Multiple-Model Q-Learning for Stochastic Reinforcement Delays

Jeffrey S. Campbell[1], Sidney N. Givigi[2], and Howard M. Schwartz[3]

***Abstract*— The main contribution of this work is a novel machine reinforcement learning algorithm for problems where a Poissonian stochastic time delay is present in the agent's reinforcement signal. Despite the presence of the reinforcement noise, the algorithm can craft a suitable control policy for the agent's environment. The novel approach can deal with reinforcements which may be received out of order in time or may even overlap, which was not previously considered in the literature. The proposed algorithm is simulated and its performance is compared to a standard Q-learning algorithm. Through simulation, the proposed method is found to improve the performance of a learning agent in an environment with Poissonian-type stochastically delayed rewards.**

***Index Terms*— Reinforcement learning, Markov Decision Process, stochastic time delay, reward, cost, jitter, multiple models**

I. INTRODUCTION

Reinforcement learning (RL) is a machine learning control scheme. It is useful for applications where an agent must learn from its interactions with an environment for which it does not have a complete model and lacks supervision. In reinforcement learning, the agent learns to map actions to states based on feedback in the form of a reinforcement signal (herein called a reward) [1]–[3]. Intuitively, the agent is rewarded for desirable behaviour, and punished for behaving poorly.

RL techniques have been applied to a diversity of control problems, including adaptive PID tuning [4], arterial traffic control [5], autonomous navigation control for virtual vehicles [6], mobile robots [7], and engine emission control [8], to name a few.

RL is often used to solve problems which can be modelled by Markov Decision Processes (MDPs). MDPs are defined by a state set S and an action set A. The decision maker can perform an action a from A to move from state s to state s' within S, with probability $P(s,a,s')$. Finally, actions in A can incur rewards from the reward set $R(s,a,s')$. Thus, an MDP can be expressed as a 4-tuple $(S,A,P(s,a,s'),R(s,a,s'))$ [1].

RL is useful because it can be simpler to define the objectives that a solution must meet rather than explicitly defining the solution itself. A designer can specify *what* needs to be done rather than *how* it needs to be done [9]. For example, if the objective is to make a robot walk quickly and smoothly, a designer may use an RL scheme to give rewards for smoothness and speed. The robot will then attempt to learn a method of movement which fulfils these goals, removing the need for the designer to program the behaviour itself. The robot may then invent superior methods for doing things which would not have occurred to the designer. Normally, it is assumed that the reward signal is undelayed, that is, the designer's rewards arrive immediately after the robot's behaviour which caused them.

In applications where control is decentralized, such as in swarms of mobile robots, a single robot may not be able to receive a reward signal immediately. For example, it may be necessary to consult with other robots in the swarm first to evaluate its control policy and produce a reward. As well, in applications where network latency is a factor, *jitter* (variable time delays) can be introduced [10]. In these ways, time delays may be introduced into the reward signal. This paper will argue that jitter can have a negative impact on traditional reinforcement learning algorithms.

In the case where the reward signal is delayed by a constant, it has been shown that the resulting deterministically delayed Markov Decision Process (MDP) can be reduced to an undelayed MDP by using a larger state space and more complex cost structure [10], [11].

When the reward signal is delayed stochastically, the problem has been modelled as a stochastically delayed MDP (SDMDP) where the number of steps between the receipt of successive rewards is assumed to be a non-negative random variable. In other words, the time between rewards is random, but they are always collected in the proper order. Under this assumption, it has been shown that it is also possible to reduce an SDMDP to an MDP with a larger state space and more complex cost structure, as in the constant-delay case [10].

II. CONTRIBUTIONS AND OUTLINE

The main contribution of this paper is to provide a novel method for generating a control policy when the assumption of successive reward collection is relaxed. This means that the rewards may arrive out of order, or even overlap. As well, the paper explores and comments on different aspects of the problem of stochastic reward delay using MATLAB simulations. We begin by introducing mathematical preliminaries in Section III. Section IV contains simulation results which demonstrate the deteriorating performance of an RL agent in an environment where the rewards are stochastically delayed. The effect of larger delays and larger environment sizes is also examined.

[1]J. Campbell is with the Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada campbelljeffrey@sce.carleton.ca

[2]S. Givigi is with the Department of Electrical and Computer Engineering, Royal Military College of Canada, Kingston, ON, Canada sidney.givigi@rmc.ca

[3]H. Schwartz is with the Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada schwartz@sce.carleton.ca

Section V contains the contribution of the novel method which can improve the quality of the control policy that the agent learns in a stochastic environment, even as the number of states in the environment grows. This method combines Q-learning with multiple model approaches used in control problems where plant parameters are prone to change [12]. Section VI briefly describes and refers to the proof of convergence for the novel algorithm which was presented in another paper. Simulation results which demonstrate the improved performance of this novel method are presented and discussed in Section VII. Finally, Section VIII contains concluding remarks.

III. MATHEMATICAL PRELIMINARIES

A. Reinforcement Learning

Reinforcement learning problems can be expressed as a finite Markov Decision Process (MDP) in terms of a 5-tuple (S,A,P,R,γ) where S is the set of states within the environment, A is the set of actions the agent may choose, and P maps $S\times A\times S\mapsto[0,1]$ which is the probability that taking action $a\in A$ while in state $s\in S$ will lead to state $s'\in S$. R is defined as the reward signal which maps $S\mapsto\mathbb{R}$ and γ is the discount factor to be applied to future rewards [11].

The aim of learning is to produce a control policy, $\pi:S\mapsto A$. This policy maps the states to appropriate actions so as to maximize a value function, which is the expectation of future cumulative discounted rewards, given by:

$$V^{\pi}(s)=R(s)+\gamma\sum_{s'}P(s,\pi(s),s')V^{\pi}(s') \tag{1}$$

where $0<\gamma<1$. This can be thought of as the expected utility of being in state s and following policy $\pi(s)$ thereafter [11].

B. Q-learning

In this reinforcement learning algorithm, the agent updates a table Q with entries $Q(s,a)$. When the agent transitions from state s at time t to new state s' at time $(t+1)$, having taken action a at time t, the table is updated according to:

$$\begin{aligned}Q_{k+1}(s,a)\leftarrow{}&Q_k(s,a)\\&+\alpha[r+\gamma\max_a Q_k(s',a)-Q_k(s,a)]\end{aligned} \tag{2}$$

where $0\leq\alpha<1$ is a learning rate, γ is a discount factor, and r is a reward received at time t [1].

C. Variably Delayed Markov Decision Processes

For the purpose of the present paper, a Variably Delayed Markov Decision Process (VDMDP) is defined as an MDP where the reward signal suffers from a Poissonian time delay. This means that if the agent performs an action, it will not receive the corresponding reward immediately. A VDMDP can be characterized by a 6-tuple (S,A,P,R,γ,λ) where λ represents the mean and variance of the Poisson distribution for the time delay.

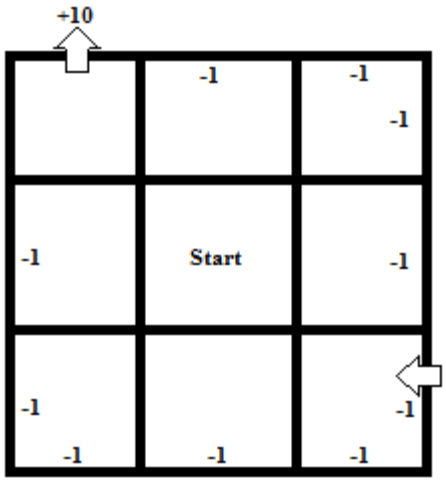

Fig. 1. 3×3 grid world simulation environment

IV. SIMULATION ENVIRONMENTS

Simulations were conducted in grid-world environments of differing sizes. In particular, square grid-worlds of size 3×3, 5×5, and 9×9 were used. In each of these environments, the agent begins in a random state. The top left-most state yields a reward of 10 for any action and moves the robot to the bottom right-most state. Bumping into a wall yields a reward of -1. Thus, an optimal policy is any which has the robot moving directly from bottom-right to top-left along some path of minimum distance. Note that diagonal moves are not allowed. A 3×3 grid world is shown in Fig. 1.

The normalized performance x of a learning algorithm at time t with mean Poisson time delay of λ can be evaluated as:

$$x_{\lambda}(t)=\frac{1}{t}\frac{\sum_{i=1}^{t}r_{\lambda}(i)}{\sum_{i=1}^{t}r_0(i)} \tag{3}$$

where $r_{\lambda}(i)$ is the series of rewards received in an environment with Poisson delay of mean λ and $r_0(i)$ is the series of rewards received when normal Q-learning is used without delay present. In the undelayed case, when the learning rate is sufficiently reduced over time and sufficient exploration is allowed, the agent's policy will converge to the optimal policy [1]. The performance of a delayed agent's policy is therefore expressed as a percentage of an optimal policy, as achieved by Q-learning, for an arbitrary MDP. This allows us to compare results across different environments in a reasonable way.

A. Effect of Larger Delays

Fig. 2 shows that as the mean delay λ grows larger, the performance of the learning agent decreases. When the mean delay is small, there is a significant chance that the reward will be assigned to the correct cause because there are only nine states. However, as the delay grows, the chance that the

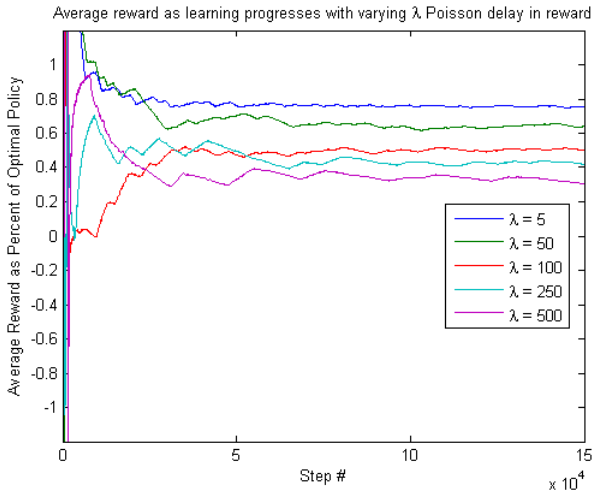

Fig. 2. Effect of increasing the mean delay λ in a 3×3 grid world

reward will be correctly assigned decreases and the agent's learning is hampered.

B. *Effect of Environment Size*

Simulations in 3×3, 5×5, and 9×9 square grid world environments, such as that in Fig. 1, showed that as the environment size increased, the performance of the learning agent decreased relative to the optimal policy for each respective environment. Using $\lambda = 5$ arbitrarily, the larger environment size had a negative effect on performance as can be seen in Fig. 3. As the environment size increases, the $+10$ reward state becomes relatively rarer and performance falters.

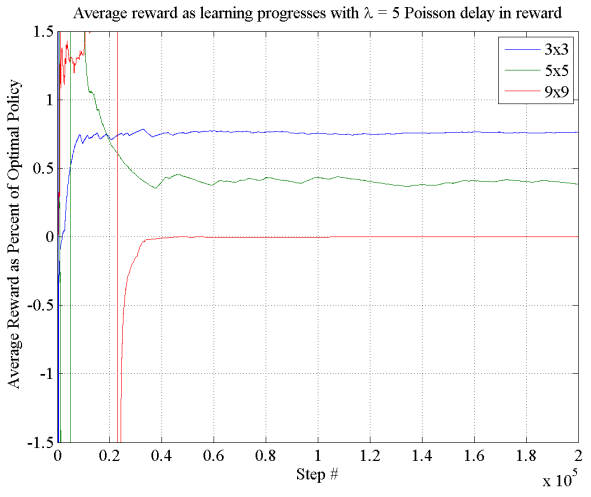

Fig. 3. Effect on performance of increasing the grid world size

As a result, the problem of stochastically delayed rewards becomes more important as the number of states in the environment grows in size because it becomes less likely that the agent will stumble luckily upon the reward state. As well, it becomes less likely that the agent will assign a reward to the proper state-action pair as the number of states increases because the correct state becomes relatively rarer. Because real-world problems often have large state spaces and because we want to an agent to be robust despite large reward delays, a new algorithm is needed to facilitate learning under these new constraints.

V. MULTIPLE-MODEL Q-LEARNING

A contribution of this paper is a solution to the observation that assigning the rewards becomes more difficult as the mean delay and environment become larger because the correct state-action cause becomes relatively rarer. When the mean delay is unknown, the agent still assigns credit properly some of the time by chance, and some learning takes place, but such luck cannot be counted on in large environments with large mean delays.

To address this, assume temporarily that the mean reward delay is known to be λ and any rewards are attributed to the actions that are most likely to be responsible. This means that the agent will assume that the cause of any reward is the state-action pair which occurred λ time steps in the past because this is the peak of the Poisson distribution. In other words, the most likely outcome of the Poisson distribution is the mean, λ. When this idea is implemented into Q-learning, it performs significantly better because credit is assigned properly more frequently. Observe that in Fig. 4, the performance of the agent takes on the shape of a Poisson distribution approximately centered around the true mean delay λ. Let the agent's estimate of the true mean λ be $\hat{\lambda}$. When the agent assumes that the mean delay is $\hat{\lambda} = \lambda + i$, the estimate strays farther from the most probable state-action cause and the agent's performance worsens. For example, if the true mean delay is $\lambda = 17$, but the agent assumes that it is $\hat{\lambda} = 20$, then we have an error of $i = 3$.

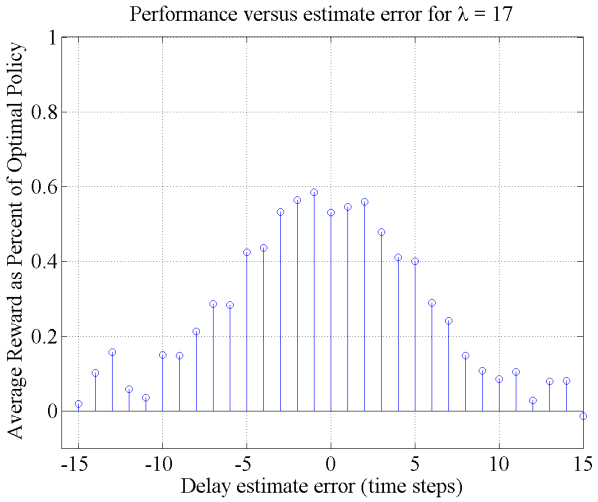

Fig. 4. 9 x 9 grid world - Performance after 150k iterations for errors in the delay estimate $i = \hat{\lambda} - \lambda$

Thus, if an estimate of the mean delay, $\hat{\lambda}$, is available the Q-table can be updated in a novel way:

$$Q_{k+1}(s_{t-\hat{\lambda}}, a_{t-\hat{\lambda}}) \leftarrow Q_k(s_{t-\hat{\lambda}}, a_{t-\hat{\lambda}}) + \alpha[r_t + \gamma \max_a Q_k(s'_{t-\hat{\lambda}}, a)] - \alpha[Q_k(s_{t-\hat{\lambda}}, a_{t-\hat{\lambda}})] \quad (4)$$

where the index $t-\hat{\lambda}$ refers to the time step which happened $\hat{\lambda}$ time steps in the past. For example, if $\hat{\lambda}=5$, then we are updating the state-action pair which occurred five time steps ago. This requires that we maintain a memory of all state-action pairs up to some threshold $\hat{\lambda}_{\max}$, which we reasonably define based on the needs of the application.

If we keep track of different delay estimates in parallel, then the agent can gradually learn which is most valuable and learn more from its environment than before. This brings us to our proposed algorithm.

A. Multiple Model Q-Learning

This is the main contribution of this paper. It may not be possible to know the mean delay *a priori*, but the agent can be made to learn the delay over time using a multiple-model method for learning the mean reward delay. Rewards can be assigned to the state-action pairs which are most likely to have caused them and policy quality is increased significantly.

Let $\hat{\lambda}_{max}$ be the maximum feasible time delay for a particular application as estimated by the designer. The action-space is augmented with N estimates for the mean reward time delay, increasing its size to $A \times N$. These estimates run from 0 time steps to $\hat{\lambda}_{max}$ time steps at some resolution. (E.g. if the resolution is 1, then $N=\hat{\lambda}_{max}+1$ if $\hat{\lambda}_{max}$ is an integer) Thus, for each state, the agent must now select both an action and a delay estimate. The models can be visualized as in Fig. 5. In each time step, a candidate model $\hat{\lambda}$ is selected and its Q-table is made active for decision making during that time step. The dashed lines indicate inactive models, which are still updated based on decision made by the active model.

Building upon the Sarsa algorithm from [1], the agent learns as detailed in Algorithm 1. First, the agent selects the most valuable time delay estimate $\hat{\lambda}^*$ where:

$$\hat{\lambda}^* = \arg\max_{\hat{\lambda}} Q(s,a,\hat{\lambda}) \quad (5)$$

and then assumes that this estimate is correct. Then, as normal, the agent selects the most valuable action according to:

$$a^* = \arg\max_a Q(s,a,\hat{\lambda}^*) \quad (6)$$

given that the agent is currently in state s under the assumption that $\lambda=\hat{\lambda}^*$. The agent then performs action a^* and arrives at state s'. A reward is produced, but the agent will not receive this reward until the stochastic delay has elapsed.

During this time step t, a reward is received which may be the sum of several rewards (caused by several actions), a single reward (caused by one action) or no rewards at all (a reward of zero). Overlap is possible as a result of the relaxation of the assumption that the rewards are delivered in proper order. For example, a reward may be delayed by six time steps and then the next reward is delayed by five time steps. In five time steps, the agent would receive the sum of the two rewards.

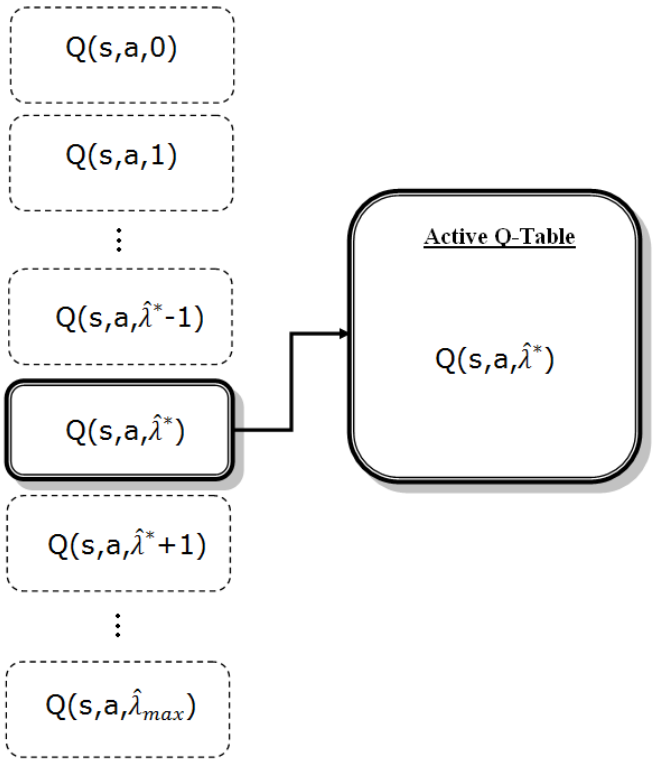

Fig. 5. Activation of a candidate model in Multiple-Model Q-Learning

Next, the agent cycles through all N estimate models and assigns the reward at time step t to the state-action pair at time step $t-\hat{\lambda}$ such that:

$$Q_{k+1}(s_{t-\hat{\lambda}}, a_{t-\hat{\lambda}}, \hat{\lambda}) \leftarrow (1-\alpha)Q_k(s_{t-\hat{\lambda}}, a_{t-\hat{\lambda}}, \hat{\lambda}) + \alpha\left[r_t + \gamma \max_{\hat{\lambda},a} Q_k(s'_{t-\hat{\lambda}}, a, \hat{\lambda})\right] \quad (7)$$

where $\hat{\lambda}$ is the $\hat{\lambda}^{\text{th}}$ model of N total models. In this way, the agent repeatedly evaluates N different estimates at once. The agent then chooses the next best candidate estimate model $\hat{\lambda}$ and the cycle continues until sufficient exploration of the environment is achieved.

After learning is finished, a policy $a=\pi(s,\hat{\lambda})$ can be produced by finding the most valuable parallel Q-table, then the most valuable action for the current state within that Q-table. This procedure is the same as during learning, except the agent is no longer allowed to explore randomly. Instead, it strictly behaves according to its Q-table.

Alternatively, the agent could create a probabilistic control policy. First, the most valuable model $\hat{\lambda}^*$ is chosen for the current state s. Then, the policy $a=\pi(s,\hat{\lambda}^*)$ indicates that the agent should select action a when in state s with probability

Algorithm 1 - Multiple Model Q-Learning

Set $\hat{\lambda}_{\max}$
Initialize state-action memory of sufficient length
Initialize α, γ (e.g. $\alpha = 0.1, \gamma = 0.9$)
Initialize $Q(s,a,\hat{\lambda})$ arbitrarily (e.g. optimistic initialization)
for episode **do**
  Initialize s
  repeat
    select $\hat{\lambda}^*$ using policy from Q (e.g. ε-greedy)
    select a^* using policy from Q assuming $\hat{\lambda}^*$ is correct (e.g. ε-greedy)
    execute action a^*
    observe r_t and s'
    for $\hat{\lambda}$ from 0 to $\hat{\lambda}_{max}$ **do**

$$Q_{k+1}(s_{t-\hat{\lambda}}, a_{t-\hat{\lambda}}, \hat{\lambda}) \leftarrow (1-\alpha) Q_k(s_{t-\hat{\lambda}}, a_{t-\hat{\lambda}}, \hat{\lambda}) + \alpha \left[r_t + \gamma \max_{\hat{\lambda}, a} Q_k(s'_{t-\hat{\lambda}}, a, \hat{\lambda}) \right]$$

    end for
    $s \leftarrow s'$
  until s is terminal or behaviour is acceptable
end for

p where:

$$p(a|s) = \frac{Q(s,a,\hat{\lambda}^*) - \min_a Q(s,a,\hat{\lambda}^*)}{\sum_{a=1}^{|A|} Q(s,a,\hat{\lambda}^*)} \tag{8}$$

where $|A|$ is the total number of actions. In this way, the agent will continuously use different paths to reach its goal.

VI. CONVERGENCE

Multiple-model Q-learning converges in a stochastic sense, as shown in [13]. The proof is an extension of the work in [14], which states that if the following four assumptions are true, and if $x_i = x_i + \alpha(F_i(x) - x_i + w_i)$, then $x(t)$ converges to x^* with probability 1.

Assumption 1 *For any i and j,* $\lim_{t\to\infty} \tau_j^i(t) = \infty$*, with probability 1.*

Assumption 2 *Any random variables are defined within a probability space* $(\Omega, \mathscr{F}, P)$ *and there exists an increasing sequence* $[\mathscr{F}(t)]_{t=0}^{\infty}$ *of subfields of* $\mathscr{F}$ *such that: a)*

1) *x(0) is* $\mathscr{F}(0)$*-measurable;*
2) *For every i and t,* $w_i(t)$ *is* $\mathscr{F}(t+1)$*-measurable;*
3) *For every* i, j*, and* $t, \alpha_i(t)$ *and* $\tau_j^i(t)$ *are* $\mathscr{F}(t)$*-measurable.*
4) *For every i and t, we have* $E[w_i(t)|\mathscr{F}(t)] = 0$*;*
5) *and there exist (deterministic) constants A and B such that*

$$E[w_i^2(t)] \leq A + B \max_j \max_{\tau \leq t} \left| x_j(\tau) \right|^2 \forall i, t$$

Assumption 3 *For every s,*

$$\sum_{t=0}^{\infty} \alpha_s(t) = \infty, \ \textit{w.p.1.} \tag{9}$$

There exists some (deterministic) constant C such that for every s,

$$\sum_{t=0}^{\infty} \alpha_s^2(t) \leq C, \ \textit{w.p.1.} \tag{10}$$

Assumption 4 *Defining the norm* $||\cdot||_v$ *on* $\mathbb{R}^n$ *as*

$$||x||_v = \max_i \frac{|x_i|}{|v_i|} \tag{11}$$

there exists a vector $x^* \in \mathbb{R}^n$*, a positive vector v, and a scalar* $\gamma \in [0,1)$*, such that*

$$||F(x) - x^*||_v \leq \gamma ||x - x^*||_v \quad \forall x \in \mathbb{R}^n \tag{12}$$

The work in [13] shows that these four assumptions are satisfied, and that therefore, $Q_k(s,a,\hat{\lambda})$ converges to $Q(s,a,\hat{\lambda})^*$ with probability 1.

VII. SIMULATION

Simulations were conducted in a 9 x 9 grid world to demonstrate how the novel algorithm can derive a control policy for the problem. The simulations were performed in a grid world environments to demonstrate the significant effect of stochastic time delays even in a relatively small and simple environment. The solutions described in this paper are intended to be carried on to other more complex applications, especially in mobile robotics, where grid world simulations are useful for testing algorithms.

During simulation, the mean delay is set to $\lambda = 13$, and the highest feasible delay estimate is set to 50 time steps. This sets the maximum number of parallel estimate models which must be maintained. Other parameters are set as the examples in Algorithm 1. Fig. 6 shows the improved performance from using the multiple-model Q-learning algorithm in a 9×9 grid world. The multiple-model Q-learning method achieves a performance of about 95% relative to single-model Q-learning in an undelayed environment. In comparison, tabular Q-learning achieves a performance of about 0% relative to the undelayed environment.

When learning is finished, the agent has travelled through the states and explored the environment, as in Fig. 7. Then, if a deterministic control policy is formed using the maximally valuable entries in the Q-table, the learning agent will behave as in Fig. 8. If the agent were to begin in state 81, (9,9) then it would move roughly along the diagonal to reach the reward state. The optimal path consists of taking some combination of the actions *up* and *left* to reach the reward state directly. Through the states most visited during learning, the preferred actions tend to be *up* (light blue) and *left* (pink). Indeed, the multiple model Q-learning policy is optimal, but performance during learning is reduced by the forced exploration present in the ε-greedy strategy. This is why performance during learning is 95% of optimal, and after learning it is 100% of optimal.

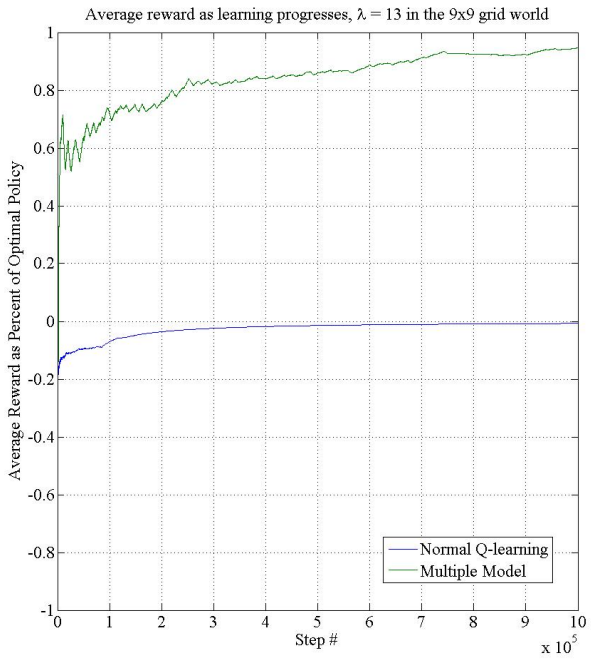

Fig. 6. Relative performance of multiple-model Q-learning vs single-model Q-learning in the 9×9 grid world with random delays present

Similarly, Fig. 9 shows the probabilistic policy formed from the same Q-table, where the agent will move in each direction with a given probability based on the current state, as described by the color scale. Warmer colors indicate a higher probability of taking that action in that state.

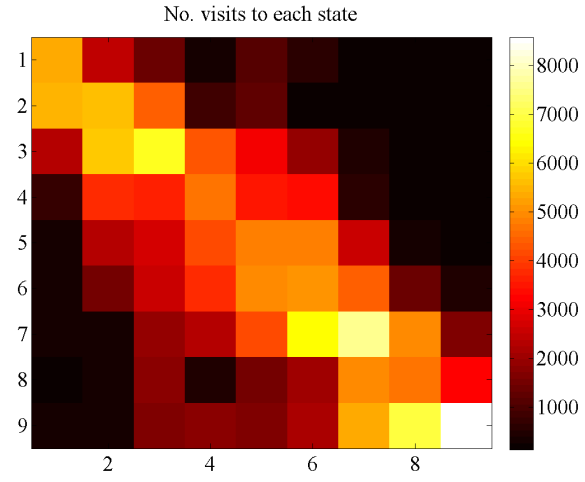

Fig. 7. How often each state is visited during learning when using multiple-model Q-learning

A. Comparison to Q-learning

As seen in Fig. 6, when using the basic Q-learning algorithm, the agent does not achieve a positive reward, meaning that no significant learning takes place. The following results

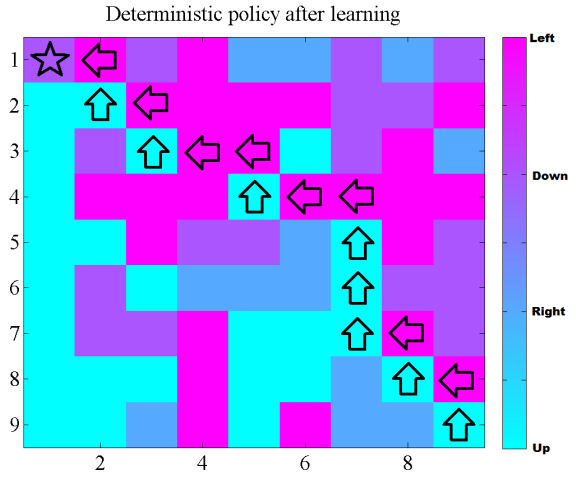

Fig. 8. Deterministic policy formed using multiple-model Q-learning

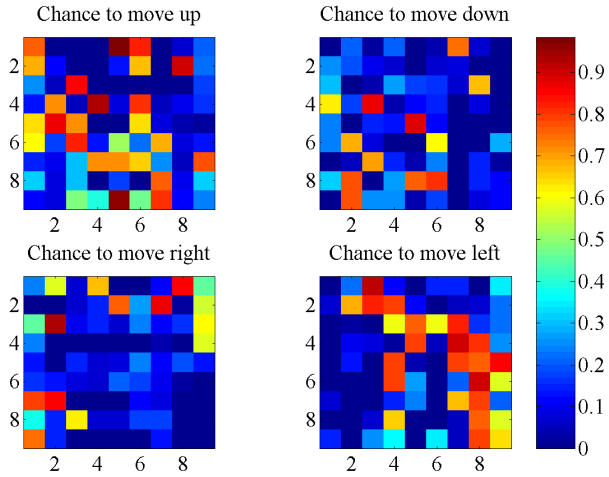

Fig. 9. Probabilistic policy formed using multiple-model Q-learning

describing the behaviour of the agent demonstrate that single-model Q-learning (wherein the delay is assumed to be $\lambda = 0$) is unable to solve the problem when stochastically delayed rewards are present. The Fig. 10 visitation information shows that if single-model Q-learning is used within the VDMDP, the agent becomes stuck in a loop during learning and explores poorly. The agent was stuck around state (6,4).

Fig. 11 displays the problematic deterministic policy formed from the exploration in Fig. 10. No clear path emerges which would move the agent from bottom-right to the reward state at top-left. Instead, the agent becomes stuck in a loop. However, the agent does learn to avoid the walls, although it fails to reach the goal state.

Fig. 12 is the probabilistic policy formed from Q-learning. The poor exploration leads to a policy of mostly random behaviour because most of the states have not been visited extensively. Note that warmer colors indicate more decisive behavior.

After running both policies for 1,000 iterations after

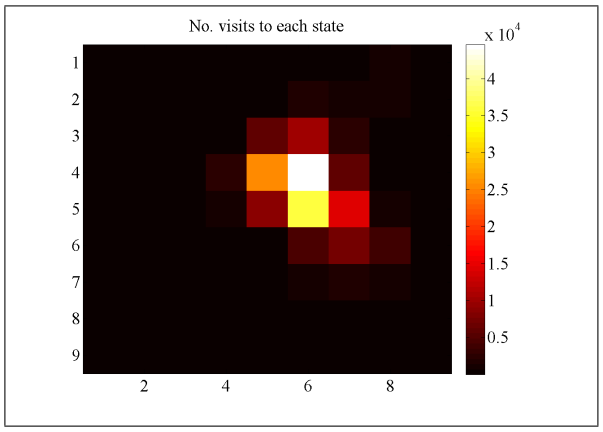

Fig. 10. How often each state visited during learning when using single-model Q-learning

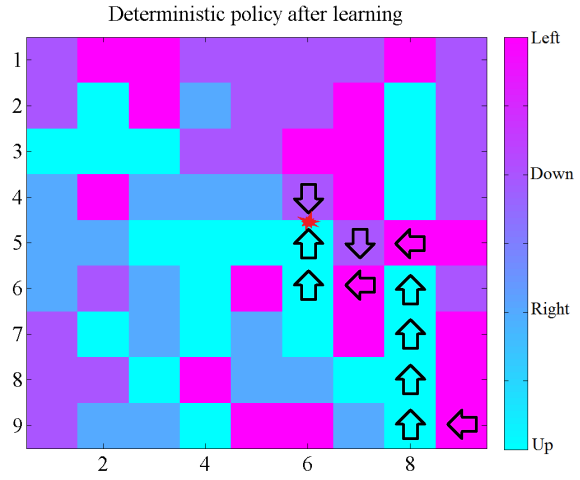

Fig. 11. Deterministic policy formed from using single-model Q-learning

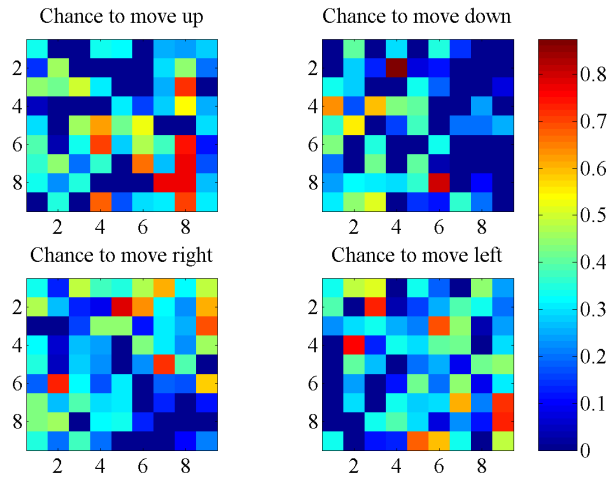

Fig. 12. Probabilistic policy formed from using single-model Q-learning

learning is complete, the multiple-model Q-learning control policy achieves an average reward of 0.59 (+10 in 17 steps, the optimal) and the Q-learning policy achieves an average reward of 0.0 (it becomes stuck in a loop). These results mean that the novel method facilitates more learning than the Q-learning method, despite the stochastic reward delay.

VIII. CONCLUSIONS

This paper introduced a novel method for learning in environments where stochastic time delays are present in the agent's reward signal. The contributions of this paper were simulations demonstrating the worsening performance of traditional reinforcement learning schemes for increasing time delays and environment size, and a novel solution which suggests a way to assign credit so as to increase the probability of learning taking place under these circumstances.

By simulation, the current paper has shown that introducing a Poissonian time delay into the reward signal of a reinforcement learning agent reduces the performance of standard Q-learning. Further, it is shown in simulation that the proposed multiple-model algorithm for assigning credit for a reward to state-action pairs which are delayed in the past by an amount near to the mean reward time delay can significantly improve performance of the learning agent and enable more learning to take place.

Future work will delve into new ways of processing the parallel delay estimate updates in parallel rather than serial fashion, since the updates are independent. It may also be fruitful to investigate using a kernel to assign less reward to delay estimates which are far from the most valuable delay estimate, $\hat{\lambda}^*$. As well, the method will be applied to new control problems and mobile robot experiments.

REFERENCES

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[2] P. Kulkarni, "Introduction to Reinforcement and Systemic Machine Learning," Reinforcement and Systemic Machine Learning, pp. 1-21, 2012.

[3] M. P. Deisenroth, G. Neumann, J. Peters, "A Survey on Policy Search for Robotics," Foundations and Trends in Robotics, 2011.

[4] M. N. Howell, T. J. Gordon, M. C. Best, "The application of continuous action reinforcement learning automata to adaptive PID tuning," Learning Systems for Control (Ref. No. 2000/069), IEE Seminar, pp.2/1,2/4, 2000.

[5] J. C. Medina, A. Hajbabaie, and R. F. Benekohal, "Arterial traffic control using reinforcement learning agents and information from adjacent intersections in the state and reward structure," 13th International IEEE Conference on Intelligent Transportation Systems, no. 3, pp. 525-530, Sep. 2010.

[6] N. Lianqiang, Li Ling, "Application of Reinforcement Learning in Autonomous Navigation for Virtual Vehicles," Hybrid Intelligent Systems, 2009. HIS '09. Ninth International Conference on , vol.2, pp.30,32, 12-14 Aug. 2009.

[7] K. Ito, Y. Imoto, H. Taguchi, and A. Gofuku, "A study of reinforcement learning with knowledge sharing," in IEEE International Conference on Robotics and Biomimetics, 2004.

[8] P. Shih, B. C. Kaul, S. Jagannathan, J. A. Drallmeier, "Reinforcement-Learning-Based Output-Feedback Control of Nonstrict Nonlinear Discrete-Time Systems With Application to Engine Emission Control," Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on , vol.39, no.5, pp.1162,1179, Oct. 2009

[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," arXiv preprint cs/9605103, Jan. 1996.

[10] K. V Katsikopoulos, “Markov decision processes with delays and asynchronous cost collection,” Automatic Control, Jan. 2003.
[11] T. Walsh, A. Nouri, L. Li, and M. Littman, “Planning and learning in environments with delayed feedback,” Machine Learning: ECML 2007, Jan. 2007.
[12] A. S. Campbell and H. M. Schwartz, “Multiple model control improvements: hypothesis testing and modified model arrangement,” Control and Intelligent Systems, Jan. 2007.
[13] J. S. Campbell, S. N. Givigi and H. M. Schwartz, “Multiple Model Q-learning for Stochastic Time-Delayed Reinforcement Learning,” submitted to Journal of Intelligent & Robotic Systems, 2014.
[14] J. N. Tsitsiklis, “Asynchronous Stochastic Approximation and Q-learning”, Machine Learning, 16, 1994, pp. 185-202.