Training a Robot to Collaborate with a Human Partner

Michael Van der Linden
Advisors: Corina Grigore, Professor Brian Scassellati

May 3, 2018

Abstract

As robots become more pervasive, more attention has been drawn to the field of human-robot collaboration. Tasks that require the precision or strength of a robot combined with the intelligence or dexterity of a human are ideal applications. But setting up safe, productive human-robot partnerships is limited by the flexibility, adaptability and usability of existing robotic platforms. The goal of this project is to build a machine-learning system that allows the Baxter robot to quickly learn through iterative trials how best to support a human partner in a collaborative task. The robot starts with no domain knowledge, and must learn the rules of the task and the preferences of its partner only through feedback from the human.

The system I built uses Q-learning to build a decision-making matrix that is improved with each iteration of the task. At each time step, the robot begins an action, which the human can accept or reject, applying a positive or negative reward. If the human accepts the action, they can evaluate it further on a 1-10 scale. The robot learns the rules of the task very quickly and then gradually learns its partner's preferences. Since the task is successfully completed with every trial, the human-robot team is productive right from the start. I simulated trials using various solutions to the problem of balancing exploitation of good actions with exploration of new ones, and each solution offered a slightly different trade-off between short-term improvement and long-term peak performance. Finally, I ran a short series of live trials with Baxter that demonstrate how quickly the robot learns the task, resulting in successful collaboration after just a few iterations.

Background

As robots become more pervasive, more attention has been drawn to the field of human-robot collaboration. Tasks that require the precision or strength of a robot combined with the intelligence or dexterity of a human are ideal applications. However, two significant obstacles to human-robot collaboration are the complexity of programming a robot to help a human with a particular task and the time cost of training humans to use the robot effectively. Machine learning can overcome both of these actions. Rather than investing in a knowledge-based program for a particular application, manufacturers can give robots the ability to learn to perform any task that they are physically capable of. Moreover, human partners do not need to be trained on the robot beyond basic safety and interfacing. Instead, they train the robot to respond correctly through natural patterns of interaction and feedback. In the future, the 1950s vision of personal, general-purpose robots to assist with day-to-day tasks might be possible with machine learning.

Goals

The central goal of this project is to build a machine learning system that allows human participants to train Baxter to intelligently help the participant in a collaborative task. Baxter will begin with a set of basic actions but no knowledge of the task at hand. Through interactive training sessions, the human participant will provide feedback at each step of the task to improve Baxter's decision-making ability. The primary deliverable is for Baxter to emerge from the training sessions with a decision-making data structure enabling it to demonstrate intelligent collaboration with its partner in a live demo. A secondary emphasis is placed on creating intuitive and speedy training sessions, so that users could easily train the robot on a variety of tasks without special training of their own. A central design philosophy for Baxter's learning curve is "decent first, refined later." This means that I want Baxter to learn an acceptable level of performance as fast as possible, and from there, improve incrementally through many trials. With this goal in mind, I am hoping to see logarithmic growth in Baxter's performance that would enable productive collaboration within the first few trials.

Collaborative Task

This project is a branch of Corina Grigore's human robot collaboration research. The project focuses on developing frameworks and algorithms that enable the robot to learn how to predict supportive behaviors during a human-robot collaboration (HRC) task. These behaviors are intended to help human workers more efficiently perform assembly tasks, like furniture assembly. For this project, we are currently using a flat pack furniture set, and focusing on a small chair requiring complex assembly. The assembly steps are organized according to a hierarchical task model, which is a branching tree of substeps such as "build seat subassembly" or "collect parts for leg 1." (fig 1.) The child substeps at every node must either be completed in a specific order or in any order.

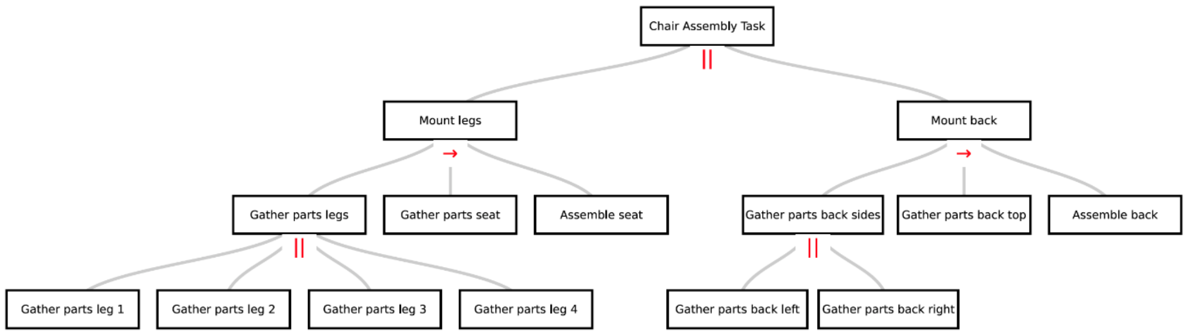
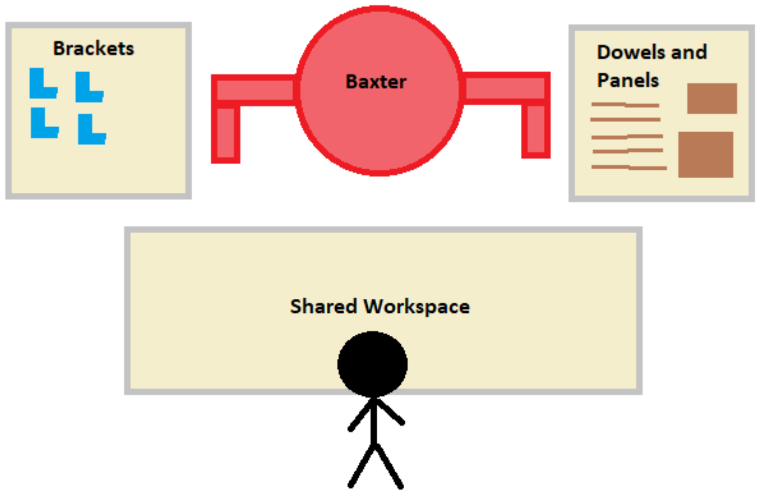


fig 1. Chair assembly hierarchical task model. (from Grigore, 2018) [1]

The robot starts with two tables of components within its reach and an empty workspace shared with the human partner. (fig. 2) It also has a set of actions it knows how to execute, with controller support for each. (fig. 3) Most of these are 'bring [component]', with the exception of 'hold,' in which Baxter offers its gripper to hold a subassembly in place for the human to work on. The 'execute action' subroutines are divided into two phases. First, the robot grabs a part and holds it over the workspace (or, for 'hold', opens its gripper over the workspace). The human can accept or reject the action with green and red buttons on the "wrists" of the robot. If the human accepts, the robot releases the component (or, for 'hold', closes the gripper). If the human rejects, the robot returns the component to the component tables (or withdraws the offered gripper). After accepting a hold, the human must press the green button one more time to release and retract the gripper. (fig. 4)



bring dowel
bring long dowel
bring seat panel
bring back panel
bring front bracket
bring back bracket
bring top bracket
hold

fig 2. Diagram of Baxter, supplies, and workspace with human partner

fig 3. Baxter's action set

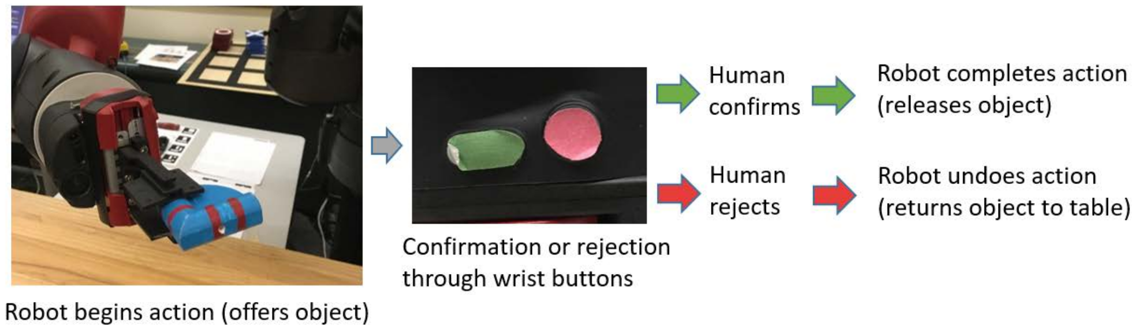

Robot begins action (offers object)
Confirmation or rejection through wrist buttons
Human confirms
Robot completes action (releases object)
Human rejects
Robot undoes action (returns object to table)

fig 4. Action control flow

The state of the robot's environment is determined by the objects that are in the workspace. Each action changes the state by adding an object, or in the case of 'hold', transforming a collection of objects into a single subassembly. Since the only perceptual systems implemented in this project are two very limited-FOV cameras on the arms, only used to accurately pick up objects, the robot keeps track of the state of the world through dead reckoning. The system is static, discrete, and highly controlled, so dead reckoning works quite well. The robot starts with no knowledge of the chair task, the rules of the hierarchical task model, or the human preferences. It must learn through human interaction what actions are desired and forbidden in each state.

Finally, the robot also keeps track of how many of each component are available for it to grab on the component supply tables. Like any reasonable person, it limits its 'bring [component]' actions to components that are still available. This dramatically speeds up the end of the task, when few components are left on the supply tables.

Implementation

The robot is controlled through the Robot Operating System, using an invaluable behavioral controller written by Alessandro Roncone and Olivier Mangin, and the framework of a chair-assembly controller written by Corina Grigore, all of the Yale Social Robotics Lab. Because of the foundation laid by these researchers, I was able to interface with the Baxter robot using high-level behavioral commands (eg. "grab front bracket") without having to worry about the kinematics of Baxter's arms, collision avoidance, or using camera feedback for precision grasping.

With the goal of fast learning in mind, I decided to use Q-learning as the learning method. Since Q-learning works by trialing various pathways from state to state from a predefined start state to an end state, it was a natural choice in a procedural task like chair assembly. In addition, the various states (combinations of objects in the workspace) were discrete and easily indexed in a Q-matrix. I use a Q-matrix in which the rows are states and the columns are actions.

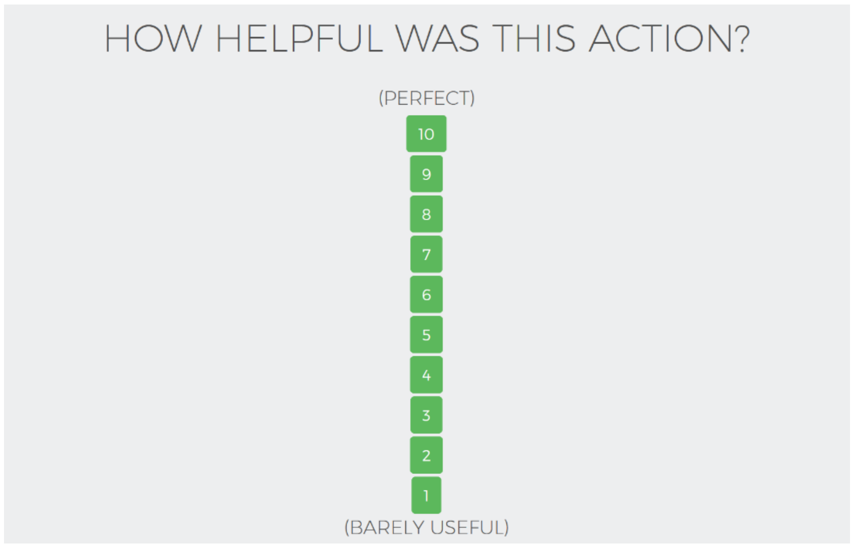
Early on, I itemized all the reachable states within the chair assembly hierarchical model, relying on the fact that other configurations of components in the workspace would be impossible as long as the human rejected any action that would take the task outside the hierarchical task model. Given this framework, it made sense to implement the Q matrix as a list of lists within python, initialized to 0. Moreover, to drive state transitions, I wrote out a transition matrix by hand which maps a (state, action) pair to a next state (or -1 if it leads to a state outside the hierarchical task model). This made state transitions trivial, so long as the human player makes sure that they and the robot stick to the rules of the hierarchical task model. If I had more time, the first thing I would do is refactor these data structures to make them more general. Each state, instead of arbitrarily indexed within the HTM, would be a binary vector of the components that may or may not exist in the workspace, allowing for any combination of components. Instead of a fixed-size matrix, Q would be a dictionary whose keys are (state, action) pairs and which would expand as more states are encountered. Instead of looking up an entry in a transition matrix, the state transition function would calculate the next component vector based on a (state, action) pair. This would allow the user to specify any set of rules they like, bounded only by the physical objects available to work with. Still, this flexibility is not central to the project, and I was still able to demonstrate sophisticated learning within the bounds of the HTM.

The main procedural loop as trials progress is described in fig 6. First, the robot uses one of the multi-armed bandit functions (see below) to choose an action to try. It begins the action and waits for positive or negative human feedback, delivered through the buttons on the robot's "wrist." If the human gives negative feedback, the robot assigns a reward of -1 to that (state, action) pair, undoes the started action, and re-chooses an action within the same state. If the human gives positive feedback, the robot completes the action and then waits for nuanced feedback, delivered via a rosbridge-connected web interface (fig 5). The human can give the action a score from 1 to 10, where 1 is "barely useful" and 10 is "perfect." The robot uses that feedback as the reward term in the Q-learning formula, which is

$$Q[state, action] = reward[state, action] + \gamma(\max_{actions}(Q[next\ state]))$$

This formula adds the immediate reward of the action to the future reward of (state, action) pairs that might follow from it. Gamma is the factor by which future rewards are discounted compared to the immediate reward. I found that the system performed best when gamma was relatively small (0.3), which is probably a result of the unfortunate "flatness" of this learning environment (more discussion on this at the end). Finally, the robot advances to the next state and restarts the cycle. This implementation ensures that the task is completed by the end of every trial, making the robot useful right from the start. This reflects a criteria in real-life working environments that new infrastructure improvements become productive as soon as possible.

An additional benefit of this implementation is how user-friendly it is. The human partner does not need to type anything into the command line. They simply accept and reject parts, and give feedback through a graphical user interface. This meets one of my original criteria of making the human interface intuitive enough that anybody could use it without special training of their own.

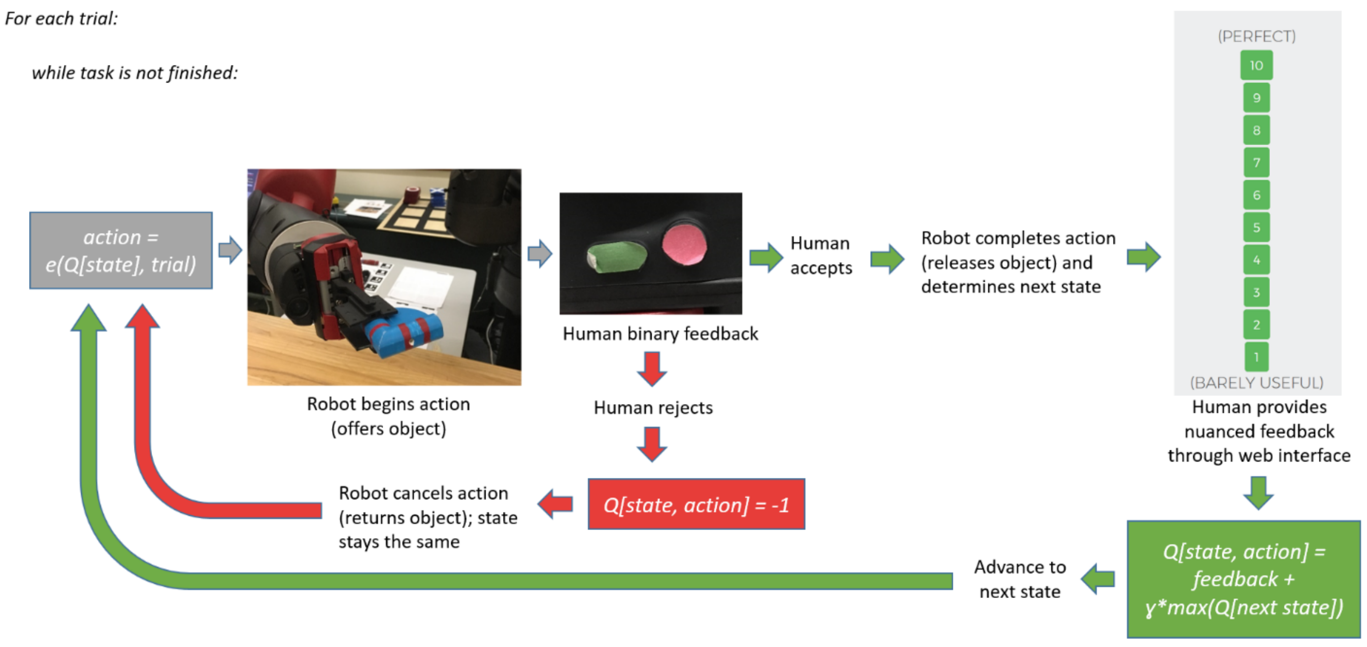

HOW HELPFUL WAS THIS ACTION?

(PERFECT)

10 9 8 7 6 5 4 3 2 1

(BARELY USEFUL)

fig 5. Web interface for nuanced, positive feedback



For each trial:
while task is not finished:
action = e(Q[state], trial)
Robot begins action (offers object)
Human binary feedback
Human accepts
Robot completes action (releases object) and determines next state
(PERFECT)
10
9
8
7
6
5
4
3
2
1
(BARELY USEFUL)
Human provides nuanced feedback through web interface
Q[state, action] = feedback + γ*max(Q[next state])
Advance to next state
Human rejects
Q[state, action] = -1
Robot cancels action (returns object); state stays the same

fig 6. Action-feedback cycle

The Multi-Armed Bandit

A central problem in all forms of machine learning is balancing the *exploitation* of actions that have a known, high payout with the *exploration* of new actions. The term "multi-armed bandit" is used to describe this class of problems. It refers to a hypothetical scenario in which a gambler, sitting in front of several slot machines with differing, unknown payouts, must decide how to play them. As he tries various machines, he must balance the exploitation of those proven to be good with exploration of those that are unknown. In my learning interactions, Baxter is repeatedly faced with a state and a set of potential actions, some of which have some demonstrated reward from previous trials and some of which have an unknown reward. This is a bandit problem, and I tested four strategies against it:

Explore-first: *If there is an unexplored action, try it. Otherwise, exploit the best.*

This strategy explores every option fully to make sure the optimal path is eventually found. However, it leads to poor performance in the short term.

Epsilon-greedy: *Explore with some small, fixed probability e. Otherwise, exploit the best option.*

This is the first of the epsilon strategies, a popular family of bandit approaches that assign or generate a probability value e = P(explore). In my task, 'explore' can mean 'pick randomly from the unexplored options' or 'pick randomly from all options' (see variations below). For epsilon-greedy I chose e = 0.2.

Epsilon-decreasing: *Set e to decrease as trials continue.*

This strategy assumes we want to explore early on and then gradually shift to mostly exploitation. It generates e with the formula e = max(0.1, (10 - trial) / 10), which sets P(explore) at 1 for the first trial, 0.9 for the second, 0.8 for the third, etc, down to 0.1 for trials 9 and beyond.

Epsilon-proportional: *Set e as inversely proportional to quality of best option.*

This option assumes we want to exploit high quality options and explore when the best choice is mediocre. It generates e with the formula e = max(0, (10 – max(options)) / 10). The tuning for this is somewhat arbitrary – I chose 0 and 10 as the key Q-values at which P(explore) is 1 and 0, respectively. Values above 10 will always be exploited. This resulted in a pretty good learning rate in this particular environment, but these parameters would need to be adjusted for other tasks and reward scales.

Within each of these strategies I tested 3 variations of ‘explore’ behavior:

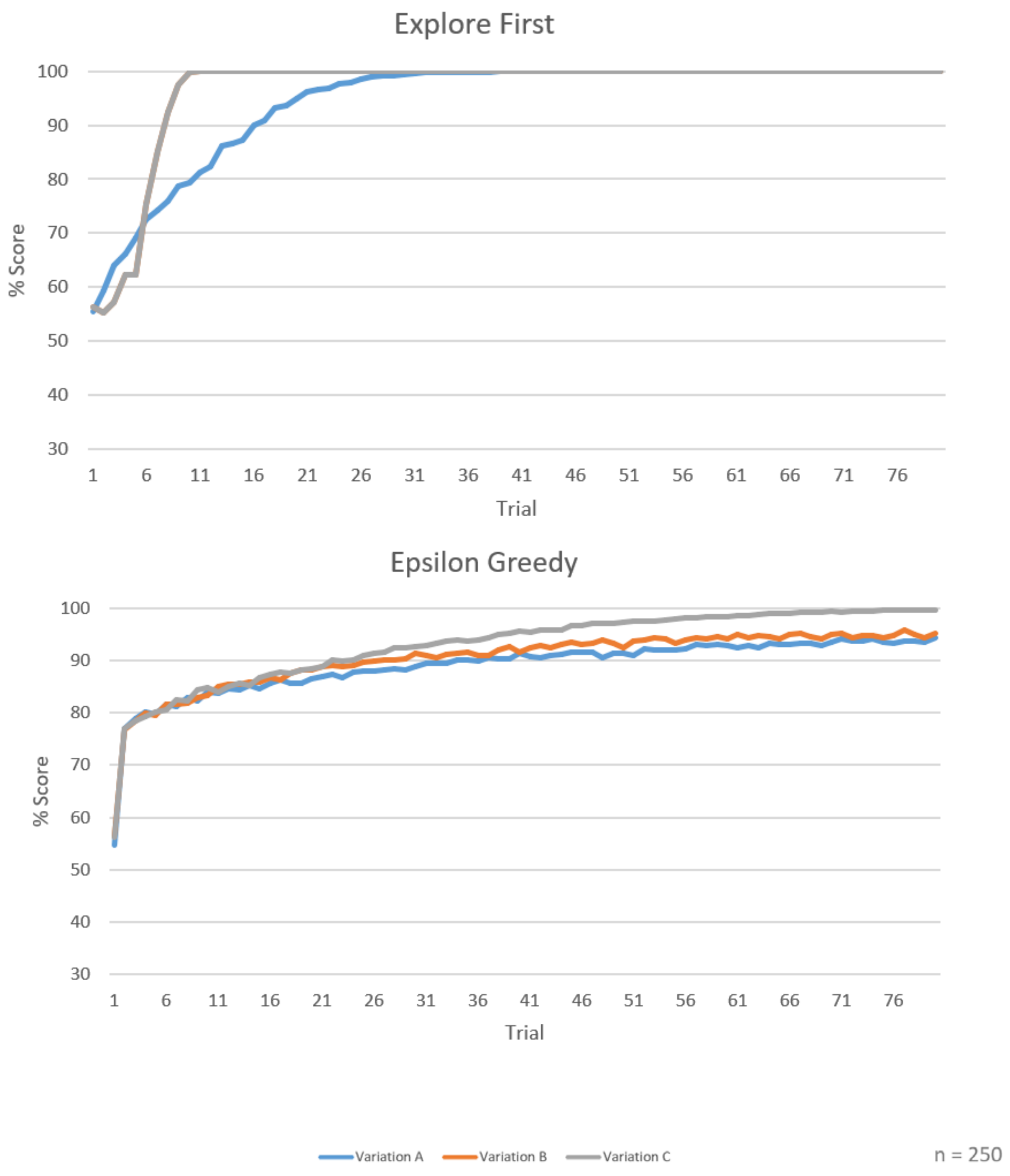
Variant A: When exploring, choose randomly from all options.

Variant B: When exploring, choose randomly from unexplored options. If none, choose randomly from all. (For ‘Explore first’, this variant does not exist)

Variant C: When exploring, choose randomly from unexplored options. If none, exploit best.

Results

For each of the three variants of each of the four strategies, I simulated 250 trials, each building on the knowledge of the last. For clarity, trials above 80 are omitted in the graphs below. I ran each of those simulations 250 times and averaged the total reward at each trial to get an accurate measurement of the improvement curve for each strategy. The results are in fig. 7. The score is displayed as the total reward for that trial divided by the maximum possible reward for an optimal trial (170). To simulate a consistent, rational human player, I created a reward matrix that the simulator checks to see if the human player accepts the action, and if so, what feedback they give. The reward matrix honors the rules of the HTM, and it also reflects a set of preferences shown in fig. 8. In each state, I gave my ideal action a score of 10, and less-preferred (but still accepted) actions diminishing scores (typically 5, 3, and 1). Rejected actions got a score of -1. For consistency, I used this set of preferences in all the simulations, and later, in my live trials, but the architecture supports any set of preferences.


Explore First
% Score
100
90
80
70
60
50
40
30
1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76
Trial
Epsilon Greedy
% Score
100
90
80
70
60
50
40
30
1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76
Trial
Variation A
Variation B
Variation C
n = 250

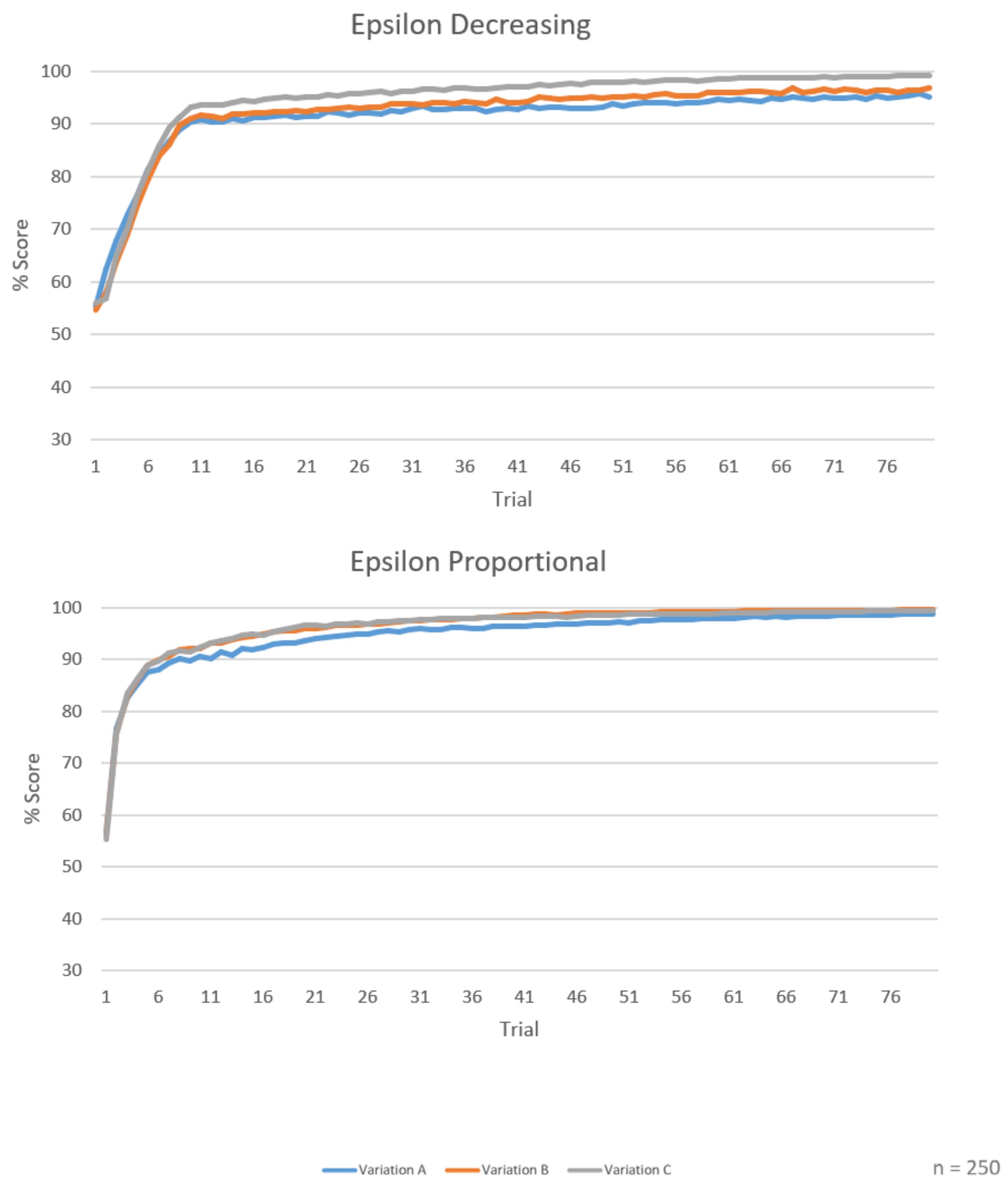


fig 7. Results of simulating each of the four exploit-explore strategies across three variations

1. I prefer to complete the back subassembly before the seat subassembly.
2. Within the back subassembly, I prefer to collect top brackets before dowels and the back panel before the long dowel.
3. Within the seat subassembly, I prefer to collect brackets before dowels and back brackets before front brackets.

fig 7. The set of preferences used in the simulated and live trials

Analyzing these results, it's clear that, other than the highly inefficient variant A for 'Explore-first', the three variants have little effect on performance. All four strategies deliver on the desired "decent first, refined later" philosophy. They quickly improve to an acceptable level of performance and then slowly converge to optimal behavior. Each strategy makes slightly different trade-offs between initial improvement speed and long-term maximum performance. 'Explore-first' grows slowly but eventually finds the perfect sequence of moves. 'Epsilon-greedy' tends to aggressively exploit the first good option it finds, so it reaches a 'shelf' early and then slowly improves from there. 'Epsilon-decreasing' and 'Epsilon-proportional' both demonstrate a slightly slower initial improvement phase but overtake 'Epsilon-greedy' in the slow convergence phase. The choice between bandit strategies depends on the application, and could even be varied depending on the current requirements of a single application. If you want your robot partner to be creative and try new approaches that might work better, you might toggle the bandit strategy to 'Epsilon-proportional.' If you want it to stick closely to the best moves it knows in a high-pressure situation, you might toggle it to 'Epsilon-greedy' with a very small or 0 epsilon.

After analyzing these simulations, I decided to use 'Epsilon-greedy' for my live trials with Baxter. Because this strategy exploits more in the first few trials than the others, it has the fastest initial rate of improvement. This early jump in quality is what I wanted to measure in person. I ran 3 consecutive live trials with e = 0.2 (see attached video). The first trial took 18 minutes and 40 seconds to complete, the second took 12 minutes, and the third only 9. In the latter two trials, Baxter made almost no errors requiring me to reject an action. However, the robot stuck closely to the acceptable path discovered in the first trial. This suggests that perhaps 'Epsilon-greedy' with e=0.2 exploits too heavily at the start, and that a better strategy might take advantage of the opportunity to explore early on.

My initial proposal stated that one deliverable would be a live demo of the robot assisting with the task intelligently after being trained through in-person trials. This result is one step better—instead of a demo of intelligent collaboration after training, I have a demo of intelligent collaboration *while* training. The robot is productive while learning. If one wanted to jump in with the robot after some prior training sessions, the robot controller exports the Q-matrix after every trial and allows for one to be loaded in on startup as a Python .pickle file. This is the exportable and importable decision-making data structure described in one of my initial goals. I have the matrices available from my live sessions in the video if one wanted to run a demo starting from where those sessions left off.

Conclusions

There are several aspects of this project that could be improved in the future. One was the unfortunate "flatness" of the task. The power of Q-learning is in how it uses past experience of future rewards (from previous trials) to weight present ones. But in this task, such backpropagation from high-value end states has little purpose, because rational humans will highly reward each state leading up to those desirable destinations. Put differently, no human trying to optimize their workflow will think, "I don't like this particular move right now, but I love the moves that stem from it. I hope my robot partner can figure it out!" As a result, the Q matrix tends to converge to a reflection of the reward matrix used in simulations. Since immediate rewards matter most, the system performs best with a small gamma that heavily discounts future rewards in the Q-value formula. This issue of flatness could be avoided with a richer task that involves some real trade-offs. For example, perhaps for each action the robot has to balance human preference with some varying external costs and benefits which the human is unconcerned with. A task like that would allow for a rich balance of short and long-term rewards.

There are also a few implementation improvements that could be made (and that I certainly would have made given a few more weeks). One would be an error correction system in the human feedback loop. Currently, if the human presses the red button, the robot will never try that action again in that particular state. This is a useful behavior because in many tasks there are truly forbidden actions, and removing them greatly narrows the search space for learning. Wasting time trying them again would not fit our fast-learning design criteria. However, if the human accidentally presses the red button during a useful action, they can permanently ruin the robot's behavior. An 'undo' feature would help. Next, as I mentioned above, I would like a more general state system that can support any combination of objects in the workspace. My current system includes just the states reachable within the HTM, but a more general system would support other sets of rules on the same set of components. The human partner could even decide in the moment which moves should be forbidden. Finally, I would like to try a user-adjustable exploit-explore parameter. This could be as simple as the value of epsilon linked to a UI slider that the human partner can control, or perhaps a selector between the various strategies available throughout each trial. This would allow the human partner to tune the robot's "adventurousness" to match the requirements of the task at hand.

In conclusion, this project demonstrates the simple power of on-line Q-learning to drive quick improvement in human-robot collaborative tasks. The "decent first, refined later" approach allows the human-robot pair to be productive right from the start while gradually refining their performance. The on-line cycle of attempted action, feedback, and correction or continuation results in a fast learning rate that doesn't require training data or many iterations. Like a human worker, the robot learns on the job. Maybe one day, when the robots take all our jobs, they will allow us to collaborate (as long as we're trained!)

Acknowledgements

Special thanks to Mark Reed, the DUS of the Electrical Engineering Department, for being a generous academic adviser for four years. Thanks to the EE and CS departments for allowing me to do a project overlapping with both. Thanks to Brian Scassellati and the Scazlab for hosting me and opening the Baxter platform for students to use. Special thanks to Olivier Mangin, Alessandro Roncone, and Corina Grigore for building the Baxter behavioral controllers that were foundational to my project. Finally, very special thanks to Corina Grigore for advising and mentoring me through the project, sharing her time and research, and helping me work through problems. I could not have done it without her.

References

[1] E. C. Grigore, O. Mangin, A. Roncone, and B. Scassellati, “Predicting supportive behaviors based on user preferences for human-robot collaboration”, in Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Extended Abstract. To Appear, Stockholm, Sweden, 2018, July 10–15.