks.

More closely related to our work in human-robot collaborative team training, [32] computationally models a team-training technique from human team studies called cross-training and uses a similar Markov decision process (MDP) task representation. In this approach, a human and a robot switch roles during training to learn a shared
mental model of the task, which helps the team adapt to each other’s preferences. While this technique allows humans and robots to learn from each other, it considers a simple task with 27 states and is limited in its ability to scale to tasks that require complex strategies for coordination.

2.1.2 Reinforcement and Transfer Learning

Many techniques use Markov decision processes (MDPs) to represent stochastic tasks in which the agent learns how to act robustly through reinforcement learning (RL) [9, 35, 40]. In the RL framework, an agent interacts with an environment, observing state $s$ at each time step and deciding to take an action $a$. The environment returns a reward $r$ and a next state $s'$. The agent continues taking actions until it reaches some specified goal state. The agent executes these episodes, each of which goes from an initial state to a goal state, repeatedly to learn how to perform the task well. Q-learning [47] is an RL algorithm that learns a Q-value function $Q(s, a)$ that specifies the expected utility of taking action $a$ from state $s$. The agent’s goal is to learn a deterministic Markov policy $\pi(s)$, which maps each state $s$ to an action the agent should take. It can be computed using $Q(s, a)$ by taking the action $a$ with the highest value at each state $s$. A policy is called optimal if it obtains the maximum expected discounted future reward from any initial state [40].

More formally, we describe the representation of an MDP as a tuple $< S, A, T, R >$. $S$ is the discrete set of world states, and $A$ is the discrete set of all possible actions. $T(s, a, s')$ is the state transition function that specifies the probability of going to next state $s'$ given the current state $s$ and action $a$, and $R(s, a, s')$ is the reward function that specifies the reward gained from taking action $a$ in state $s$ and transitioning to next state $s'$. The policy $\pi(s)$ learned from training specifies the action the agent should take at each state $s$. While RL is one approach to learn a policy in an MDP, it can be slow if it is used to learn each task from scratch separately. Even with human input and guidance, it can be inefficient in the computation of policies for new task variants.

To speed up learning, transfer learning in MDPs [34, 44, 46] is applied to learn
a new task given similar, previously learned tasks. Prior works primarily consider
the transfer of knowledge from one source task to one target task. In [25], low-level,
primitive actions are generalized to create temporally extended high-level skills, called
options. For example, the option openDoor may consist of policies that tell the robot
how to first reach, then grasp, and finally turn the door handle. To port the options
over to several problems, they use two different representations: a representation in
problem-space, which is Markovian and specific to a particular task, and a representa-
tion in agent-space, which can be non-Markovian and shared among many problems.
Options are learned in agent-space rather than problem-space, allowing the options to
be reusable in other tasks with the same agent-space. For example, in the lightworld
domain, an agent has light sensors and uses them to find a key and unlock the door
to complete the task. The light sensor readings are in agent-space because they are
transferrable across lightworld instances, however the problem-space contains more
specific information like the room numbers, x and y coordinates of the agent, and
whether or not the agent has the key. Transfer learning is also applied to model-
based approaches [42] to improve performance and sample efficiency. These works,
however, only transfer knowledge from one source task to one target task and cannot
be directly used when there are multiple source tasks.

For perturbation training, we want to incorporate many source tasks, each a slight
variation of the other, to assist in learning a new but related task. This requires
determining which previous task is most relevant to the current one, and then using
both that knowledge and prior experience to quicken the learning process for the
new task. The Policy Reuse in Q-learning (PRQL) algorithm [17, 45] is one transfer
learning approach that closely aligns with the goals of perturbation training. This
algorithm learns a policy for a new task through weighted blending of past policies
during the reinforcement learning process. The approach is parsimonious, in that it
considers the similarity of a new policy to existing policies and adds it to the library of
policies only if it is sufficiently different from those already present. This algorithm,
however, only addresses single-agent tasks.

In addition to policy reuse, many other mechanisms exist for selecting the most
relevant source task(s) for learning a target task. One work [43] creates a hierarchy of
tasks ordered by solution difficulty. This relative ordering is used to select source tasks
that can be learned more quickly than the given target task, thereby reducing total
training time. Another approach [41] develops an expert algorithm to choose which
policy to use in a new task. The agent has a set of candidate policies, or “experts,”
from a previously learned MDP, and the goal is to use these experts intelligently to
obtain performance in the new task close to the best expert’s performance. In [14],
many classifiers are used, each focusing on a different resolution of the data. The
classifiers in low resolution can be transferred between tasks, while the ones in high
resolution are more specific to the particular task. For example, in low resolution,
the task of recognizing the letter “C” can be transferred to recognize “G”, but at a
higher resolution, these letters have unique characteristics that differentiate them.

Some works use explicit task similarity measures to guide the blending of multiple
relevant source tasks for use in a target task. One work [10] explores multiple measures
of similarity between MDPs, including transfer time $d_T$, policy overlap $d_P$, Q-values
$d_Q$, and reward structure $d_R$. Given a set of tasks, they choose one task as the target
task, and use the others as source tasks to speed up learning in that target task.
The measure $d_T$ between two tasks represents the time taken to “converge” when
knowledge is transferred from one task to the other. In this work, $d_T$ is computed
between the chosen target task and every other task, which requires actually running
multiple transfer experiments. They compare the other three similarity measures
to $d_T$ to determine how well each of these measures approximates the true gain in
transfer time. They found that there was no one “best” similarity metric across the
domains tested.

Another graph-based method is used in [15], where each task is mapped onto a
graph, and a function learned on this graph is used to transfer parameters between
tasks. Similar to $d_T$ in [10], which is a similarity metric of the time taken to transfer
to a new task, transferability in this graph is defined as the direct change in perform-
ance between learning with and without transfer. However, this method considers
classification tasks, solved using biased logistic regression, rather than reinforcement
learning tasks as in [10]. A very recent work by [5] computed an automated measure of similarity between MDPs by sampling \(<s, a, s'>\), thereby comparing the similarities in transition functions. However, these methods are limited in their ability to transfer the meaningful characteristics of similar execution strategies between MDPs with different transition and reward functions. We also observed an additional challenge in our own experiments, which is that MDPs with seemingly similar features (very small changes in $S, A, T,$ or $R$) may produce very different execution strategies.

While these transfer learning approaches are useful for transferring knowledge, not all provide a framework for multiple agents. Works involving many agents (e.g. [7]) primarily focus on the transfer from one source to a single target task. One work [27] builds on the policy reuse concept for multiple agents to learn a policy for a new task based on policies from $n$ previous types, and improves transfer learning efficiency through consideration of a restricted form of non-stationarity in the transition function.

In our problem, we assume that each collaborative task is performed by one human and one robot, and we have multiple source tasks for which the relevance to the target task must be distinguished automatically. Since humans do not satisfy the form of non-stationarity in [27], we chose to augment the original PRQL algorithm [17]. However, PRQL does not address multi-agent tasks. Having multiple agents introduces many challenges, one of which is communication, a key element in collaborative teams [26].

### 2.2 Multi-Agent Communication

In multi-agent problems, it is important for agents to communicate information about their local state and/or intentions so that all agents can cooperate effectively. Some works [53, 16] model their problem using Petri-Nets, a mathematical model used in distributed systems. In collaborative multi-agent tasks, beliefs and states of agents can be represented using a Petri-Net, and agents can decide, using this model, when to proactively 'tell' and 'ask' information [53]. A Petri-Net can also model multi-
party conversations, which are then used, through conversation pattern recognition, to anticipate information needs of other teammates [16].

While these works explore multi-agent communication, they are not directly transferable for problems represented as Markov Decision Processes (MDPs). In our problem, we have two agents, a human and a robot that must communicate using knowledge encoded in the Q-value function. We gain inspiration from several works [37, 52, 21] that select communication actions based on what will maximize the expected joint reward. Also, similar to [8] where opponent modeling is used for better action prediction, we consider human actions in our model when making decisions. We store a joint Q-value function that can be used to determine the optimal joint action from every state. In the communication process, we use the Q-values and human input to decide whether at each time step to suggest an action or whether to accept/reject a human’s suggestion. Suggestions with accepts and rejects are also used in the communication protocol described in [29]. As a preliminary step, we allow the human and robot to have full observation of the environment and to communicate prior to each joint action.

As a next step, we will include partially observable tasks and costs for communicating information to other agents. Many previous works have looked into approaches for limited communication where each agent only has local, rather than global, information of the environment. Some of these works [18, 49, 50] use decentralized partially observable MDPs (DEC-POMDPs), where each agent takes an action and receives a local observation and a joint immediate reward. The agent must decide when to communicate as there is an associated cost with every communication. One of the approaches [49] uses an agent’s perception of belief divergence in the team to determine whether to communicate; if there is low divergence, the team is coordinated and communication may not be necessary whereas high divergence implies the need for coordination. Works using variations of POMDPs or multi-agent MDPs (MMDPs) [39, 51, 36, 54, 56] also address limited communication with different sets of assumptions or approaches.

Other works have studied not just when agents should communicate, but also
what they should communicate. In [52], an agent can either ‘tell’ its local state to the other agent, ‘query’ for the other agent’s local state, or ‘sync’ to exchange both local states. Another work [37] more specifically addresses what agents should communicate by determining which observations, when combined with the current joint belief, result in the highest expected reward. Determining who to communicate with is also important, as shown in [55]. In this work, a coordination set is computed to determine which subset of agents to send information to since it may be infeasible to communicate to all $N$ agents for large $N$. In a recent work [4], the standard MDP is augmented with an explicit representation of teammates’ plans that are updated as new information is shared between members. Reasoning over and predicting intentions and plans of other agents can also improve communication and coordination in multi-agent teams.

Inspired by these many works on multi-agent communication, we hope to extend our framework in future work for more general cases where the robot must decide when, what, and who to communicate with, act under local observations, and reason over the intentions of other agents.
Chapter 3

Adaptive Perturbation Training (AdaPT) Algorithm

3.1 Modifications to Prior Work

The closest transfer learning approach to ours is the Policy Reuse in Q-learning (PRQL) algorithm, which uses a library of previously learned policies to intelligently guide the search for a new policy. In PRQL, the agent explores actions that have resulted in high reward in previous tasks. Each policy has an associated weight that is updated during learning to reflect how well that policy performs on the new task. The higher the weight of the policy, the more likely it will be chosen for the current episode. Also, as the agent learns, the confidence in these weights increases, as represented by the steadily increasing temperature parameter. Initially, the past policies heavily guide exploration, but after many iterations, the agent relies solely on the newly learned policy for execution in the new task.

The Adaptive Perturbation Training (AdaPT) algorithm augments PRQL to learn more quickly in new task variants. PRQL provides an initial framework for transferring knowledge from a library of policies to a new variant of a task. The benefit of PRQL, compared to other transfer learning approaches, is that it automatically learns what previous knowledge is relevant for the new task through simulated trials. Simulation of accumulated reward is a task similarity measure that is more easily
observable by humans than features such as transition matrices. This may help the robot generalize in a way that is more compatible with how people learn. Further, small changes in transition or reward may lead to drastic changes in strategies, but using accumulated reward as the measure allows the robot to choose the value function that receives the highest reward over time, and is thus the most useful, in the new task.

While PRQL can more intelligently guide exploration, it does not take full advantage of prior knowledge to learn well in new tasks. It begins with a value function initialized with all zeros and uses previously learned policies to guide the learning of this new function. In our algorithm, we instead directly adapt previously learned knowledge to learn more quickly in a new task. To do this, AdaPT uses Q-value functions rather than policies from previously learned tasks to more easily adapt knowledge. With every simulated execution of the new task, we update the values in all value functions to reflect what the agent learned. This allows us to simultaneously adapt all value functions to be more helpful for the new task. Finally, AdaPT returns the value function with the highest weight as the new policy. This is distinguished from PRQL which always returns the newly learned policy. In both AdaPT and PRQL, if this new Q-value function is sufficiently different from the existing ones in the library, it can be inserted as a new function, allowing the agent to continuously build knowledge.

Figure 3-1 pictorially shows the difference between AdaPT and PRQL. The Q-value functions are represented as points in this space where two value functions closer in this space are more similar. $Q_1, Q_2, Q_3$ are three value functions learned from three training tasks. $Q_{new}^*$ represents the unknown optimal value function the agent is aiming to learn for the new task. PRQL starts with an initial value function $Q_{init}^n$ and uses the three other value functions to guide this new value function closer to $Q_{new}^*$. The three functions $Q_1, Q_2, Q_3$ however are unchanged. In AdaPT, we instead adapt all value functions from training to make them closer to $Q_{new}^*$. Finally, the one closest to $Q_{new}^*$, as determined by weights in the algorithm, is chosen as the value function to be used in the new task. This new value function can then be added as a
Figure 3-1: This chart shows the pictorial difference between AdaPT and PRQL. While PRQL learns a new value function that is guided by prior knowledge, AdaPT directly adapts the previous value functions to learn well in new tasks.

new function in the library and can be used for future tasks. Our approach of directly adapting prior knowledge to learn in new situations is inspired by how humans adapt, which is by matching new tasks to previous similar cases [23].

3.2 AdaPT Algorithm

The inputs to the AdaPT algorithm (Figure 3-2) include the new task to perform, \( \Omega \); a copy of the Q-value functions that the robot learned from practicing on \( n \) training tasks \( \{Q_1, ..., Q_N\} \); the initial temperature \( \tau \) used for policy selection; the temperature decay, \( \Delta \tau \); the maximum number of episodes, \( K \); the maximum number of steps in each episode, \( H \); and standard \( \gamma, \alpha, \) and \( \epsilon \) parameters used for \( \epsilon \)-greedy Q-learning. Given these inputs, the robot simulates in order to learn a joint Q-value function for the new task \( \Omega \).

The algorithm begins by initializing the weights of all Q-value functions from the library to 0 in line 1. Line 2 initializes the counter for each value function, which tracks the number of episodes in which a value function is selected for execution. Then, line 3 simulates for \( K \) episodes, each time choosing a Q-value function \( Q_k \) to execute based on the current temperature \( \tau \) and the weights \( W_1, ..., W_N \) (line 4). The chosen \( Q_k \) is used for action selection with an \( \epsilon \)-greedy approach (with probability \( \epsilon \), a random
Algorithm: AdaPT (Ω, \{Q_1, ..., Q_N\}, τ, Δτ, K, H, γ, α, ε)

1. Set the weights of all Q-value functions: \( W_i = 0, \forall i = 1, ..., N \)
2. Set the number of episodes Q-values \( Q_i \) has been chosen \( U_i = 0, \forall i = 1, ..., N \).
3. for \( k = 1 \) to \( K \).
4. Choose a Q-value function, \( Q_k \), for the current episode, where the probability of selecting a value function as \( Q_k \) is determined by the following equation:
   \[
P(Q_j) = \frac{e^{W_j}}{\sum_{p=1}^{N} e^{W_p}} \quad \forall j = 1, ..., N
   \]
5. \(< R, Q_1, ..., Q_N > = \text{Update-QValues}(Q_1, ..., Q_N, Q_k, H, γ, α, ε)\)
6. Set \( W_k = \frac{W_k U_k + R}{U_k + 1} \)
7. Set \( U_k = U_k + 1 \)
8. Set \( τ = τ + Δτ \)
9. end for
10. Return Q-value function with maximum weight

Figure 3-2: The Adaptive Perturbation Training (AdaPT) algorithm takes as input a new task, a library of values functions from previously learned tasks, and other parameters. It adapts all value functions simultaneously and finally returns the function with the highest weight to execute in the new task.

The Update-QValues algorithm (Figure 3-3) takes as input the Q-value functions \( Q_1, ..., Q_N \); the Q-value function \( Q_k \) chosen for action selection in the current episode; the maximum number of steps per episode, \( H \); and standard \( γ, α, \) and \( ε \) parameters used in \( ε \)-greedy Q-learning. The algorithm returns the reward \( R \) obtained in that
Algorithm: Update-QValues ($Q_1, \ldots, Q_N, Q_k, H, \gamma, \alpha, \epsilon$)

1. Set the initial state $s$ randomly
2. for $h = 1$ to $H$
3. With probability $\epsilon$,
4. $<a_h, a_r> = \text{Random joint action}$
5. With probability $1 - \epsilon$,
6. $<a_h, a_r> = \max_{a'_h, a'_r} Q_k(s, a'_h, a'_r)$
7. Receive next state $s'$ and reward $r_h$
8. Update all Q-value functions $Q_i$, $\forall i = 1, \ldots, N$:
   $Q_i(s, a_h, a_r) = (1 - \alpha)Q_i(s, a_h, a_r) + \alpha[r + \gamma\max_{a'_h, a'_r} Q_i(s, a'_h, a'_r)]$
9. Set $s \leftarrow s'$
10. end for
11. $R = \sum_{h=0}^{H} \gamma^h r_h$
12. Return $<R, Q_1, \ldots, Q_N>$

Figure 3-3: Update-QValues is a subfunction of AdaPT that executes one episode of the task (from initial to goal state) and updates all Q-value functions to adapt to the new task.

The episode and updated Q-value functions $Q_1, \ldots, Q_N$. The episode is initialized with a randomly selected state $s$ (line 1). The algorithm then executes a random joint action with probability $\epsilon$ in line 4. With probability $1 - \epsilon$, the joint action with maximum value in the current Q-value function $Q_k$ is chosen (line 6). The reward is received in line 7, and then all Q-value functions are updated in line 8 using the Q-learning value update. In line 9, the state is updated, and in line 11, the accumulated reward is calculated over this episode, discounted using $\gamma$. Finally, the reward and the updated Q-value functions are returned (line 12).

Note that AdaPT and Update-QValues explicitly represent both the human and the robot in the state and action space. Learning $Q(s, a_h, a_r)$ allows the robot to learn and make decisions over this joint space. Although this increases the complexity of the problem, it allows the robot to consider human actions in its decision-making and choose optimal joint actions for the team. $Q(s, a_h, a_r)$ is sufficient for offline learning in
simulation, where the robot chooses actions for both itself and the simulated human. However, working with a real human poses an additional challenge, in that the robot does not make decisions on behalf of the human. Therefore, it is also useful to store and update a local value function \( Q(s, a_r) \) during learning, as this enables the robot to consider the best action over all possible next human actions when the next step of the human is uncertain. (Note that it is also possible to store \( P(a_h|s, a_r) \) and use this to estimate \( Q(s, a_r) \).)

### 3.3 Computational Results

We compare the performance of our algorithm to PRQL over two domains and present the results here.

#### 3.3.1 Fire Extinguishing Task

The first task involved a team of two, one human and one robot, extinguishing five fires associated with five neighboring houses on a street block. The goal of the task was to put out the fires as quickly as possible and minimize damage to the houses. In variants of the task, each of the fires was initialized with varying levels of intensity, from level 0 (no fire) to level 3 (large fire). During the task execution, fires increased by one intensity level with some probability up to a maximum level of 4, which indicated that the house had burned to an unrecoverable state. A fire’s intensity would decrease more quickly if both the human and robot extinguished it together, rather than working separately on different fires.

The probability that a fire would increase to level 4 was higher if the environment had high dryness conditions, and the probability that a fire would spread to neighboring houses was higher if the environment had high wind conditions. Both wind and dryness conditions were reported by noisy sensors on a \([0,9]\) integer scale. The team had to decide the best way to work together depending on the environmental conditions. The first ‘base’ task the team trained on had no wind or dryness \((w = 0\) and \(d = 0\)). The second training task, which was the first perturbation the team
experienced, had some wind and no dryness \((w = 5 \text{ and } d = 0)\). The third and last training task had no wind and some dryness \((w = 0 \text{ and } d = 5)\). The teams were then tested on combinations of wind and dryness. In the AdaPT condition, a Q-value function was learned using Q-learning for each of the three training tasks, and these 3 value functions were given as input to the algorithm. In the PRQL condition, a policy was learned from each of the training tasks, which became the library given as input.

Information about the state of the world, available actions, consequences and goal criteria were encoded for the robot in an MDP. The MDP for this task was formulated as a tuple \(\{S, A_H, A_R, T, R\}\), where:

- \(S\) is the finite set of states in the world. Specifically, \(S = < I_A, I_B, I_C, I_D, I_E >\), \(I_i \in [0, 4]\), where each value \(I_i\) in the vector denotes the intensity of fire \(i\) and is an integer ranging from \([0,4]\). Intensity 0 indicates no fire, 1-3 indicate low-to high-intensity fires and 4 indicates burnout. The size of the state space is 3,125.

- \(A_H\) is the finite set of actions that the human can execute. \(A_H \in \{A, B, C, D, E, \text{WAIT}\}\). Actions \(A, B, C, D, E\) each represent the action of the human toward extinguishing fires \(A, B, C, D, E\), respectively. \(\text{WAIT}\) is a no-op in which the human takes no action.

- \(A_R\) is the finite set of actions that the robot can execute. \(A_R \in \{A, B, C, D, E, \text{WAIT}\}\) represent the robot’s actions toward extinguishing the fires.

- \(T : S \times A_H \times A_R \rightarrow \Pi(S)\) is the state-transition function that specifies the probability of transitioning to state \(s'\) given the current state \(s\), human action \(a_h\) and robot action \(a_r\). We write the probability of the transition as \(T(s, a_h, a_r, s')\). For perturbed variants of the task, the wind and dryness levels affected the transition function. The probability of a fire spreading was based on both the wind level and the intensity of the fire. If the wind level \(w\) was greater than 0, the probability of an intensity 3 fire spreading was \((w \times 10 + 10)\%\), while the
probability at intensity 2 was 10% less and intensity 1 was a further 10% less. If the wind spread to a neighboring house, the intensity of the neighboring fire increased by one level. At most, five fires increased in intensity from the fire spreading at each time step. If the dryness level \( d \) was greater than 0 and a fire was at level 3, the probability of a house burning down during the next step was \( (d \times 10 + 10)\% \).

\( T \) also modeled the effect of human and robot actions on the fires for all tasks. Both team members working on the same fire decreased its intensity by three levels with 90% probability and by two levels with 10% probability. Separately extinguishing fires resulted in a decrease of one level with 90% probability and of two levels with 10% probability.

- \( R: S \times A_h \times A_r \times S' \rightarrow R \) is the reward function that specifies the immediate reward for a given state \( s \), human action \( a_h \), robot action \( a_r \), and next state \( s' \). We write the reward as \( R(s, a_h, a_r, s') \). For the experiment task, -10 reward was assigned for each time step, and an additional negative reward was added in direct proportion to the intensities of the fires at each time step. At task completion, -100 reward was assigned for each house that had burned down.

In our experiments, we used \( \gamma = 1 \) to represent a finite horizon problem. Other parameters were initialized as follows: \( \tau = 0, \Delta \tau = 0.01, H = 20, \epsilon_{sugg} = 0, \epsilon_{acc} = 2, \epsilon = 0.1, \) and \( \alpha = 0.05 \).

We first compared AdaPT and PRQL over time to evaluate how quickly each algorithm learned a new task. Figure 3-4 shows that the average accumulated reward of both algorithms are similar after a number of iterations. However, AdaPT is able to learn much more quickly in the beginning, which can be especially beneficial for large, complex problems.

We then compared the algorithms given limited simulation time because in a real-world scenario, the robot will have limited time to make a decision. Here, we include AdaPT, PRQL, and Q-learning from scratch. Because standard PRQL starts with an uninformative prior rather than an intelligent one, we also included a comparison
Figure 3-4: This chart compares the performance of AdaPT and PRQL on the fire task as the number of iterations increase. After every 1000 iterations, the task is executed and the average accumulated reward received is shown over time.

Figure 3-5: This chart compares the performance of AdaPT, PRQL, and Q-learning on the fire task when PRQL is seeded with different priors. The x-axis displays the wind and dryness values of the test tasks (e.g. (7,3) represents a task with wind=7 and dryness=3). The y-axis shows average accumulated reward received on an execution of the task after learning on the test task for 1500 iterations. Less negative reward denotes better performance on the task.
to PRQL with different priors to analyze the effect of the prior. To do this, we ran PRQL three times, each time initialized with one of the Q-value functions learned from training. We then took the maximum and minimum reward PRQL received, which corresponded to the reward from using the best and worst prior from training respectively.

All the algorithms simulated on the training tasks for 400,000 iterations and on the new task for 250 ms (human reaction time), which corresponded to 1500 iterations. Figure 3-5 shows the performance of these algorithms on various test tasks, averaged over 50 simulation runs. AdaPT performs significantly better than the original PRQL (with an uninformative prior) and Q-learning on most test tasks. This is because it takes more full advantage of previously learned value functions by directly adapting them rather than starting with a new uninformative policy guided by past knowledge.

PRQL performs well with the best prior but achieves significantly lower performance than AdaPT when using the worst prior. Thus, if PRQL were seeded with an intelligent prior, it could perform well, but it does not have the capability to automatically do this. From previous transfer learning approaches, it is evident that determining which training task is most similar to a new task is not trivial. The AdaPT algorithm, however, provides the framework automatically to choose the value function that is most similar by simultaneously adapting all and choosing the best for execution.

3.3.2 GridWorld Task

The second domain was a collaborative version of a gridworld task similar to the one used in [17]. The objective was for the team, including the human agent and the robot agent, to reach the goal location while maximizing the number of tokens obtained, as in [1], and minimizing the number of pits crossed over. If the two agents collectively get a token, the team receives higher reward than if they separately collect tokens. The MDP for this task was formulated as a tuple \( \{S, A_H, A_R, T, R\} \), where:

- \( S \) is the finite set of states in the world. Specifically, \( S = < (H_{row}, H_{col}), (R_{row}, R_{col}) > \).
Table 3.1: This table compares AdaPT, PRQL, and Q-learning on the collaborative grid world task given limited simulation time. We include the standard PRQL approach with an uninformative prior and also PRQL with the best and worst priors from the training tasks. We report the mean and standard error for each, averaged over 100 randomly selected test tasks.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean</th>
<th>Standard Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdaPT</td>
<td>-24.16</td>
<td>1.43</td>
</tr>
<tr>
<td>PRQL (Uninformative Prior)</td>
<td>-34.25</td>
<td>0.93</td>
</tr>
<tr>
<td>PRQL (Best Prior)</td>
<td>-21.79</td>
<td>1.32</td>
</tr>
<tr>
<td>PRQL (Worst Prior)</td>
<td>-30.91</td>
<td>0.58</td>
</tr>
<tr>
<td>Q-learning</td>
<td>-35.57</td>
<td>0.84</td>
</tr>
</tbody>
</table>

The human's position on the grid is indicated by \((H_{row}, H_{col})\) and the robot's position is indicated by \((R_{row}, R_{col})\). Each value denotes the row or column of the agent's location on the 10x10 grid. The size of the state space is 10,000.

- \(A_H \in \{UP, DOWN, LEFT, RIGHT, WAIT\}\). These actions correspond to moving in each of the four cardinal directions on the grid. \(WAIT\) is a no-op in which the human takes no action.

- \(A_R \in \{UP, DOWN, LEFT, RIGHT, WAIT\}\) represent the robot's actions toward moving around the grid.

- \(T : S \times A_H \times A_R \rightarrow \Pi(S)\) specifies the transition function. The stochasticity in the task is modeled similar to grid world tasks, in which the robot has 80% chance of taking the action it chose, 10% chance of moving to the left of that action, and 10% of moving to the right.

- \(R : S \times A_H \times A_R \times S' \rightarrow R\) specifies the reward function. For the task, -1 reward was assigned for each time step. The human or robot earned +1 if they individually collected a token, but if they collected the token together, they received +5, to encourage collaboration. If either the human or robot landed on a pit, -5 reward was given. The team received +20 reward when both the human and robot reached the goal location.
Figure 3-6: This figure shows a pictorial representation of the 10x10 gridworld task domain. The yellow tokens and black pits are placed in fixed locations on the grid. The four green locations were the four training task goal locations. The team was then tested on 100 random test goal locations.

Figure 3-7: This chart compares the performance of AdaPT and PRQL on the gridworld task as the number of iterations increase. After every 1000 iterations, the task is executed and the average accumulated reward received is shown over time.
The 10x10 grid had fixed token and pit locations, as shown in Figure 3-6. The team trained on four different training tasks, each of which involved a different goal location. The four goal locations were the four corners of the grid, specified by the green cells in the figure. Similar to the fire domain, we first compared AdaPT and PRQL over many iterations to evaluate how quickly they learned given a new task. The test task was generated with a new randomly selected goal location. Figure 3-7 shows that PRQL is able to perform well over time but it takes many iterations to learn, while AdaPT is able to learn much more quickly. This difference is more clearly seen in this domain, as compared to the fire task, as it is a larger problem and can take much longer to learn without an intelligent prior.

We then compared the performance of AdaPT, PRQL with different priors, and Q-learning from scratch given limited simulation time. We averaged the reward obtained over 100 random goal locations. All the algorithms simulated on the training tasks for 1,000,000 iterations and on the new task for 10,000 iterations.

Table 3.1 shows the performance of these algorithms. AdaPT performs significantly better than PRQL with an uninformative prior and Q-learning from scratch. We then initialized PRQL with the value function learned from each of the training tasks and chose the highest and least reward received, corresponding to the best and worst priors respectively. When PRQL is initialized with the worst prior, it achieves lower performance than AdaPT, but with the best prior, it performs better. However, as noted earlier, PRQL does not have the framework to determine this best prior, while AdaPT is automatically able to determine similar tasks.
Chapter 4

Human-Robot Perturbation
Training Model

While our algorithm AdaPT provides a framework for adapting to new task variants, it does not support real interaction with people. We thus include a co-learning framework and a bidirectional communication protocol to better support teamwork. These two features combined with our AdaPT algorithm form our computational human-robot perturbation training model.

4.1 Co-Learning Framework

Often, robots solve complex problems in simulation by practicing for many iterations using reinforcement learning. When working with a human, this is impractical as it can take hours or even days to compute optimal policies for complex problems. Furthermore, robots learning everything about a task offline is not representative of the way people learn together in a collaborative way.

Thus, to better support interaction with people, we add a co-learning framework through human-robot practice sessions. The robot simulates between each interaction with the human, which allows the robot to refine its policy or 'catch up' its learning before working with the person again. This co-learning framework is a way for the human and robot to learn together as human teams would.
4.2 Human-Robot Communication Protocol

In effective collaborative teams, communication between team members is also very important [26]. The algorithm Human-Robot-Communicate in Figure 4-1 describes how the robot communicates with the human in order to coordinate task execution. At each step, one of the teammates initiates communication. The initiator can either suggest actions for both teammates or simply update the other teammate on what their own action will be, allowing the teammate to make a decision accordingly. If a suggestion is made, the other teammate has the option to accept or reject it.

When the robot is initiating, it can either suggest a joint action or update the human on its own action. It suggests a joint action only if the optimal joint action is much better than the ‘average.’ The average is calculated as the expected value over all possible joint actions where the robot is assumed to act optimally and the human can take any possible action. If the optimal is not much better than the average, it is not necessary for the robot to suggest because no matter what the human does, the reward received will be similar. When the human initiates and provides a suggestion, the robot decides whether to accept based on how close the human’s suggested joint action is to the optimal. If the human’s suggestion is not much worse than the optimal in terms of expected discounted accumulated reward, the robot accepts. If there is a large difference between the optimal and the human’s suggestion, it is best for the robot to reject, as this will greatly increase the reward the team receives.

There are two threshold values, \( \epsilon_{\text{sugg}} \) and \( \epsilon_{\text{acc}} \), that help determine for each communication step, whether or not the robot should suggest or accept respectively. A low threshold for suggesting means that the robot is being less risky by always giving suggestions to the human that it is confident are best for the team. A high threshold means that the robot trusts the human to make the right decisions, and it will only suggest if the optimal joint action reward is much higher than the average reward. For accepting human suggestions, a low threshold means that the robot will reject human suggestions very frequently. If it is set to be too low, this can be detrimental to team performance if two equally-good actions have slightly different Q-values, and
decisions are sensitive to these small changes. A high threshold means that the robot will accept human suggestions even if they don’t yield high reward. This can be useful if the robot is trying to act according to human preference rather than according to the optimal Q-value function.

As an interesting next step, these threshold values can be automatically modified in training to reflect expertise in the specific domain or confidence during task execution. If the robot has much more information or confidence about the task than the human does at a particular time, the robot can suggest more often, and more often reject human suggestions. If the human is known to have more domain knowledge or confidence, the robot can suggest less and accept human suggestions more, even if that contradicts the optimal actions calculated from the stored Q-value function.

The communication algorithm Human-Robot-Communicate takes as input a tuple that specifies the human action $a_h^*$ and the robot action $a_r^*$ before the communication step. If the human is initiating communication, these actions come from human input. If the robot is initiating, the actions come from calculating the optimal joint action. We input two threshold parameters: $\epsilon_{\text{sugg}}$, to select whether to make a suggestion or an update; and $\epsilon_{\text{acc}}$, to decide whether to accept or reject the suggestions made by the human. The joint Q-value function $Q(s, a_h, a_r)$ and the robot Q-value function $Q(s, a_r)$ are also taken as input. Using these parameters and human input, the Human-Robot-Communicate function returns the final joint action agreed during communication.

The way this is incorporated into AdaPT is that at each time step, instead of simply calculating the optimal joint action $<a_h^*, a_r^*>$, as is done in line 6 in Update-QValues, the optimal joint action or human input $<a_h^*, a_r^*>$ is passed into Human-Robot-Communicate. The communication algorithm is then executed with human input being received as necessary through text or speech, and then a final joint action is returned. Update-QValues then resumes execution by using this joint action and receiving the next state.

When the robot is initiating communication (line 1, Case 1), it calculates the optimal action $a_r'$ over all possible subsequent human actions (line 2). If the joint action
Algorithm: Human-Robot-Communicate ($<a^*_h, a^*_r>$, $\epsilon_{sugg}$, $\epsilon_{acc}$, $Q(s, a_h, a_r)$, $Q(s, a_r)$)

1. **Case 1:** Robot is the initiator
   2. Calculate the optimal $a'_r$ using $Q(s, a_r)$
      \[ a'_r = \arg \max_{a_r} Q(s, a_r) \]
   3. if $(Q(s, a^*_h, a'_r) - E_{a_h} [Q(s, a_h, a'_r)]) > \epsilon_{sugg}$
      4. Return $<a^*_h, a^*_r>$
   5. else
      6. Return $<a^*_h, a'_r>$ where $a^*_h$=received human input

7. **Case 2:** Human is the initiator
   8. Calculate optimal $a'_r$ given human action $a^*_h$
      \[ a'_r = \arg \max_{a_r} Q(s, a^*_h, a_r) \]
   9. if $a^*_h$ == null (human provided an update)
      10. Return $<a^*_h, a'_r>$
   11. else (human provided a suggestion)
      12. if $(Q(s, a^*_h, a'_r) - Q(s, a^*_h, a^*_r)) > \epsilon_{acc}$
         13. Return $<a^*_h, a'_r>$
      14. else
         15. Return $<a^*_h, a^*_r>$

Figure 4.1: The Human-Robot-Communicate algorithm provides a computational framework for the robot to make decisions when communicating with a person.

has a much higher value (as determined by the threshold $\epsilon_{sugg}$) than the expected value over all human actions of $Q(s, a_h, a'_r)$ (line 3), the robot suggests the optimal joint action in line 4. If the joint action’s value is not much better than the average value over all human actions, the robot chooses to simply update the human, allowing the person to choose his or her own action (line 6).

If the human is initiating communication (line 7, Case 2), the joint action passed in, $<a^*_h, a^*_r>$, is either a suggested joint action provided by the human, or an update from the human with $a^*_r$ being null. The robot first calculates the optimal robot action $a'_r$ given the human action $a^*_h$ (line 8). If the human only communicated an update and $a^*_r$ is null (line 9), the robot would simply return $<a^*_h, a'_r>$ in line 10.
If the human made a suggestion, the robot would calculate, given $a^*_h$, the difference between taking the calculated optimal robot action $a'_r$ and taking the action suggested by the human $a^*_h$. If the difference is more than $\epsilon_{acc}$ (line 12), the robot rejects the suggestion and returns $<a^*_h, a'_r>$ in line 13. If the calculated optimal joint action is not much better than the suggestion, the robot accepts and returns $<a^*_h, a^*_r>$ in line 15. We use this two-way communication protocol because providing suggestions and updates is a natural way for a team to coordinate on joint tasks.
Chapter 5

Human Subject Experiments

We conducted human-subject experiments with both simulated and embodied robots to confirm that our human-robot perturbation training model enables a robot to draw from its library of previous experiences in a way that is compatible with the process of a human partner learning through perturbations.

This is a challenging task, as evidenced by previous human teamwork studies which demonstrated that human teams with members possessing accurate but dissimilar mental models performed worse than teams with members with less-accurate but similar mental models [28].

5.1 Aim of Experiments

Specifically, we aim to answer the following questions through our experiments:

1. Does our human-robot perturbation training model, which includes the AdaPT algorithm, the co-learning framework, and the communication protocol, support a person in learning through perturbations and achieving high team performance under new task variants?

2. Can a team achieve high performance by simply experiencing perturbations regardless of the algorithm the robot uses or does the robot’s algorithm affect team performance significantly?
3. Does training on perturbations using AdaPT sacrifice performance on the base task, compared to procedurally-trained teams who train repeatedly only on the base task?

4. Can training in a simulated environment using AdaPT transfer to produce effective team performance with an embodied robot?

5.2 Hypotheses

To answer the first two questions, we compare two models for perturbation training, one using our AdaPT algorithm and the other using the standard Q-learning algorithm with no library. Both conditions include the co-learning framework and the communication protocol. We include this comparison to analyze the effect of the human’s experience of perturbations vs. the effect of the robot’s algorithm. If both algorithms achieve high team performance, it might be enough to just induce perturbations in the human’s training and implement any algorithm for the robot. However, if AdaPT performs much better than perturbation Q-learning, the algorithm that the robot uses is key to achieving high team performance.

**H1** AdaPT teams that train on perturbations will achieve significantly better outcomes on objective and subjective measures of team performance compared with Q-learning teams that train on perturbations, when tested on both novel task variants and variants similar to the training tasks.

We hypothesize that the robot’s use of a library of prior experiences is compatible with human strategies for decision-making, as numerous studies have demonstrated that exemplar-based reasoning involving various forms of matching and prototyping of previous experiences is fundamental to our most effective strategies for decision-making [30, 13, 23]. For example, naturalistic studies have indicated that skilled decision-makers in the fire service incorporate recognition-primed decision-making, in which new situations are matched to typical cases where certain actions are appropriate and usually successful [23].
To answer question 3, we hypothesize that AdaPT will not sacrifice performance on the base task by training under perturbations. While human team literature indicates that procedurally trained teams, who train only on one task, will perform better than perturbation-trained teams on that task, we hypothesize that our model will maintain performance on the base task while also adapting to new task variants.

**H2** Teams that train on perturbations using AdaPT will achieve significantly better outcomes on objective and subjective measures of team performance as compared to teams that train procedurally on the base task using Q-learning for novel task variants. Both teams will achieve comparable measures of performance on the base task.

Finally, to answer the last question, we aim to demonstrate that human-robot training in a simulation environment using AdaPT produces effective team performance with an embodied robot partner.

**H3** Teams that train on perturbations using AdaPT in a simulation environment and then execute novel task variants with an embodied robot partner will achieve comparable measures of performance to teams that execute the novel task variants in the simulation environment.

This hypothesis is motivated by human-robot cross-training experiments which indicated that people were able to work effectively with a physical robot after training in a virtual environment [32]. The study used a virtual environment that displayed a virtual workspace and robot, which mirrored a physical task and robot. The virtual robot moved in the scene using the same controller as the physical robot. In our study, the simulation environment does not display a virtual robot; therefore, the question to be addressed is whether this result persists when the simulation environment supports the learning of strategies as sequences of discrete actions but does not support familiarization with robot motions.

For all hypotheses, we measured objective team performance using the reward accumulated during task execution. We also measured the number of times that
participants accepted and rejected the actions suggested by the robot, assuming that a larger proportion of accepted suggestions indicated that the participants agreed with the robot’s decisions with regard to completing the task effectively and quickly. Finally, we measured the participants’ subjective perception of team performance using a Likert-style questionnaire administered at the end of the experiment.

5.3 Experiment Methodology

Forty-eight teams of two (one person and one robot) were tasked with working together to extinguish a set of fires. The description of the task and computational model is elaborated in Chapter 3. Thirty-six of those teams performed both their training and testing in a simulation environment; 12 teams performed the training in a simulation environment and the testing with a physical robot partner. The experiment was conducted between-subjects. Each participant in the simulation-only experiment was randomly assigned to undergo either perturbation training using AdaPT, perturbation training using Q-learning or procedural training using Q-learning. All participants in the mixed simulation-hardware experiments were assigned to perturbation training using AdaPT.

Perturbation teams trained together on three different task variants, two times each. For the perturbation AdaPT condition, a value function was learned for each variant using Q-learning, and these Q-value functions became the library given as input to AdaPT. For the perturbation Q-learning condition, one set of Q-values was updated using Q-learning during the training on perturbations. This training scheme was deliberately chosen to consist of a relatively small number of interactions for learning complex joint strategies for coordination across multiple task variants. Procedural teams trained together on one variant of the task a total of six times. For these tasks, we used Q-learning throughout and transferred Q-values between tasks to learn one Q-value function over the six rounds. A pictorial representation of the structure of each of these conditions is depicted in Figure 5-1.

All teams were then tested on three new variants of the task and evaluated on
Figure 5-1: This figure shows the structure of the robot's learning for the conditions used in human subject experiments. The perturbation AdaPT condition stores a library and uses the AdaPT algorithm. The perturbation Q-learning condition updates one value function for all perturbed tasks. In the procedural Q-learning condition, one value function is updated based on repeated experience with a task.

their performance. All teams received the same three test variants of the task, which were different from the tasks used during training. To simulate time pressure, the teams were given a limit of 10 seconds in which to communicate with each other and make a decision about their next action.

For all sessions, the participant and the robot communicated using Human-Robot-Communicate. They alternated with regard to who initiated communication. During the test sessions, the first communication was always initiated by the robot in order to maintain consistency. The teammate initiating communication had the option to either suggest the next actions for both teammates or to simply update the other teammate on their own next action (i.e. which fire they planned to extinguish next).

5.3.1 Training Procedure

The experimenter introduced participants to the task using a PowerPoint presentation, which included a description of the fire scenario, the goal for the task, the human-robot communication protocol and a tutorial of the simulation interface. To give the robot equivalent initial knowledge before beginning the task, we ran Q-
learning offline for 500,000 iterations over the entire state space on a deterministic version of the task without wind or dryness. Each participant then completed two practice sessions in order to familiarize themselves with the interface. For the first practice session, participants received 15 seconds per time step to communicate with the robot to decide on an action. From the second practice session on, participants received only 10 seconds per step.

The experiment GUI displayed five fires in a row, representing the five house fires. Each fire was annotated with its intensity level, with 1 indicating low intensity, 2 medium intensity, 3 high intensity, and 4 burned out/unrecoverable. During task execution, the GUI prompted the participant to communicate with the robot and displayed feedback on its chosen actions.

After the familiarization phase, the participants performed three training sessions. Each session consisted of two execution rounds to provide the human and robot two attempts at the same variant of the task. This allowed the participant to learn from the first round and improve their coordination strategy for the second. Each round consisted of a complete episode, from the initial fire state to the goal state. Prior to the first round, the robot was provided 200,000 iterations to learn an effective joint policy for the task. Between the first and second rounds, the robot was again provided 200,000 iterations to refine its policy. This is the co-learning framework that allows the robot to simulate between each human interaction and co-learn with the human, as human team members do. The 200,000 iterations took between 10 and 50 seconds to compute, depending on the difficulty of the task. Participants were given a short questionnaire during this interval prompting them to respond to Likert-style statements regarding their subjective perceptions of the team’s and robot’s performance (similar to that used in [32]).

Among the procedurally trained teams, the human and robot trained on the same environmental conditions, \( w = 0 \) and \( d = 0 \), for all sessions. For perturbation-trained teams, the first training session was identical to the first session for the procedural training group. The second and third sessions, however, included slight changes to the environment that affected the progression of the fires: The second session
involved some wind \((w = 6)\) and no dryness \((d = 0)\), allowing for fires to spread. The third session involved no wind \((w = 0)\) and some dryness \((d = 6)\), resulting in more burnout from high-intensity fires. Information about wind and dryness conditions was provided to the participants and the robot with some noise. The robot thus trained on an approximate model of the environment. This decision was made to mirror real world circumstances where a robot does not always have an accurate model of the task.

### 5.3.2 Testing Procedure

Following the training sessions, participants performed three testing sessions. The first test-session task was similar to the procedural training tasks, in that it involved no wind or dryness; however, the fires were initialized at different intensities than in the training tasks. The remaining two test-session tasks involved different wind and dryness levels. These task variants were more drastic than the training sessions experienced by the perturbation-trained teams and involved non-zero levels of both wind and dryness: Test task 2 had \(w = 2\) and \(d = 9\), and test task 3 had \(w = 9\) and \(d = 2\). During the test phase, the robot was constrained with regard to the time allotted for learning and had to make a decision within a reaction time similar to that of a human (250 ms), which corresponded to 1,500 iterations.

**Simulation Experiment Protocol**

For the simulated robot experiments, all three testing sessions were executed with no deviation from the training protocol, using the same GUI screen.

**Robot Experiment Protocol**

For the embodied robot experiments, the participants trained in simulation and then worked with a PR2 robot during the testing sessions, which were identical to those from the simulation experiments. To represent each fire, three lights were placed underneath cups on a table and were controlled using an arduino. The lights were
activated in order to indicate the intensity of each fire: one light for low intensity, two for medium, three for high, and zero for no fire or burnout. Additionally, the same GUI screen from the training sessions was used to display the state of the fires, the team’s action choices and the time remaining for each time step. The participant and the robot communicated using a defined language structure. Google web speech recognition was used to identify human speech commands, and errors from the recognition system were manually corrected by the experimenter. A button was placed next to each of the five representations of fire for both the human and the robot. To extinguish a fire, the appropriate button was pressed by each teammate and the lights changed appropriately to reflect the subsequent state of the fire. Before the testing sessions began, participants were given a brief overview of the setup and the language protocol and worked with the robot to put out low-intensity fires in order to practice working with the PR2. Finally, the participants completed all three testing sessions. The experiment setup for the human-robot experiments is shown in Figure 5-2.
5.4 Human Subject Experiment Results

This section summarizes statistically significant results and insights obtained from the experiments. We define statistical significance at the $\alpha = .05$ level. For objective measures of performance, we measured reward accumulated during each test session and used the two-sided Mann-Whitney U-test. Differences in the proportion of accepts and rejects in each test session were evaluated using a two-tailed Fisher’s exact test. For subjective measures of performance, we used the two-sided Mann-Whitney U test to compare Likert-scale answers from the post-experiment questionnaire.

5.4.1 Simulated Robot Experiments

Thirty-six participants, all recruited from a university campus, were randomly assigned to one of three training conditions: perturbation training using AdaPT, perturbation training using Q-learning and procedural training using Q-learning. We tested for statistically significant differences between perturbation AdaPT and perturbation Q-learning treatments, as well as between perturbation AdaPT and procedural Q-learning treatments. Results are shown in Figure 5-3.

Perturbation AdaPT vs. Perturbation Q-learning

First, we compare the two algorithms for perturbation training, AdaPT and classical Q-learning, to evaluate the effect of the human’s experience of perturbations vs. the effect of the robot’s algorithm. We found that perturbation AdaPT-trained teams received statistically significantly higher reward on test task 1, the base task with $w = 0$ and $d = 0$ ($p = 0.0037$), and on test task 3 with $w = 9$ and $d = 2$ ($p = 0.0491$) compared to perturbation Q-learning teams. This was not the case for test task 2, however, most likely because it was similar to the final training session (high dryness) and thus the Q-values reflected the last strategy well. To confirm that the Q-values are overridden at each session and that perturbation Q-learning only works well on tasks similar to the last training session, we ran additional simulation experiments over a range of values (summarized in Table I) and different orders of training sessions.
Figure 5-3: This chart compares the performance of the three different conditions used in human subject experiments with simulated robots: perturbation AdaPT, perturbation Q-learning, and procedural Q-learning. The y-axis shows average accumulated reward received on the execution of the task with the human subject after the robot simulated on the test task for 1500 iterations. The x-axis displays the wind and dryness values of the test tasks (e.g. (2,9) represents a task with wind=2 and dryness=9). Less negative reward denotes better performance on the task.

We found that perturbation Q-learning teams performed well on tasks most similar to the last training session, but never performed statistically significantly better than AdaPT.

We also observed that perturbation AdaPT participants accepted the robot’s suggestions more frequently than perturbation Q-learning participants on test task 1 ($p = 0.08$), supporting H1. Subjective measures from the post-experiment questionnaire indicated that participants agreed more strongly that they felt satisfied with the team’s performance on test task 1 ($p < 0.01$), lending further support to H1. This result shows that the algorithm the robot uses, rather than just the human’s experience of perturbations, is key to achieving high team performance.

**Perturbation AdaPT vs. Procedural Q-learning**

We also compare perturbation and procedural training, as is done in human team studies [19], to evaluate the benefit of perturbations in training. For the first test task, which had identical environmental conditions to the procedural training sessions ($w = 0$ and $d = 0$), there were no statistically significant differences in reward between
the two training methods. They received similar accumulated reward, on average.

For the second test task \((w = 2 \text{ and } d = 9)\), a substantial variant of the base task and the perturbed training tasks, perturbation AdaPT-trained teams received statistically significantly higher reward than procedural teams \((p = 0.0045)\), supporting H2. In the third test task \((w = 9 \text{ and } d = 2)\), we observed that perturbation AdaPT teams got higher reward on average, but this result was not statistically significant.

Subjective measures from the post-experiment questionnaire further support H2. Participants in the AdaPT treatment agreed more strongly that the team adapted well to different scenarios, as compared to participants in the procedural-training treatment \((p = 0.002)\). AdaPT participants also agreed more strongly that their teammate gave useful suggestions during testing, compared to procedurally trained participants \((p = 0.02)\). These results show that our perturbation training model can support a human and a robot in generalizing to new task variants without sacrificing performance on the base task that procedural teams train on.

### 5.4.2 Embodied Robot Experiments

Twelve participants were recruited from a university campus for robot experiments and were all assigned to the perturbation AdaPT condition. We compared objective measures of performance between these embodied robot experiments and the perturbation AdaPT simulation experiments in order to determine whether people are able to effectively train in simulation and then work with an embodied robot. The results are shown in Figure 5-4.

Interestingly, teams that worked with an embodied robot performed better on all three test tasks, on average, than teams that worked with the robot in simulation, thereby supporting H3. While the performance improvement working with an embodied robot was not statistically significant, it is nonetheless encouraging that no degradation in performance was observed and that simulation training promoted effective human-robot teamwork.
Figure 5-4: This chart compares the performance of teams who tested with a simulated robot vs. an embodied robot. The y-axis shows average accumulated reward received on the execution of the task with the human subject after the robot simulated on the test task for 1500 iterations. The x-axis displays the wind and dryness values of the test tasks (e.g. (2,9) represents a task with wind=2 and dryness=9). Less negative reward denotes better performance on the task.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

Today, robots often perform repetitive tasks and are deployed to work separately from people. By taking advantage of the relative strengths of humans and robots in joint human-robot teams, we can potentially accomplish what neither can do alone. In order to have fluid human-robot collaboration in these teams, it is important to train the teams to achieve high performance and flexibility on a variety of tasks. This requires a computational model that supports the human in learning and adapting to new tasks.

In this work, we design and evaluate a computational learning model for perturbation training in human-robot teams, motivated by human team training studies. Perturbation training involves inducing disturbances during training that require a team to handle unexpected circumstances or new variants of a task. We first present a learning algorithm AdaPT that augments the PRQL algorithm [17] to learn more quickly in new task variants. Rather than starting with an uninformative prior and using past knowledge to guide the learning in new tasks, we directly adapt previous knowledge. This is similar to the way people perform recognition-primed decision making, where they match new experiences to previous situations they have seen [23].

We compared the performance of AdaPT and PRQL over two domains, a disaster
response domain that involved extinguishing fires based on environmental conditions and a gridworld task similar to ones used in previous work [17, 1]. We found that AdaPT achieved higher performance more quickly than PRQL because PRQL starts with an uninformative prior and takes much longer to learn a new task. We seeded PRQL with different priors to analyze the effect of the prior on the learning, which we found to have a significant impact on performance. When PRQL was initialized with a very good prior, it achieved high performance, but performed poorly when initialized with an unintelligent prior. However, computing this prior can be difficult and PRQL does not provide a function for selecting this prior. In contrast, AdaPT automatically provides this capability by updating all previous strategies and finally choosing the one that works best.

While the AdaPT algorithm allows the robot to adapt to new tasks given previously learned tasks, it does not support interactions with people. We thus include a human-robot co-learning framework and a bidirectional communication protocol to form a computational model for perturbation training. In the co-learning framework, the robot simulates between each interaction with the person, which allows the robot to learn alongside the human. Often, reinforcement learning algorithms will learn completely offline, but in our work, we want the robot to learn with the person similar to the way people learn collaboratively with each other.

The bidirectional computational communication protocol allows the robot to make communication decisions when working with a person. When the robot is initiating communication, it must decide whether to suggest a joint action or simply update the person on its own action. When the human is initiating and provides a suggestion, the robot must determine whether to accept or reject it. We define in terms of the value function how the robot makes each of these decisions when working with a person.

To evaluate whether this model supports learning with real people, we conducted human subject experiments comparing two models for perturbation training and a model for procedural training. We included an alternative perturbation training algorithm, standard Q-learning with no library, to determine whether a human’s experience of perturbations is enough to achieve high team performance or if it is important
for the robot to have the right model. If both algorithms achieve high team performance, it might be enough to just induce perturbations in the human’s training and implement any algorithm for the robot. However, in our experiments, we found that there was a significant difference between the two algorithms. Perturbation-trained teams using AdaPT outperformed perturbation-trained teams using Q-learning in both objective \((p = 0.0037 \text{ for test task 1 and } p = 0.0491 \text{ for test task 3})\) and subjective measures of performance on multiple task variants. This suggests that the robot’s algorithm is important in achieving high team performance.

We also included a comparison of perturbation training using AdaPT to procedural training, in which team members practice repeatedly on one task, which we call the base task. The goal was to determine if AdaPT maintains performance on this base task after training on multiple perturbations. We found that perturbation AdaPT teams do not sacrifice performance on the base task, on average, when compared to procedurally trained teams.

Finally, we analyzed the effect of introducing an embodied robot, the PR2, into the task. All participants trained in simulation but half tested with the simulated robot while the other half tested with an embodied robot. We found that human-robot training in a simulation environment using AdaPT resulted in effective team performance with the PR2. This is a promising result, as this means people may be able to train and learn strategies in the task with low-cost simulated robots and then work effectively with embodied robots.

6.2 Future Work

While this work is a step in improving human-robot team training, the approach can be augmented to be more scalable to larger teams and to larger problems. In our work, we consider teams of two consisting of one human and one robot. However, in many real-world scenarios, there will be many team members collaborating on a task. Simply adding more agents to our current framework would be intractable, as we would need to consider and model the actions of all other team members. To reduce
the complexity of the problem, as previous work [55] has shown, agents can consider the actions of just a few other agents most closely related to their role. Further, in real-world tasks, there may be a cost associated with communication, which means that the robot must decide when and what to communicate. They may not be able to communicate on every decision, as is done in this work.

To scale the approach to larger problems, we also need to incorporate hierarchical or approximation techniques [6, 48] that allow a robot to learn quickly in complex tasks with large state spaces. Hierarchical frameworks allow the robot to learn higher-level structure of skills rather than simple low-level actions. Learning in this structure can be quicker than standard reinforcement learning approaches. Approximation techniques can also allow the robot to generalize information gained at every task execution to more quickly update its knowledge.

We also consider a simplified disaster response domain in our experiments, but working with real data often poses additional challenges. Data can be noisy and there can be other factors affecting the task that are not modeled in the representation of the task. Testing our model with real data will be helpful in analyzing how well the algorithm and model generalizes. Also, to deploy a real robot, we would need to augment the robot's capabilities to perform more complex tasks that may require additional manipulation, motion planning, vision, and sensing.

Taking further steps, it would be interesting to provide a richer framework for the robot to adapt with people. For example, the robot can include a more explicit model of human behavior and use this to anticipate human actions and maintain a belief of human goals and intentions. This is a difficult problem because learning these unobservable factors affecting human behavior can be intractable if the robot has to maintain all possible beliefs. The robot can also adapt to various people based on their preferences. This requires maintaining multiple models, determining a person's preference online, and adapting further to personalize for the individual. Including these types of adaptive features can significantly improve collaboration in human-robot joint teams.
Bibliography


