frontiers in Robotics and AI

ORIGINAL RESEARCH
published: 26 June 2018
doi: 10.3389/frobt.2018.00079

Reactive Reinforcement Learning in Asynchronous Environments

Jaden B. Travnik[1,2], *Kory W. Mathewson*[1,2], *Richard S. Sutton*[1] *and Patrick M. Pilarski*[1,2]*

[1] *Reinforcement Learning and Artificial Intelligence Laboratory, Department of Computing Science, University of Alberta, Edmonton, AB, Canada,* [2] *Bionic Limbs for Improved Natural Control Laboratory, Department of Medicine, University of Alberta, Edmonton, AB, Canada*

The relationship between a reinforcement learning (RL) agent and an asynchronous environment is often ignored. Frequently used models of the interaction between an agent and its environment, such as Markov Decision Processes (MDP) or Semi-Markov Decision Processes (SMDP), do not capture the fact that, in an asynchronous environment, the state of the environment may change during computation performed by the agent. In an asynchronous environment, minimizing reaction time—the time it takes for an agent to react to an observation—also minimizes the time in which the state of the environment may change following observation. In many environments, the reaction time of an agent directly impacts task performance by permitting the environment to transition into either an undesirable terminal state or a state where performing the chosen action is inappropriate. We propose a class of *reactive reinforcement learning algorithms* that address this problem of asynchronous environments by immediately acting after observing new state information. We compare a reactive SARSA learning algorithm with the conventional SARSA learning algorithm on two asynchronous robotic tasks (emergency stopping and impact prevention), and show that the reactive RL algorithm reduces the reaction time of the agent by approximately the duration of the algorithm's learning update. This new class of reactive algorithms may facilitate safer control and faster decision making without any change to standard learning guarantees.

Keywords: reinforcement learning, asynchronous environments, resource-limited systems, reaction time, real-time machine learning

OPEN ACCESS

Edited by:
Joschka Boedecker,
Albert Ludwigs Universität Freiburg, Germany

Reviewed by:
Heiko Hamann,
Universität zu Lübeck, Germany
Daniel Alexander Braun,
Max-Planck-Institut für biologische Kybernetik, Germany

****Correspondence:***
Patrick M. Pilarski
pilarski@ualberta.ca

Specialty section:
This article was submitted to Computational Intelligence, a section of the journal Frontiers in Robotics and AI

Received: *31 January 2018*
Accepted: *08 June 2018*
Published: *26 June 2018*

Citation:
Travnik JB, Mathewson KW, Sutton RS and Pilarski PM (2018) Reactive Reinforcement Learning in Asynchronous Environments. Front. Robot. AI 5:79. doi: 10.3389/frobt.2018.00079

1. INTRODUCTION

Reinforcement learning (RL) algorithms for solving optimal control problems are comprised of four distinct components: acting, observing, choosing an action, and learning. This ordering of components forms a protocol which is used in a variety of applications. Many of these applications can be described as synchronous environments where the state of the environment remains in the same state until the agent acts at which point the environment immediately returns its new state. In these synchronous environments, such as Backgammon (Tesauro, 1994) or classic control problems, it is not necessary to know the computation time to perform any of the protocol's components. For this reason, most reinforcement learning software libraries, such as RL-Glue (Tanner and White, 2009), BURLAP[1] or OpenAI gym[2], have functions which accept the agent's action, and return the new state and reward immediately. These functions remain convenient for simulated environments where the dynamics of the environment can be computed easily (Sutton and Barto, 1998). However, unlike synchronous environments, asynchronous environments, also referred to as dynamic environments, do not wait for an agent to select an action before they change state (Kober et al., 2013; Pilarski et al., 2015; Russell and Norvig, 2016). The computation of RL protocol components (acting, observing, choosing an action, learning) takes time and an asynchronous environment will continually change state during this time (Degris and Modayil, 2012; Hester et al., 2012; Caarls and Schuitema, 2016). This can negatively affect the performance of the agent. If the agent's reaction time is too long, its chosen action may become inappropriate in the now changed environment. Alternatively, the environment may have moved into an undesirable terminal state.

In this paper, we explore a very simple alternative arrangement of the reinforcement learning protocol components. We first investigate a way to reorder SARSA control algorithms so that they are able to react to the most recent observation before learning about the previous time step; we then discuss convergence guarantees of these reordered approaches when viewed in discrete time (following Singh et al., 2000). Then, we examine a asynchronous continuous-time robot task where the reaction times of agents affect the overall task performance—in this case, breaking or not breaking an egg with a fast-moving robotic arm. Finally, we present a discussion on the implementation of reactive algorithms and their application in related settings.

1.1. Related Background

The focus of most contemporary RL research is on action selection, representation of state, and the learning update itself; the performance impact of reaction time is considered less frequently, but is no less important of a concern (Barto et al., 1995). Several groups have discussed the importance of minimizing reaction time (Degris and Modayil, 2012; Hester et al., 2012; Caarls and Schuitema, 2016). Hester et al. noted that existing model-based reinforcement learning methods may take too much time between successive actions and presented a parallel architecture that outperformed traditional methods. Caarls and Schuitema extended this parallel architecture to the online learning of a system's dynamics (Caarls and Schuitema, 2016). Their learned model allowed for the generation of simulated experience which could be combined with real experience in batch updates. While parallelization methods may improve performance, they are computationally demanding. We propose an alternative approach when system resources are constrained.

[1]http://burlap.cs.brown.edu/

[2]https://gym.openai.com/

2. TEMPORAL DELAYS IN ASYNCHRONOUS ENVIRONMENTS

Temporal-difference (TD) control algorithms like SARSA and Q-Learning (Watkins and Dayan, 1992; Rummery et al., 1994; Sutton and Barto, 1998) were introduced with synchronous discrete-time environments in mind; these environments are characterized by remaining stationary during the planning and learning of the agent. In synchronous environments, the time to perform the individual components of the SARSA algorithm protocol has no impact on task performance. Specifically, the time it takes to react to a new game state in chess has no influence over the end of the game. In asynchronous environments however, the time it takes for the agent to react to new observations can drastically influence its performance on the task. Such as in the original formulation of a cart-pole, the agent applied it's actions *left* and *right* at discrete time intervals (Barto et al., 1983). These time intervals were set small enough so that the pole would not fall further than the agent would be able to recover.

As a concrete example, imagine an asynchronous environment we here call Hallway World with a left turn leading to the terminal state, as shown in **Figure 1**. The agent starts an episode near the bottom of a hallway and has two actions: *move left* and *move up* which move the agent in a direction and continue to move the agent in that direction

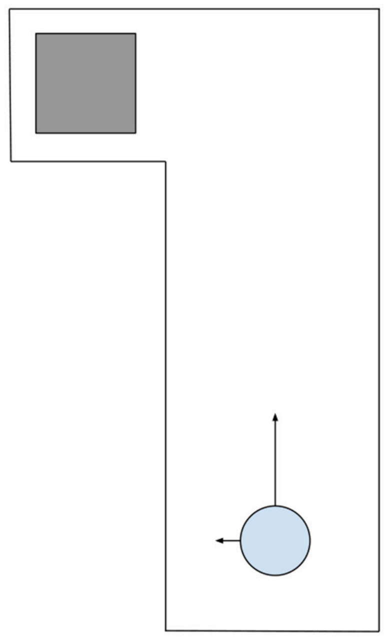

FIGURE 1 | The Hallway World task with the agent (the blue circle) starting near the bottom of the hallway. The gray square denotes the terminal state. The arrows pointing away from the circle denote the 2 actions which move the agent leftwards or upwards and continue moving the agent in that direction until interrupted.

with constant velocity until interrupted with the other action, hitting a wall, or arriving in the terminal state. If the agent hits a wall it receives a reward of -1 and comes to a stop. When the agent reaches the terminal state it will receive a negative reward directly proportional to the duration of the episode. In this way, the agent is motivated to get to the terminal state as quickly as possible without touching the walls. The only observation that the agent can make is to determine if there is a wall on its left.

The optimal policy in Hallway World is for the agent to *move upward and observe the wall continually until an opening in the wall is observed then immediately move leftwards toward the terminal state*. SARSA (Algorithm 1) applied using on-line learning is unable to learn this optimal policy because it is restricted by the delay between observing the opening in the wall and moving toward the terminal state. Specifically, when an agent, that had previously learned to take the *left* action using SARSA, observes the opening in the wall it would choose the *left* action but it would not be able to take this action until it had spent time learning about the previous action and observation during the learning step (**Figure 2**, top row). Assuming that the components of the algorithm (acting, observing, choosing an action, and learning) each take some constant amount of time t_c, if a SARSA agent observes an opening in the wall, it must choose to move left and learn about the previous state-actions before taking the action, this would add $2t_c$ onto episode time, thereby affecting the total reward and task performance. Thus, overall performance in Hallway World decreases with the time the agent spends selecting an action and learning, irrespective of how these components are performed.

Algorithm 1 |SARSA: An on-policy TD control algorithm

```
Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$
        Observe $R, S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S', A \leftarrow A'$
```

Algorithm 2 |Reactive SARSA: A *reactionary* on-policy TD algorithm

```
Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Take action $A$
    Repeat (for each step of episode):
        Observe $R, S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S', A \leftarrow A'$
```

3. REACTIVE SARSA

To minimize the time between observing a state and acting upon it, we propose a modification to conventional TD-control algorithms: *take actions immediately after choosing them given the most recent observation*. We propose a straightforward new algorithm, Reactive SARSA, as one example of this modification (Algorithm 2); in each step of the learning loop, the agent observes a reward and new state, chooses an action from a policy based on the new state, immediately takes that action, then performs the learning update based on the previous action. Illustrating how Reactive SARSA responds in Hallway World, when an agent that had previously learned to take the *left* action using Reactive SARSA observes the opening in the wall, it would choose the *left* action and immediately take this action. This immediate response would reduce the time it takes for the agent to reach the terminal state compared to an agent using the SARSA algorithm (see **Figure 2**).

The trade-off induced by this re-ordering is in how quickly an agent can see the results of its actions. In SARSA, where observation follows taking an action, if the consequences of an action can be readily observed, the agent can see these consequences. An agent using Reactive SARSA would be unable to see these consequences immediately as it must perform its learning update.

The slight reordering of RL algorithm protocol components does not effect convergence in discrete time. Here, we provide a basic theoretical sketch that, in discrete-time synchronous tasks, Reactive SARSA learns the same optimal policy as SARSA, in the same manner. As is illustrated in **Figure 3**, this equivalence is trivially evident by observing that in both algorithms the first 2 actions are selected using the initial policy. In each subsequent step t, actions are chosen using the policy learned on the last step, and the policy updates happen with identical experiences (**Figure 3**).

If we redefine Hallway World as a synchronous environment where the agent moves a constant distance for each action instead of continually moving, the same policies and performance would be expected between both algorithms and this is what we found in practice. The difference between reactive and non-reactive algorithms is the order of the RL components (acting, observing, choosing an action, and learning).

4. EXPERIMENTS

To explore the differences between the SARSA and Reactive SARSA learning algorithms in asynchronous environments, we designed a reaction-time-dependent task with similar qualities to the Hallway World described above and illustrated in **Figure 1**.

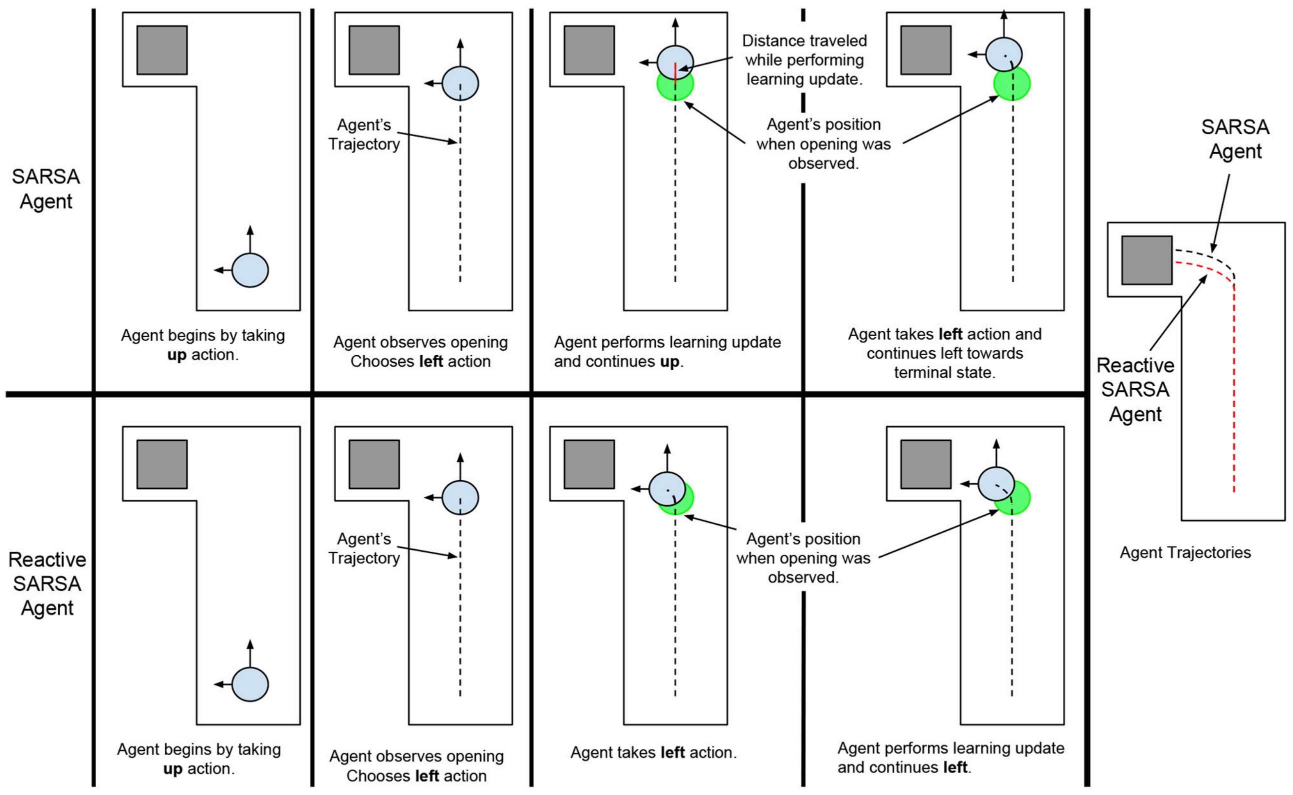
SARSA Agent
Agent begins by taking **up** action.
Agent's Trajectory
Agent observes opening Chooses **left** action
Distance traveled while performing learning update.
Agent's position when opening was observed.
Agent performs learning update and continues **up**.
Agent takes **left** action and continues left towards terminal state.
Reactive SARSA Agent
Agent begins by taking **up** action.
Agent's Trajectory
Agent observes opening Chooses **left** action
Agent takes **left** action.
Agent's position when opening was observed.
Agent performs learning update and continues **left**.
SARSA Agent
Reactive SARSA Agent
Agent Trajectories

FIGURE 2 | The behaviors of a SARSA agent **(Top)** and a Reactive SARSA agent **(Bottom)** on an Asynchronous Hallway World. Overlapping the trajectories of the agents illustrates that the Reactive SARSA agent produces a shorter trajectory and gets to the terminal state before the SARSA agent.

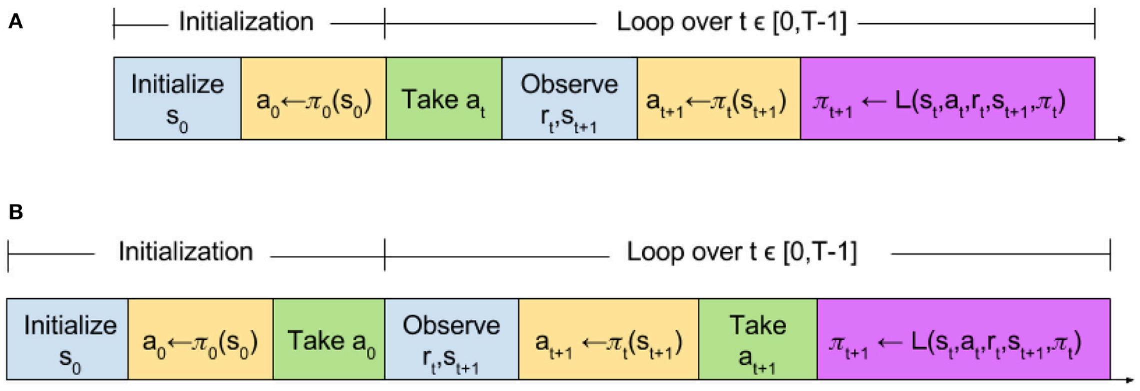
A
Initialization
Loop over t ϵ [0,T-1]
Initialize s_0
$a_0 \leftarrow \pi_0(s_0)$
Take a_t
Observe r_t, s_{t+1}
$a_{t+1} \leftarrow \pi_t(s_{t+1})$
$\pi_{t+1} \leftarrow L(s_t, a_t, r_t, s_{t+1}, \pi_t)$
B
Initialization
Loop over t ϵ [0,T-1]
Initialize s_0
$a_0 \leftarrow \pi_0(s_0)$
Take a_0
Observe r_t, s_{t+1}
$a_{t+1} \leftarrow \pi_t(s_{t+1})$
Take a_{t+1}
$\pi_{t+1} \leftarrow L(s_t, a_t, r_t, s_{t+1}, \pi_t)$

FIGURE 3 | Time step comparison of **(A)** standard and **(B)** reactive reinforcement learning algorithms. The function L refers to a learning function which updates the policy π. The learning function L is not limited to the SARSA learning update and encompasses any learning update such as Q-learning.

The task was performed using one joint of a robotic arm (an open-source robotic arm, Dawson et al., 2014 shown in **Figure 4**). We conducted two experiments with the same episodic stopping task. The arm started at one extreme of the joint rotation range and was then rotated quickly toward the other end of its range. The agent needed to stop the rotation as soon as possible following an indication to stop that was observed by a state change from "Normal" to "Emergency."

The agent had two actions: *stop* and *move*. If the agent chose to *stop* while in the "Normal" state, the agent would receive a constant reward of -1, remain in the "Normal" state, and the arm would continue rotating. If the agent chose to *move* while in the "Normal" state, the agent would receive a reward of 0, remain in the "Normal" state, and continue rotating. Once the "Emergency" state had been observed, the reward for either action would be a negative reward proportional to the amount of time (in μs)

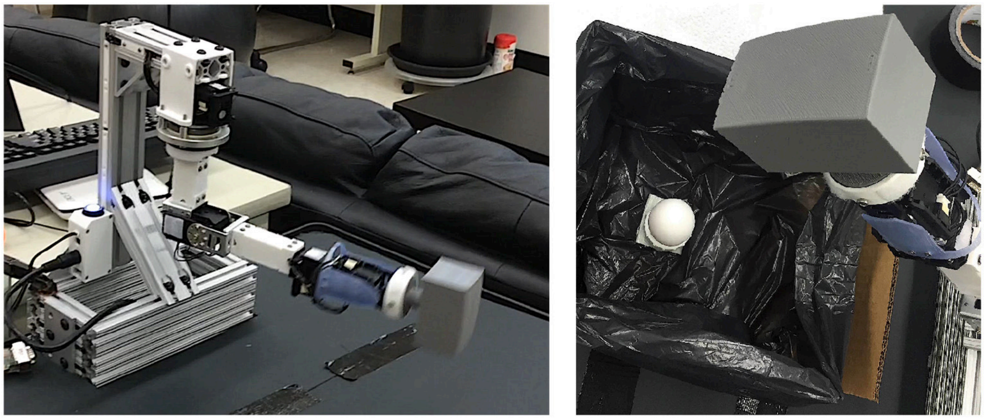

FIGURE 4 | Experimental setup, showing the robot arm in motion for the first experiment **(Left)** and the robot arm poised to impact an egg during the second experiment **(Right)**.

spent in the "Emergency" state. When the agent chose to *stop* in the "Emergency" state, it transitioned to the terminal state, thereby ending the episode. This reward definition was chosen as a convenient means of valuing reaction time; the distance traveled during the reaction time would also have been a valid alternative.

If complete information about the stopping task was available, optimal performance could be obtained through direct engineering of a control system designed to stop the arm as soon as the state changes. However, the agent did not know which state was the "Emergency" state and used its experience to learn what to do in any given state. By excluding the complex state information of many real-world robotic tasks and using this simple stopping task, we were able to investigate the effects of reaction time on overall performance and the differences between conventional and reactive TD-control algorithms.

4.1. Experiment 1

To explore the effect of the reactive algorithms on reaction time and task performance, the robotic arm was programmed to move at a constant velocity along a simple trajectory (**Figure 4**, left). The experiment involved 30 trials, each of which was comprised of 20 episodes with the agent starting in a "Normal" state and switching to the "Emergency" state after some random amount of time. The standard and Reactive SARSA agents were compared with greedy policies, $\gamma = 0.9$, $\lambda = 0.9$, and $\alpha = 0.1$.

There are many potential causes for time delays in the learning step. One example maybe waiting for a path planning algorithm, like Hybrid A*, to complete (Wei et al., 2017). Another example comes from the idea of predictive knowledge representations. Here knowledge is represented and learned as a collection of predictions about a robots observed experience. Such knowledge may be updated and computed during each cycle. One approach to building this knowledge is the Horde architecture. Horde introduces the idea of *demons* which learn predictions about the environment and can build on each other to achieve a scalable method of knowledge learning (Sutton et al., 2011). A Horde architecture with 2,576 demons (predicting the position, velocity, temperature, load and other measures) was experimentally validated on the robotic arm. On the experimental setting testing this setup, average computation time of one demon's prediction took 3.33 μs. The more predictions one wants to make, the longer the duration of learning, thus the reaction time increases. Specifically, the time delays of 50, 100, 250, and 500 ms on the experimental hardware are equivalent to a horde architecture of approximately 15,000, 45,000, 75,000, and 150,000 demons, respectively. It is clear that more predictions increase the reaction-time, and the addition of time delays in the following experiments was used to appropriately simulate the addition of more predictions. To simulate the performance of these additional predictions and modulate in a controlled fashion the effect of longer learning steps, these time delays were added to the learning update step.

Figure 5 shows how the the duration of learning influenced the task performance. The figure shows the average cumulative episodic return for the last 10 episodes, once both agents had learned policies. As the delay increased, both algorithms suffered performance decreases, but the standard SARSA algorithm performed worse with larger variability. While Reactive SARSA was affected by increasing time delays, the impact was less severe. Specifically, median reaction time of the Reactive SARSA was approximately half of the added learning delay. This effect is most likely because the transition from "Normal" to "Emergency" state occurred at a uniformly random selected time and since the majority of the duration of a step consists in learning, the state change occurred on average halfway through the learning step. This also accounts for the increasing reaction time in standard SARSA, as it must wait a full additional time step before reacting to the "Emergency" state.

4.2. Experiment 2

The second experiment considered a human-robot interaction task which demanded cooperation between a human and robotic arm to not crush an egg (**Figure 4**, right). The robot arm was

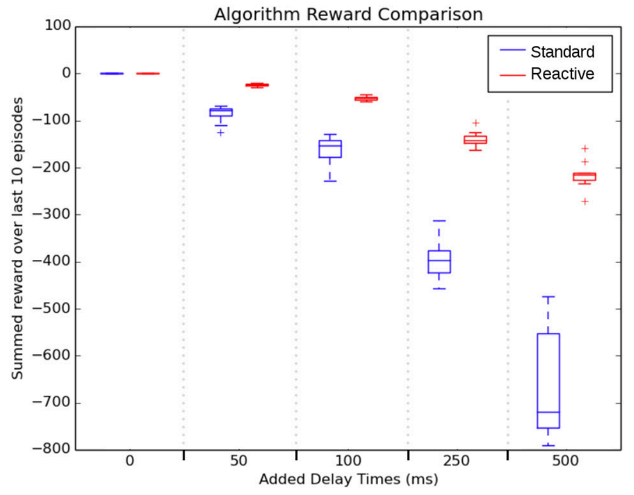

FIGURE 5 | Key result: a comparison of summed reward over the last 10 episodes of 30 trials across 5 different learning delay length lengths during robot arm motion. Reactive SARSA had a significantly reduced reaction time when compared to the standard SARSA algorithm for all delay lengths.

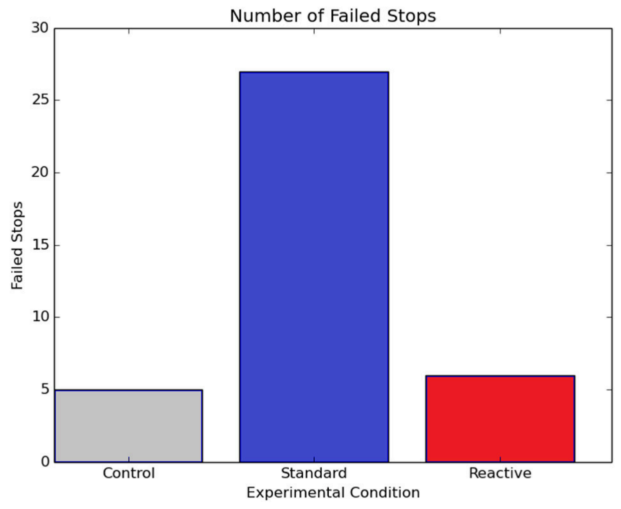

FIGURE 6 | The total number of failed stops for each algorithm during the robot's acceleration toward a breakable object (Experiment 2), summed over all four participants. For all subjects, Reactive SARSA had far fewer failed stops than standard SARSA.

positioned above a target, in this task an egg, and would move at a constant velocity toward the target. The human was told to press a button to stop the arm before crushing the egg, and to try to stop it as close to the egg as possible without touching it[3]. The learning task for the RL agent was to learn to stop as soon as the participant pressed a button. For the first 10 episodes of a trial, the participant trained using a hard-wired stopping algorithm which automatically stopped the arm when the participant pressed a button. For the remaining 40 episodes of the trial, previously learned SARSA and Reactive SARSA agents were used, each algorithm was used for 20 episodes, and the algorithm used was randomly alternated on each episode. The state changed from "Normal" to "Emergency" when the participant pressed a button. All three algorithmic conditions: (1) control, (2) SARSA, and (3) Reactive SARSA, included a constant 50 ms delay to simulate a longer learning step (e.g., the time it would take to update the predictions for 15,000 demons). The algorithm used on a trial was hidden from the participant. Four individuals participated in the experiment, providing a total of 80 episodes of each algorithm. All participants provided informed consent as per the University's Ethics Review Board and could voluntarily end the experiment at any time if they wished.

Figure 6 shows the total of all failed stops ("broken eggs") for each algorithmic condition as summed across all four participants. Reactive SARSA had approximately the same number of failed stops as the optimal control strategy whereas the standard SARSA performed significantly worse with more than four times as many failed stops.

In addition to comparing the number of failed stops, and thus crushed eggs, of each algorithmic condition, the time between the state change from the button push of the participant reaction time of the agent was recorded and is presented in **Figure 7**. The effects

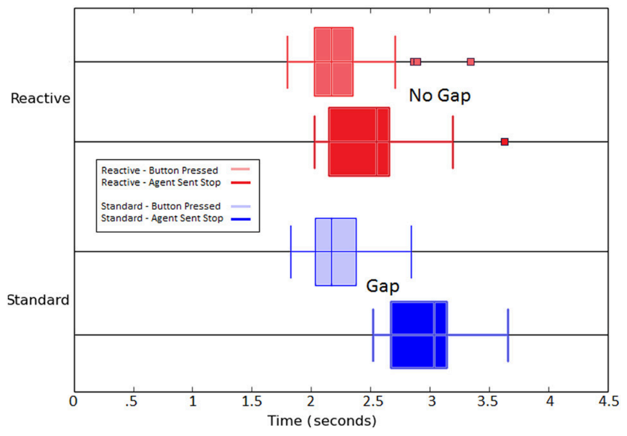

FIGURE 7 | Boxplot comparison of the distributions of events over all episodes between Reactive SARSA and the standard SARSA algorithm. Zero on the x-axis is the the moment the arm begins moving. The overlap of the button press and reactive agent's action indicates that the reactive agent has negligible delay in its reaction to the participant's input (seen in the overlap between light red and dark red, top). The standard agent's ability to act is delayed by the length of learning (visible gap between light blue and dark blue, bottom).

of longer learning on reaction time is evident in this figure as the standard SARSA algorithm agent's Stop action is trailing behind the button press by approximately the length of the learning update and delay; this is contrasted by the tight overlap of the stimulus and action for the Reactive SARSA agent.

5. DISCUSSION

Our results indicate that rearranging the fundamental components of existing TD-control algorithms (act, observe,

[3] All damaged eggs were eaten.

choose action, learn) has a beneficial effect on performance in asynchronous environments where task performance is reaction-time dependent. A reactive agent can perform better in these environments as it can act immediately following observations. As would be expected, this effect becomes especially prominent as the duration of learning operations increases. Although the current experimental design added a simulated delay to the learning update step, our results indicate that as the time between observing and acting grows, performance in these environments deteriorates, regardless of the source of these delays. As standard RL algorithms perform learning and state representation construction [e.g., tile-coding, (Sutton and Barto, 1998), deep neural networks, (Silver et al., 2016), etc.] between observing and acting, additional computation time is necessary. In asynchronous environments, as these steps become longer, the order of algorithmic components [acting, observing, choosing an action, and learning] becomes more critical. As we have shown, performance in asynchronous environments is inversely proportional to the total length of time between observations and acting.

One alternate means of addressing delay-induced performance concerns may be to create a dedicated thread for each of the RL algorithm components (c.f., Caarls and Schuitema, 2016). We believe this is a promising area for continued research. As suggested, reactive algorithms in this work may have great utility when applied to single-thread computers as they do not require multiple threads so while the order of algorithmic components might seem at first like a minor implementation detail, it may prove critical when applied to these systems.

Put more strongly, we believe that allowing an RL agent to learn an optimal ordering of its learning protocol or to interrupt learning components for more pressing computations are interesting subjects of future work. As a thought experiment, imagine an oracle-agent that has perfectly modeled its environment, knowing the outcome of every possible action. If this environment is asynchronous and provides more positive rewards for completing a task as quickly as possible, then, in order for this oracle-agent to maximize its reward, it should eliminate all computations which are not necessary as they delay the agent. Since it has perfectly modeled its environment, learning does not and will not improve its model. Moreover, if by predicting the state using its perfect model, the agent can achieve a perfect state prediction without observing, observation is also an unnecessary computation. Thus the oracle-agent can eliminate learning and observing and can simply act.

Similar to the oracle-agent, some human experts, such as video-game speed runners and musicians, are sometimes able to perform their talents without actually observing the consequences of their actions because they have memorized a long sequence of optimal actions and can act out this sequence without needing to observe its results (Mallow et al., 2015; Talamini et al., 2017). By viewing the order of algorithmic components of learning algorithms as modifiable, an agent is freed to be able to find an optimal ordering of its learning protocol which may allow it to interrupt long lasting computations (e.g., analyzing an image) for more pressing computations (e.g., avoiding a pedestrian).

The ability for an artificial agent to re-order its computational components could prove to be better utilized in the case of multi-threaded systems. Computationally limited systems that have the capacity for operating systems that support multithreading, such as TinyOS, have the ability to be more expressive in the way they structure their computation (Levis et al., 2005). Artificial agents taking advantage of the techniques found in multithreaded systems could schedule their own learning update via tasks, invoke hardware interrupts, and delay expensive learning computations dynamically.

6. CONCLUSIONS

RL algorithms are built on four main components: acting, observing, choosing an action, and learning. The execution of any of these components takes time, and while this may not affect synchronous discrete-time environments, it is a critical consideration for asynchronous environments, especially when task performance is proportional to the reaction time of the agent. *An agent should never have to wait to take an action after receiving up-to-date observations*. In this paper we present a novel reordering of the conventional RL algorithm which allows for faster reaction times. We present a simple sketch for algorithmic equivalence in synchronous discrete-time settings and show improved performance in an asynchronous continuous-time stopping task which is directly linked to agent reaction time. These results indicate that (1) reaction time is an important consideration in asynchronous environments, (2) the choice of when in a loop the RL agent should act affects an agent's reaction time, (3) reordering of the components of the algorithm as suggested here will not affect an agent's performance in synchronous discrete-time environments, (4) reactive algorithms reduce the reaction time, and thus improve performance, potentially also decreasing the time it takes for an agent to learn an optimal policy. This work, therefore, has wide potential application in real-world settings where decision making systems must swiftly respond to new stimuli.

ETHICS STATEMENT

This study was carried out in accordance with the recommendations of University of Alberta Human Ethics Review Board. The protocol was approved by the University of Alberta Human Ethics Review Board. All subjects gave written informed consent in accordance with the Declaration of Helsinki.

AUTHOR CONTRIBUTIONS

Experimentation, primary drafting of the manuscript, and data analysis by JT. Algorithm and concept design by JT, KM, RS, and PP. Study design by JT and PP. Study supervision by PP. All authors were responsible for writing, editing, and approving the manuscript.

FUNDING

This research was undertaken, in part, thanks to funding from the Canada Research Chairs program, the Canada Foundation for Innovation, the Alberta Machine Intelligence Institute, Alberta Innovates—Technology Futures, DeepMind, and the Natural Sciences and Engineering Research Council.

ACKNOWLEDGMENTS

The authors thank the other members of the Bionic Limbs for Natural Improved Control Laboratory and the Reinforcement Learning and Artificial Intelligence Laboratory for many helpful thoughts and comments.

REFERENCES

Barto, A. G., Bradtke, S. J., and Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artif. Intell.* 72, 81–138. doi: 10.1016/0004-3702(94)00011-O

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Sys. Man Cybern.* 5, 834–846. doi: 10.1109/TSMC.1983.6313077

Caarls, W., and Schuitema, E., (2016). Parallel online temporal difference learning for motor control. *IEEE Trans. Neural Netw. Learn. Syst.* 27, 1457–1468. doi: 10.1109/TNNLS.2015.2442233

Dawson, M. R., Sherstan, C. , Carey, J. P., Hebert, J. S., Pilarski, P. M. (2014). "Development of the Bento Arm: An improved robotic arm for myoelectric training and research," in *Proceedings of Myoelectric Controls Symposium (MEC)* (Fredericton, NB), 60–64.

Degris, T., and Modayil, J. (2012). "Scaling-up knowledge for a cognizant robot," in *Notes AAAI Spring Symposium Series* (Palo Alto, CA).

Hester, T., Quinlan, M., and Stone, P. (2012). "RTMBA: a real-time model-based reinforcement learning architecture for robot control," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (St. Paul, MN).

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: a survey. *Int. J. Robot. Res*. 32, 1238–1274. doi: 10.1177/0278364913495721

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., et al. (2005). "TinyOS: an operating system for sensor networks," in *Ambient Intelligence* (Berlin;Heidelberg: Springer), 115–148.

Mallow, J., Bernarding, J., Luchtmann, M., Bethmann, A., and Brechmann, A., (2015). Superior memorizers employ different neural networks for encoding and recall. *Front. Syst. Neurosci.* 9:128. doi: 10.3389/fnsys.2015.00128

Pilarski, P. M., Sutton, R. S., and Mathewson, K, W. (2015) . "Prosthetic devices as goal-seeking agents," in *Second Workshop on Present and Future of Non-Invasive Peripheral-Nervous-System Machine Interfaces: Progress in Restoring the Human Functions* (Singapore).

Rummery, Gavin A., and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Cambridge: Cambridge University.

Russell, S.J., and Norvig, P., (2016). *Artificial Intelligence: A Modern Approach.* Berkeley, CA: Pearson Education Limited.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 484–489. doi: 10.1038/nature16961

Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Mach. Learn.* 38, 287–308. doi: 10.1023/A:1007678930559

Sutton, R. S., and Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge: MIT Press.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). "Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction," in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (Taipei), 761–768.

Talamini, F., Alto, G., Carretti, B., and Grassi, M. (2017). Musicians have better memory than nonmusicians: A meta-analysis. *PLoS ONE* 12:e0186773. doi: 10.1371/journal.pone.0191776

Tanner, B., and White, A. (2009). RL-Glue: language-independent software for reinforcement-learning experiments. *J. Mach. Learn. Res.* 10, 2133–2136. doi: 10.1145/1577069.1755857

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput.* 6, 215–219. doi: 10.1162/neco.1994.6.2.215

Watkins, C. J. C. H., and Dayan, P. (1992). Q-learning. *Mach. Learn.* 8, 279–292. doi: 10.1007/BF00992698

Wei, G., Hus, D., Lee, W. S., Shen, S., and Subramanian, K. (2017). Intention-Net: integrating planning and deep learning for goal-directed autonomous navigation. arXiv preprint arXiv:1710.05627.