The 2016 AAAI Fall Symposium Series:
Shared Autonomy in Research and Practice
Technical Report FS-16-05

POMDPs for Risk-Aware Autonomy

William Curran
Oregon State University
Corvallis, Oregon
curranw@oregonstate.edu

Cameron Bowie
Oregon State University
Corvallis, Oregon
bowiec@oregonstate.edu

William D. Smart
Oregon State University
Corvallis, Oregon
bill.smart@oregonstate.edu

Abstract

Although we would like our robots to have completely autonomous behavior, this is often not possible. Some parts of a task might be hard to automate, perhaps due to hard-to-interpret sensor information, or a complex environment. In this case, using shared autonomy or teleoperation is preferable to an error-prone autonomous approach. However, the question of which parts of a task to allocate to the human, and which to the robot can often be tricky. In this work, we introduce A^3P, a risk-aware task-level reinforcement learning algorithm. A^3P represents a task-level state machine as a POMDP. In this paper, we introduce A^3P, a risk-aware algorithm that discovers when to hand off subtasks to a human assistant. A^3P models the task as a Partially Observably Markov Decision Process (POMDP) and explicitly represents failures as additional state-action pairs. Based on the model, the algorithm allows the user to allocate subtasks the robot or the human in such a way as to manage the worst-case performance time for the overall task.

1 Introduction

As robots become prevalent in our homes and workplaces, it becomes important that they adapt to new environments and perform novel tasks. Autonomy allows the robot to work for an extended period of time without human intervention. The autonomous execution of many tasks are performed more efficiently when compared to manual teleoperation.

Autonomy works well in controlled environments with robot-friendly affordances. However, engineers design most devices with a human user in mind. Turning small black dials on a black stereo system is a difficult task for a robot to do autonomously. Additionally, without the user in control of the robot, the autonomous system must perform image processing, path planning, object manipulation routines, and more in order to accomplish the task. This typically requires a lot of computation, reduces battery life, and diminishes the robot's ability to parallelize physical tasks. Autonomy can also be error-prone in new environments. Developing full autonomy for all tasks in all environments is simply not possible with today's technology. Despite recent advances in the field of autonomous robotics (DARPA 2014; Muszynski, Stuckler, and Behnke 2012; Pitzer et al. 2011), we are still many years away from a fully autonomous adaptive robot.

By applying a shared autonomy approach, we can separate low-level reactivity from higher-level reasoning, and give the latter to a human operator in cases where an autonomous system might perform poorly. This gives the user more direct control over the robot, relieving many of the issues of pure autonomy (Goodrich and Schultz 2007). It also allows the robot to perform effectively in new environments without extensive reprogramming. Robots tend to be error-prone, but shared autonomy alleviates this through collaboration (Pitzer et al. 2011).

In this paper, we present As Autonomous As Possible (A^3P), an algorithm that autonomously selects from a set of approaches to use when performing parts of a given task. By combining reinforcement learning with a task-level state machine architecture, we are able to learn risks associated with each approach. This allows us to make an informed decision and better balance execution time and risk. We represent tasks as state machines, and make decisions about whether to use autonomy, shared autonomy, or teleoperation for each of the states, depending on which is most effective for the given sub-task. A^3P allows us to automatically incorporate new autonomous capabilities into the model as they emerge.

Risk aversion is also an important aspect in robotics. Many autonomous approaches fail some small percentage of the time, or take much longer to perform the task due to some changed environmental condition (i.e. lighting). These failures show up as outliers in the reward signal. Reinforcement learning uses all rewards to develop a long-term average representing the value of an action in specific state. By using a long-term average, these failure-associated reward outliers are lost. In this work, we want to find the optimal action in relation to the risk associated with that action. Therefore, we develop a reinforcement learning algorithm that represents these outliers and underlying transition functions, essentially allowing the user to choose the robot's level of risk.

To experimentally validate A^3P we use a standard navigation problem. In this problem a simulated robot must navigate through a corridor and perform a task. We add an optional shortcut through a narrow doorway with an associated error rate (Figure 2). The narrow doorway represents the

cost-benefit risk analysis, as the robot may have difficulty navigating with precision, or something may be blocking the doorway. We test the efficacy of our approach by empirically analyzing the convergence of A^3P, directly comparing the true learning solution to our learned solution, and comparing a traditionally-learned policy to our risk averse policy.

2 Background

To motivate our approach, we outline previous work performed in the field of robot autonomy, describe the reinforcement learning technique used in this work, and give an overview our hierarchical state machine infrastructure.

2.1 Robot Autonomy and Teleoperation

In robotic competitions full autonomy is typically the ultimate goal for robotic systems (DARPA 2014). With full autonomy, the user only intervenes with the robot on the tasking level. The user allocates tasks to a robot and trusts it to complete those tasks without any additional intervention. This dramatically increases the efficiency of tasks, which is valuable in many disciplines.

However, autonomy suffers in situations when computer vision fails, or when the learned task is too dissimilar to the current task. When performing manipulation tasks, the approach must use object detection or segmentation. These computer vision techniques assist the robot to find the location, shape and size information of devices. By using these device attributes and shape primitives, the robot can accurately manipulate the object. Even though there is a large research focus toward these computer vision techniques, the methods do not robustly detect objects in difficult scenes (Pitzer et al. 2011).

When autonomy fails, the user can waste more time than if they simply teleoperated the robot, or completed the task themselves, and damage can occur to the robot or the surroundings. This adds a large cost to autonomous failures. Despite this cost, the benefits of using autonomy greatly outweigh the benefits of using other approaches (Muszynski, Stuckler, and Behnke 2012), and roboticists should use them if available.

By using shared autonomy, designers can separate the low-level reactivity from the higher level reasoning, and give the higher level reasoning task to the user. Human users are skilled at classifying objects in the world and performing high-level task-based reasoning and can use these abilities to advise the robot (Goodfellow et al. 2010). By telling the robot the shape, location and size of the object to manipulate, the robot can then easily perform the task.

Shared autonomy is an effective compromise when autonomy is not practical, but it still requires the attention of the user. Shared autonomy requires an implementation for each type of question or prompt and requires a high-level decision making ability to determine which question to ask. Furthermore, the ability to understand and predict robot behavior is important, but may not be intuitive in autonomous approaches.

During teleoperation, a user controls a robot via an interface to perform a complex task (Dragan, Lee, and Srinivasa 2013). Teleoperation's predictability and robustness to environmental changes makes it the traditional approach for robotic movement. There exists a variety of interfaces for many robot types, such as ground vehicles (Fisher, McDermott, and Fagan 2009), aerial vehicles (Ollero et al. 2003), wheelchair arms (Edwards, Alqasemi, and Dubey 2006), and many more.

However, teleoperation is often tedious and difficult (Dragan, Lee, and Srinivasa 2013). The user must work through an interface that is different from actual human motion to accomplish a complex task. In addition, the robots sensors provide a limited field of view. Due to this inability to understand the robots position or perspective within the environment, it is difficult for users to navigate effectively (Bruemmer et al. 2005).

2.2 Reinforcement Learning

The Markov Decision Process (MDP) models a fully-observable sequential decision processes. A MDP is a 4-tuple (S, A, P, R), where S is a finite set of states, A is a finite set of actions, P is the probability that a certain action a leads from state s to state s' and R is the reward obtained from taking action a in state s resulting in state s'.

A POMDP is a generalized partially-observable extension of the MDP. A POMDP is a 6-tuple (S, A, O, T, γ, R), where S, A, and R are the same as a MDP, O is a set of observations, T is a set of conditional transition probabilities and γ is a set of conditional observation probabilities. In a POMDP, the agent cannot directly observe the underlying state, and must probabilistically determine its state. In this work we use a POMDP framework with reinforcement learning.

Reinforcement learning is a tool within the field of multi-agent or single-agent learning where agents take an action, observe the environment, and receive a reward based on the new environment (Sutton and Barto 1998). Reinforcement learning finds an optimal policy, $\pi(s, a) = P(a|s)$, that maximizes a reward function, R.

To develop this optimal policy, reinforcement learning uses the current state of the agent, s, the action taken, a, the resultant state, s', and a reward corresponding to the system-level success or failure of the state and action, R.

The reinforcement learner's policy represents the actions an agent should take in each state. Learning algorithms based on value functions estimate a value function $Q^\pi(s, a)$ that computes the long-term reward of state s to derive a policy $\pi(s)$ (Kalyanakrishnan and Stone 2009). In this work we use Q-Learning (Sutton and Barto 1998):

$$\begin{aligned} Q_{t+1}(s_t, a_t) \leftarrow & Q_t(s_t, a_t) + \alpha[R_{t+1}(s_t, a_t) \\ & + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)] \end{aligned}$$

with values represented in a discrete tabular structure known as the Q-Table. We also enforce ϵ-greedy action selection, where the best action is chosen with probability $1 - \epsilon$ and a random action with probability ϵ.

The reward function encompasses the high-level goal of the system. When an agent takes an action that is good for

the system, the reward function returns to the agent a positive reward proportional to how much it helped the system-level reward. Reinforcement learners also receive a correspondingly smaller reward when their actions are detrimental to the system. In this way agents can iteratively update to converge to the optimal policy according to their reward design.

Risk-sensitive reinforcement learning typically represents risk states as error states (Geibel and Wysotzki 2005), are only concerned with the variance (Borkar 2002; Liu, Goodwin, and Koenig 2003) or worst-case reward outliers (Coraluppi and Marcus 1999). In this work we deal with already learned approaches. These approaches typically have an error rate, causing large reward outliers. We represent these reward outliers as risk in the learning problem, and let the user decide their own level of risk-averseness.

2.3 Hierarchical State Machine

SMACH (Bohren 2014) is a hierarchical task-level architecture for creating complex robot behavior. It uses a hierarchical state machine infrastructure to represent the states and state transitions associated with complex tasks. A programmer can use this infrastructure to easily debug each independent state, as well as plug-and-play states for different applications.

SMACH is a Robot Operating System (ROS) (Quigley et al. 2009) package that has been used in a variety of complex tasks, mainly on the PR2 platform. Using SMACH, researchers have been able to quickly prototype tasks such as retrieving drinks from a refrigerator, autonomously plugging in (Figure 1), and playing pool.

We extend the SMACH framework to allow programmers to add multiple state transition options with the same state outcome. The A^3P infrastructure then uses reinforcement learning to decide which state transition to take.

3 As Autonomous As Possible

In this section we describe the As Autonomous As Possible (A^3P) algorithm. We first describe a general overview of A^3P and implementation details. We then discuss the learning details built into the A^3P algorithm.

3.1 A^3P Overview

In this work we introduce a risk-aware task-level reinforcement learning algorithm that can adapt an end-user's risk tolerance. A^3P learns a task-level policy where states are subtasks and actions are approaches in accomplishing that subtask. We cannot apply reinforcement learning to this type of problem, as many state-action pairs have extreme reward outliers we want to represent. For example, when a robot executes a task, teleoperation errors, object opacity, lighting conditions and more affect the efficiency of execution. These cause an extreme outlier in the reward function. In direct reinforcement learning, non-outlier rewards eventually outweigh this outlier, and it becomes lost in the average. We want the converged Q-Table to also represent these extreme outliers, rather than only the long-term averages.

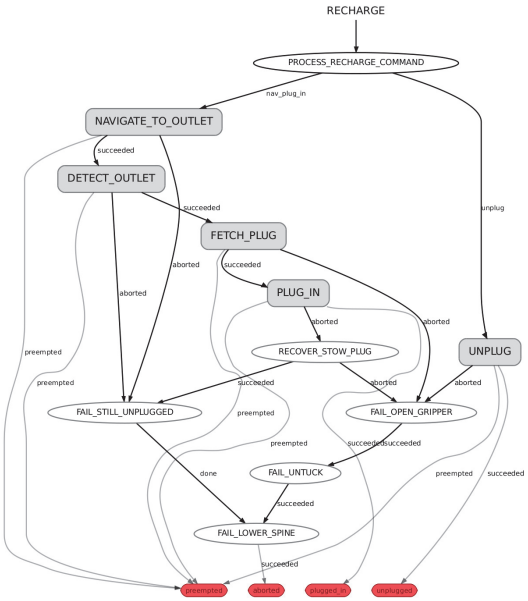

Figure 1: Example SMACH state machine used to allow the PR2 robot to charge itself.

First, we assume that all reward sampling is from a Gaussian distribution. We can make this assumption because rewards are based on time. Every task a human or computer executes can be accomplished a stochastic amount of time, and we assume these distributions can be represented by a Gaussian.

Second, we dynamically expand the Q-Table include another state-action pair. We do this by adding a mean and standard deviation to each state. We iteratively update this mean and standard deviation during learning. Whenever a new reward is received, we perform a standard outlier test. If the new reward is within 4 standard deviations of the mean, we update the associated Q-Value. If the new reward is greater than 4 standard deviations from the mean of every existing state-action pair, we create a new state-action pair with the associated Gaussian initialized to $\mathcal{N}(reward, 1)$. Dynamically adding a new state-action pair for extreme outliers allows for a representation of risk. State-action pairs associated with extreme outliers are now treated as new states with a higher associated risk.

This turns our problem into a POMDP, as we no longer know which state we are in until we receive a reward. To help alleviate the partial observability, we iteratively accumulate the state transition counts, and calculate the probability of arriving at each state. Using this information, we can calculate how risky a state-action transition is by taking into account both transition probability and the Q-Value.

Lastly, we need to be able to merge state-action values when their associated Gaussian distributions are representing the same Gaussian. We perform a standard t-test each learning iteration on all associated state-action Gaussians. If $p < 0.01$, we merge the means and standard deviations, and remove one Q-Value. At the end of the learning process, the

Q-Table is only as large as the number of true state-action pairs and reward outliers.

A^3P is also backwards-compatible with the traditional SMACH infrastructure. States which require a stochastic transition need to be reinforcement learning states, while all other states can use the built-in SMACH framework. However, the user must return a reward in the traditional SMACH states if they want those states to be represented in A^3P. We borrow from the concept of delayed rewards (Stringer, Rolls, and Taylor 2007; Watkins 1989), and use "state delayed" rewards. Whenever a state machine transitions from a learning state to a non-learning state, A^3P adds the rewards. This continues until the state machine transitions to another learning state, or a terminal state. The original learning state is then updated with the accumulated reward. This makes A^3P backwards compatible with the current SMACH framework with minimal effort from the user.

3.2 Learning Implementation

We used Q-Learning in the learning implementation of the SMACH state machine. Each state represents a sub-task, each action represents the high-level approach to finish that sub-task, and rewards are the time taken to accomplish task s with approach a. In our example, we use teleoperation, shared-autonomy or full autonomy as actions. However, additional approaches can be easily added.

Since we perform learning during the distribution modeling, we needed to make some modifications to the exploration function. A typical ϵ-greedy exploration function only explores a percentage of the time. Since we dynamically resize our Q-Table, we need to explore new states more often. Theoretically, with a fixed exploration rate, we sample each state-action pair enough times for the true reward distribution to converge, but this is computationally expensive. Instead, we can weigh the exploration rate based on the number of times each new state-action pair was sampled. This is purely a convenient optimization. In this work, we arbitrarily choose the number of samples to be 1000. In 1000 samples the mean and standard deviation would either converge to an existing state-action pair, and be merged, or diverge and be explored traditionally.

4 Experimental Validation

We experimentally validate the effectiveness of A^3P with a simple navigation problem. The act of navigating to a goal can be accomplished with many different approaches (actions). We introduce the corridor domain and then analyze the effectiveness and accuracy of A^3P.

4.1 Corridor Domain

In our corridor domain, the robot begins on one end of a U-shaped corridor, with the goal to navigate to the other end. There is a shortcut through a narrow doorway that removes the need to navigate through the corridor (Figure 2). Many other robots use this corridor and the door has a small percentage chance of being blocked. The robot must then perform some task (i.e. knob turning) at the end of navigation and then terminate.

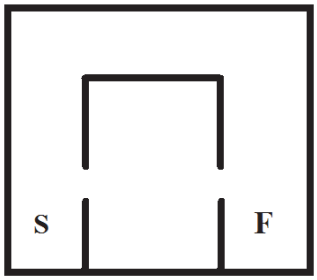

Figure 2: Corridor domain map.

We simulate the motion of a PR2 (Garage 2014) robot. This robot has a large footprint, and therefore cannot move through the small door unless it has tucked arms. However, it takes time to tuck its arms. Additionally, the narrow doorway may be difficult for autonomous approaches to move through or it may be blocked, resulting in large reward outliers. The simple and risk averse solution is to simply move through the corridor. However, this approach takes longer on average. The riskier approach is faster on average, but may take longer than the corridor approach. A^3P learns this distribution, allowing the user to make informed decisions about risk.

We built a SMACH state machine for this problem (Figure 3) with two transition paths. The first path makes the robot tuck its arms before navigating, and the second path does not. Each navigation state chooses between teleoperated control, shared autonomy or full autonomy. A^3P learns the level of risk associated with each route, and communicates to the user how long the task will take for the robot to accomplish given an amount of risk.

Each navigation approach includes multiple scenarios as described in Table 1. We estimate these scenarios with real-world occurrences, but includes an assumed probability. When using autonomy, the robot will always choose the optimal path. If the arms are tucked, the robot will choose to use the shortcut, otherwise it will use the corridor. When using shared autonomy, the robot will always choose the optimal path to where the user clicks. If the arms are tucked, the user may choose to move the robot down the corridor with some probability, or use the shortcut, otherwise the robot must use the corridor. Lastly, with teleoperation, the user may choose to use the corridor or the shortcut, with a larger probability of shortcut error if the arms are not tucked. Since the door is narrow, there is an additional state that represents ineffectively moving through the doorway. The corridor does not have this error because it is wide. At the end of navigation, the robot executes a task which can be performed with teleoperation or autonomy.

We chose a Gaussian distribution for the reward associated with each state-action using the assumption that autonomy is the most effective approach with the tightest variance, followed by shared autonomy and then teleoperation. The robot takes 120 seconds on average to navigate through the corridor, and 60 seconds through the shortcut. We scale these values based on the approach. If using teleoperation,

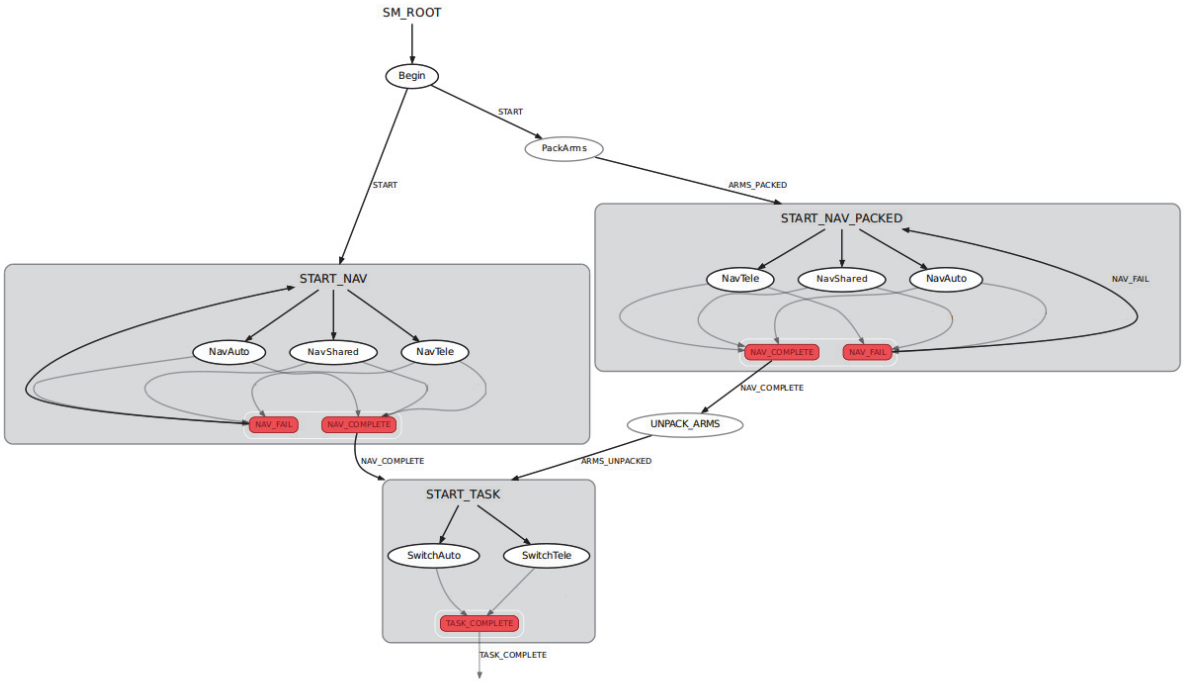

Figure 3: SMACH state machine used to navigate through the corridor domain.

State	Action	Result	P	$\mathcal{N}(\mu, \sigma)$
Packed	Auto	Shortcut	100%	(60, 2)
Packed	Auto	Shortcut Error	5%	(240, 2)
Packed	Auto	Corridor	0%	N/A
Packed	Shared	Shortcut	50%	(75, 5)
Packed	Shared	Shortcut Error	5%	(300,5)
Packed	Shared	Corridor	50%	(150, 5)
Packed	Tele	Shortcut	50%	(120, 10)
Packed	Tele	Shortcut Error	15%	(480,10)
Packed	Tele	Corridor	50%	(240, 10)
Unpacked	Auto	Shortcut	0%	N/A
Unpacked	Auto	Corridor	100%	(120, 2)
Unpacked	Shared	Shortcut	0%	N/A
Unpacked	Shared	Corridor	100%	(150, 5)
Unpacked	Tele	Shortcut	50%	(120, 10)
Unpacked	Tele	Shortcut Error	30%	(480, 10)
Unpacked	Tele	Corridor	50%	(240, 10)

Table 1: True mean, standard deviation and transition probabilities for our corridor domain.

we scale the time taken by 2, if using shared autonomy we scale by 1.25, and we do not scale autonomy. Additionally, the more impact the user has on the approach the larger the variance is. We assigned autonomous approaches a small variance of 2 seconds, shared autonomy 5 seconds, and teleoperation 10 seconds.

These probabilities and distributions have no impact on the effectiveness or analysis of A^3P, but rather the real-world application of this exact scenario. We chose an intuitively solvable domain to be able to accurately analyze A^3P with simplified data. Obtaining the real-world values of these distributions with a user study is a simple task, yet out of scope of this work. We intend to show A^3P developing a state-action pair for every element in Table 1 and learn the underlying distribution of the rewards and probabilities.

State	Action	MSE P	MSE $\mathcal{N}(\mu, \sigma)$
Packed	Auto	.015%	(0.94, 0.10)
Packed	Shared	.043%	(3.85, 0.69)
Packed	Tele	.027%	(9.73, 2.20)
Unpacked	Auto	0.00%	(0.01, 0.00)
Unpacked	Shared	0.00%	(0.06, 0.07)
Unpacked	Tele	.039%	(7.87, 1.79)

Table 2: To validity A^3P, we compare the transition probabilities and gaussians learned by A3P to the truth (Figure 1. The mean square error across each state/action pair is low and shows that A^3P learned an accurate policy.

4.2 Results

To verify the effectiveness of A^3P, we test convergence and performance in the corridor domain. We first empirically test the Q-Table convergence of A^3P by analyzing how many states A^3P dynamically created and merged. We then test performance by comparing the learned Q-Table to the Gaussians and values within Table 1. Lastly, we compare the pol-

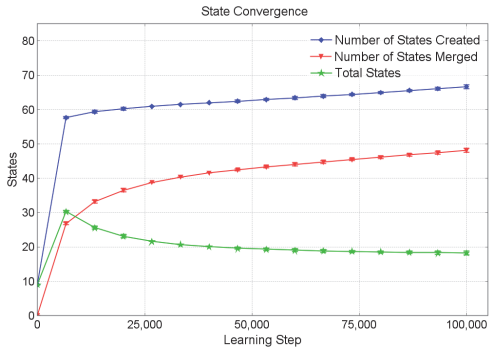

Figure 4: The A^3P Q-Table converged to the true number of states: 13 navigation (Table 1), 2 task, and 2 start states. These results were found over 10 statistical runs and error bars are included.

icy learned with traditional Q-Learning to the policy developed by A^3P.

The total number of states represented by A^3P should converge to the true distribution of states. Figure 4 demonstrates that over 10 statistical runs, the number of states converge to this true distribution. For the dynamic state creation to converge, the mean and standard deviation for each state must have also converged to a value representing the true gaussian. It is impossible to prove that every state has been represented, because there always exists a probability low enough that the value is not sampled. These points occur so rarely that they are likely not worth representing.

The mean and standard deviations representing each state/action pair should converge to the true mean and standard deviations in Table 1. Table 2 shows the mean squared error between the learned policy by A^3P and the true probability transitions and gaussian rewards. We show that each state was accurately represented and A^3P learned a policy similar to the true policy.

The policy learned by traditional Q-Learning does not take into account risk, and reward outliers are lost. To validate this claim, we apply traditional Q-Learning to the corridor domain. In this learned policy, the robot was told to tuck its arms, autonomously navigate through the shortcut, and then autonomously perform the task. Alternatively, we told A^3P to be entirely risk averse, which resulted in an optimal risk-free policy. In this policy the robot was told to autonomously navigate through the corridor without tucking its arms and then autonomously perform the task. The policy learned by A^3P resulted in a longer execution time, but without the risk (Figure 5).

5 Conclusion

The key strengths of A^3P are its simplicity, generality and risk informativeness. A^3P makes it easy to build task-level state machines that learns risk of using an autonomous or alternative approach when deciding how to perform a task. A^3P is also general enough to work in any scenario given the various tasks and approaches. Also, since A^3P takes into account reward outliers, they are no longer lost in the mean.

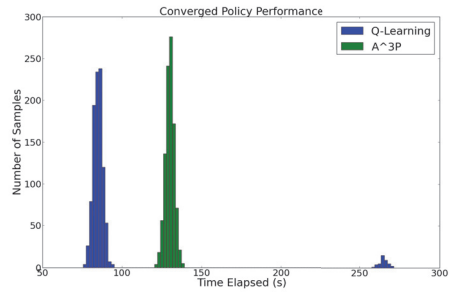

Figure 5: The policy learned by A^3P is an optimal risk-free policy. The policy learned by traditional Q-Learning results in undesirable outliers.

The user can leverage risk information when choosing how to perform a task.

During our experimental validation, we use the execution time as a reward function. A^3P is general enough to use any reward function. For example, CPU usage and network latency can be used as additional information in the reward function. A^3P learns outliers where network strength may be low, or where CPU usage is high. Suppose there is a scenario where the robot must use CPU intensive computer vision techniques while navigating, A^3P learns that teleoperation may be more efficient than autonomy due to CPU load. Or on the contrary, if network latency is high, autonomy may be more efficient than teleoperation. These scenarios, where the best task-level transitions are non-intuitive, are where A^3P is most useful. We also only used teleoperation, shared autonomy and full autonomy approaches, but any approach can be used.

Overall, A^3P is an effective approach to develop a cost-benefit analysis of executing specific approaches to accomplish many tasks. It can autonomously detect which approaches do not work in a specific environment, which approach is inefficient, or which approach works best given the current environment. Knowing this information can help roboticists be risk-aware when making important decisions regarding robot autonomy and teleoperation, and then to leverage the benefits of each approach. A^3P can be found at the Oregon State Personal Robotics Lab GitHub repository: https://github.com/OSUrobotics/.

References

Bohren, J. 2014. Smach. http://wiki.ros.org/smach.

Borkar, V. S. 2002. Q-learning for risk-sensitive control. *Mathematics of Operations Research* 27(2):294–311.

Bruemmer, D. J.; Few, D. A.; Boring, R. L.; Marble, J. L.; Walton, M. C.; and Nielsen, C. W. 2005. Shared Understanding for Collaborative Control. *IEEE Transactions on Systems, Man, and Cybernetics* 35(4):494–504.

Coraluppi, S. P., and Marcus, S. I. 1999. Risk-sensitive and Minimax Control of Discrete-Time, Finite-State Markov Decision Processes. *Automatica* 35(2):301 – 309.

DARPA. 2014. DARPA Grand Challenge. http://archive.darpa.mil/grandchallenge/index.html.

Dragan, A.; Lee, K.; and Srinivasa, S. 2013. Teleoperation with Intelligent and Customizable Interfaces. *Journal of Human-Robot Interaction* 1(3).

Edwards, K.; Alqasemi, R.; and Dubey, R. 2006. Wheelchair-Mounted Robotic Arms: Design and Development. In *The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics 2006*, 613–618.

Fisher, A.; McDermott, P.; and Fagan, S. 2009. Bandwidth Allocation in a Military Teleoperation Task. In *4th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 287–288.

Garage, W. 2014. PR2 Robot. https://www.willowgarage.com/pages/pr2/overview.

Geibel, P., and Wysotzki, F. 2005. Risk-sensitive Reinforcement Learning Applied to Control Under Constraints. *Journal of Artificial Intelligence Research* 24(1):81–108.

Goodfellow, I. J.; Koenig, N.; Muja, M.; Pantofaru, C.; Sorokin, A.; and Takayama, L. 2010. Help Me Help You: Interfaces For Personal Robots. In *5th ACM/IEEE International Conference on Human-Robot Interaction*, 187–188.

Goodrich, M. A., and Schultz, A. C. 2007. Human-robot Interaction: A Survey. *Found. Trends Hum.-Comput. Interact.* 1(3):203–275.

Kalyanakrishnan, S., and Stone, P. 2009. An Empirical Analysis of Value Function-based and Policy Search Reinforcement Learning. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, 749–756.

Liu, Y.; Goodwin, R.; and Koenig, S. 2003. Risk-averse Auction Agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 353–360.

Muszynski, S.; Stuckler, J.; and Behnke, S. 2012. Adjustable Autonomy for Mobile Teleoperation of Personal Service Robots. In *The 21st IEEE International Symposium on Robot and Human Interactive Communication*, 933–940.

Ollero, A.; Alcazar, J.; Cuesta, F.; Lopez-Pichaco, F.; and Nogales, C. 2003. Helicopter Teleoperation for Aerial Monitoring in the Comets Multi-Uav System. In *3th IARP Workshop on Service, Assistive, and Personal Robots*, 137–142.

Pitzer, B.; Styer, M.; Bersch, C.; DuHadway, C.; and Becker, J. 2011. Towards Perceptual Shared Autonomy for Robotic Mobile Manipulation. In *2011 IEEE International Conference on Robotics and Automation*, 6245–6251.

Quigley, M.; Conley, K.; Gerkey, B. P.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: An Open-Source Robot Operating System. In *ICRA Workshop on Open Source Software*.

Stringer, S. M.; Rolls, E. T.; and Taylor, P. 2007. Learning Movement Sequences With a Delayed Reward Signal in a Hierarchical Model of Motor Function. *Neural Networks* 20(2):172–181.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning I: Introduction*.

Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, King's College, Cambridge, UK.