51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition
07 - 10 January 2013, Grapevine (Dallas/Ft. Worth Region), Texas
AIAA 2013-0875

Approximation of Agent Dynamics Using Reinforcement Learning

Kenton Kirkpatrick* and John Valasek†
Texas A&M University, College Station, Texas 77843-3141

Reinforcement Learning for control of dynamical systems is popular due to the ability to learn control policies without requiring a model of the system being controlled. It can be difficult to learn ideal control policies because it is common to abstract out or ignore completely the dynamics of the agents in the system. In this paper, Reinforcement Learning-based algorithms are developed for learning agents' time dependent dynamics while also learning to control them. Three algorithms are introduced. Sampled-Data Q-learning is an algorithm that learns the optimal sample time for controlling an agent without a prior model. First-Order Dynamics Learning is an algorithm that determines the proper time constants for agents known to have first-order dynamics, while Second-Order Dynamics Learning is an algorithm for learning natural frequencies and damping ratios of second-order systems. The algorithms are demonstrated with numerical simulation. Results presented in this paper show that the algorithms are able to determine information about the system dynamics without resorting to traditional system identification.

I. Introduction

In recent years, Reinforcement Learning (RL) has been an extensively investigated area of research in the field of Machine Learning. It has been a popular tool for solving problems such as dynamical system control, gain scheduling, maze navigation, and game playing. There has been wide success in many of the applications, but researchers have often encountered problems when casting the control of dynamical systems as an RL problem. The state-to-action mapping provided by RL techniques makes the use for control problems attractive, and this is especially the case with Q-learning due to its proven convergence to optimality. RL methods like Q-learning are appealing because they can achieve this mapping experimentally without the need of a model. Although this is indeed the case, implementing these in practice has proven to be very difficult.

Studying the problems associated with this implementation reveals that often the failure to implement RL methods in dynamical systems are not caused but the basic approach of methods like Q-learning. Typically, the problem is either a failure in properly representing the problem or inaccurate function approximation. When choosing to implement the popular Watkins' Q-learning in a dynamical system scenario, it is necessary to realize that the algorithm does not explicitly account for time. Time dependency is often either overlooked or handled outside of the learning process, but accounting for time in the selection of actions (or control) is needed. For instance, when handling sampled-data systems, small changes to sample time can cause drastic changes to the stability of the control policy determined.

One research area that has recently received a lot of attention has been the control of cooperative multi-agent systems through the use of Q-learning-based algorithms. Learning to control multiple agents for the purpose of cooperatively achieving a specified goal is an appealing research topic with high complexity. Some research in this area has involved comparing the effects using Q-learning-based methods to determine joint action selection between different agents to the learning of agent actions independently.[1] Other research has investigated stochastic game extensions to this and treated the system as non-cooperative by having agents

*Graduate Research Assistant, Vehicle Systems & Control Laboratory, Aerospace Engineering Department. Student Member AIAA. *kentonkirk@gmail.com*

†Professor and Director, Vehicle Systems & Control Laboratory, Aerospace Engineering Department. Associate Fellow AIAA. *valasek@tamu.edu*. *Website: vscl.tamu.edu/valasek*

consider only themselves with no knowledge of the existence of other agents.[2] Systems of agents that need to coordinate their actions without knowledge of each other's actions has been determined to be an important area of research for the general application to systems of agents that do not have the ability to communicate with one another.[3,4] Even so, other research has been conducted involving the improvement of Q-learning approaches for determining joint actions through the use of Bayesian inference to estimate strategies.[5]

In most of the research scenarios discussed above, multiple agents are simulated in games that have no dependence on time. Time-dependent agent dynamics cause a fundamental change to the system, and considering the control of time dependencies in multi-agent systems has received little attention. This may be due to the fact that it is difficult to learn control policies for a single time-dependent agent using RL approaches. To address this, a topic of research that needs to be investigated is the learning of an optimal sample time for a sampled-data system. Controlling real continuous systems generally requires computer-based control, so sampling of the continuous system is necessary. Without considering the sample time used, problems arise in attempting to use a policy derived in simulation on an actual hardware experiment. For this scenario, if a model of the dynamics exists it is trivial to determine the best sample time by classical methods. However, here we consider the case where a model is not available and RL is being utilized for its model-free approach. This requires some way to determine the optimal sample time without the use of a model.

One more area of investigation that is needed for the learning of multi-agent system control policies is to learn some approximation of the dynamics. The learning process does not explicitly account for agent time dynamics, so that information is essentially abstracted out of learning. Since the main benefit of RL approaches like Q-learning is to learn a control policy without the need of a model, the benefits of determining a model have been overlooked. It can be very useful for some knowledge of the dynamics to inform the decisions made by the agents, and this is especially true in heterogeneous multi-agent systems. A full model may not be necessary, but some approximation of the individual agent dynamics can be very beneficial for determining global behavior rules.

In this paper, RL-based control approaches are extended to systems with unknown time dynamics. The scope of this paper is limited to the cases of simulated examples where the simulations have time dynamics but the RL agent learning to control the system has no access to that information. The systems considered are all sampled-data systems, and they will exhibit first- or second-order dynamics depending upon the algorithm being investigated. Three algorithms are introduced. One is capable of determining the optimal sample time for these sampled-data systems using RL-based algorithms while simultaneously learning the control policy, and it is named Sampled-Data Q-learning (SDQL). The second is an RL-based algorithm capable of learning some approximation of agent dynamics for a first-order system, and it is called First-Order Dynamics Learning (FODL). The final algorithm is similar to the FODL algorithm and is for second-order systems, called Second-Order Dynamics Learning (SODL). The end result of this research is the ability to learn an optimal sample time for agents, an approximation of agents' time dynamics, and individual agent control policies. This collective result allows for the control of heterogeneous multi-agent systems by means of hierarchical commands provided by a high-level agent with all of the knowledge learned by these algorithms.

This paper is organized as follows. In Section II, the basics of Reinforcement Learning are discussed, with an emphasis on Q-learning. Section III introduces the Sampled-Data Q-learning algorithm and includes simulation results to demonstrate it. The First- and Second-Order Dynamics Learning algorithms are introduced in Section IV with accompanying results, and conclusions and open challenges are discussed in Section V.

II. Q-learning

There are multiple classes of algorithms that fall within the definition of Reinforcement Learning. Currently, the RL algorithms that are most used in research are Temporal-Difference (TD) methods. TD methods are actually a conceptual combination from two other classes of algorithms known as Dynamic Programming (DP) and Monte Carlo.[6] Like Monte Carlo, TD methods use experience through interaction with the system to update the quality of the value function without the need of a model. Like DP, TD methods do not have to wait until the end to improve the value function, but rather update it along the way. This improves convergence time and also makes TD methods usable in online learning. In this section, the TD algorithm known as Q-learning will be discussed, along with the limitations that lead to the development of extended

Q-learning-based algorithms for the use of learning in dynamical sampled-data systems.

Of the various formulations of RL algorithms, Watkins' Q-learning has been the most accepted and utilized algorithm for its proven convergence to the optimal action-value function.[6] Q-learning is a TD method that learns the optimal action-value function in an off-policy manner.[7] This means that the policy used during a learning episode is not necessarily the same as the one that is updated at each timestep. A similar implementation that uses on-policy learning is known as Sarsa, and is often used in the same learning situations as Q-learning.[8,9] The Q-learning algorithm is based upon an action-value update rule that uses a greedy policy to determine a predicted value for the state-action pair at the next (future) timestep.[10,11] The actual action selection may not be done using a greedy policy, and in fact it is typically better for optimality to include some degree of exploration in the policy.[12] The rule used for updating the action-value function is as follows.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{1}$$

In Equation 1, Q is the action-value function, s is the state at the current timestep, a is the action selected for the current timestep using the agent's policy (e.g., ε-greedy), α is the step-size parameter, r is the reward received from the system, s' is the future state (due to taking action a), a' is the action that would be taken using a greedy policy when in state s', and γ is the weight for the future value. The selection of γ affects convergence time, and it is always within the range (0,1). The value of γ can either be kept constant throughout learning, or it can be chosen to vary episodically so that later learning episodes value the future prediction differently than early episodes. The value of α is always in the interval (0,1), and can either be held constant or varied by some user-defined function. In some cases, α is designed to decrease within an episode according to how often the agent revisits the same state, essentially punishing the agent for repeating itself unnecessarily.

The update rule of Equation 1 is the backbone of the famous Watkins' Q-learning algorithm. It is an off-policy TD algorithm with a user-determined policy for selecting actions at each timestep, but uses a fully-greedy policy with the action-value function when updating the action-value function. Rather than utilizing past information to perform this update, this algorithm uses the predicted future state-action pair chosen greedily. The Watkins' Q-learning algorithm is displayed in Algorithm 1.

Algorithm 1 Q-learning[6]

- Initialize $Q(s,a)$ arbitrarily
- Repeat for each episode:
  - Initialize s
  - Repeat for each timestep:
    - Choose a from s using policy derived from $Q(s,a)$ (e.g., ε-Greedy)
    - Take action a, observe r, s'
    - $Q(s,a) \leftarrow Q(s,a) + \alpha\ [\ r + \gamma\ \max_{a'}\ Q(s',a')$ - $Q(s,a)\ ]$
    - $s \leftarrow s'$
  - Until s is terminal

The policy for action selection has a drastic effect on the convergence of the Q-learning algorithm. Some balance of exploration and exploitation is needed to properly learn the full state-space and guarantee convergence to the optimal policy. An ε-greedy policy uses a user-defined probability, ε, that determines whether to choose actions randomly or according to the action-value function each time a new action must be taken. This speeds up the convergence time by reinforcing paths that have already been designated either good or bad while still allowing for new paths to be explored for optimality.[13] In the early episodes the agent is required to explore due to lack of knowledge, regardless of the value assigned to ε. This ε-greedy policy is used as the action selection method for Q-learning in this paper.

III. Sampled-Data Q-learning

When Algorithm 1 is used in dynamical systems, the problem of handling time-dependencies arises. Because it is much faster (and safer) to perform the learning in a computer simulation rather than online with a hardware system, researchers often turn to using dynamics models. The issue of ensuring an accurate model is obvious, but it is often overlooked that using the final control policy on a hardware model results in a sampled-data system. The control policy is handled using a discrete computer, but the policy is learned assuming continuous dynamics. In some cases, the user realizes this and assumes a sampled-data system in the simulated learning, but the Q-learning algorithm will converge to a policy that assumes the same sample time will always be used and that the chosen sample time is best. Here, an attempt to overcome this issue is addressed by wrapping the sample time into the learning process.

A. Sampled-Data Q-learning Algorithm

Incorporating the sample time, denoted T, into the learning process requires determining the value of individual sample times without adversely affecting the stability of the system during an episode. It would therefore be wise to not allow a sample time to change during a single episode. However, to determine optimal action-value functions for a range of sample times requires incorporating it into the state-space. It is therefore necessary to append the state-space with T while not allowing it to be affected by the action-space.

This gives rise to the question of how a particular sample time is to be selected when it cannot be affected by the action-space. It is necessary that a value be associated with each possible selection of T, but T must be held constant throughout an episode. It is therefore proposed that a state-value function for T be determined using Monte Carlo-based learning.[6] The value function can be updated according to an every-visit Monte Carlo method, shown in Equation 2.

$$V_T(T) \leftarrow V_T(T) + \alpha(R - V_T(T)) \tag{2}$$

This update rule will be the basis for Sampled-Data Q-learning. At the beginning of each episode, the sample time value-function, V_T, can be used to select T for the episode according to a user-defined policy. The sample time is appended to the system state vector, s, so that the normal Q-learning update rule will determine separate control policies according to different values of T. When the reward for a given timestep is determined, it is used to update both Q, and the total rewards for the episode are updated. At the end of the episode, V_T is updated by using the average return, R. This causes V_T to be updated such that episodes which experience more positive rewards than negative rewards result in the value associated with that particular T increasing. Likewise, when an episode experiences more negative rewards than positive, the total value for T after the episode ends will have decreased. Using this new sampled-data value function and update rule, the Sampled-Data Q-learning algorithm becomes as shown in Algorithm 2.

B. SDQL Results

For the system described above, a single agent was simulated as a robot that translates forward with a constant speed and rotates the heading angle ψ to change direction. The rotational dynamics are described as a first-order differential equation with time constant τ. The environment the robot is allowed to traverse is a 20m by 20m square with the origin at the center. The governing equations of motion are shown in Equations 3-5, where the subscript c denotes the commanded value.

$$\dot{x} = V\cos\psi \tag{3}$$

$$\dot{y} = V\sin\psi \tag{4}$$

$$\dot{\psi} = (\psi_c - \psi)\tau^{-1} \tag{5}$$

At each timestep, the learner evaluates the current state and chooses the appropriate action based on an ε-greedy policy in a Q-learning scheme. The possible actions are rotate clockwise ($-\Delta\psi$), no rotation ($\Delta\psi = 0$), and rotate counterclockwise($+\Delta\psi$). At each T, the robot is required to take a new action, which corresponds to a new commanded next state. The commanded next action occurs in intervals of 45 degrees. The action-space is shown in Equation 6.

Algorithm 2 Sampled-Data Q-learning (SDQL)

- Initialize $Q(\tilde{s},a)$ arbitrarily
- Initialize $V_T(T)$ arbitrarily
- Repeat for each episode:
  - Choose T using policy derived from $V_T(T)$ (e.g., ε-Greedy)
  - Initialize $R = 0$
  - Initialize s, initialize $\tilde{s}$ by appending T to s
  - Repeat for each sample timestep, T:
    - Choose a from $\tilde{s}$ using policy derived from $Q(\tilde{s},a)$ (e.g., ε-Greedy)
    - Take action a, observe r, $\tilde{s}'$
    - $Q(\tilde{s},a) \leftarrow Q(\tilde{s},a) + \alpha\ [\ r + \gamma\ \max_{a'}\ Q(\tilde{s}',a')\ -\ Q(\tilde{s},a)\ ]$
    - $\tilde{s} \leftarrow \tilde{s}'$
  - Until $\tilde{s}$ is terminal
  - $R = \text{average}(r)$
  - $V_T(T) \leftarrow V_T(T) + \alpha\ [\ R\ -\ V_T(T)\ ]$

$$a \in A = [\ -45^\circ \quad 0^\circ \quad +45^\circ\ ] \tag{6}$$

After simulating this problem using the SDQL algorithm, the result is a determined best sample time and a control policy based on that sample time. The values for the individual sample times after 10,000 learning episodes are shown in Table 1.

Table 1. SDQL Robot Result

$T(sec)$	V_T
0.01	29.1
0.02	29.9
0.03	2.4
0.04	42.9
0.05	51.4
0.06	73.4
0.07	2271.8
0.08	14.7
0.09	29.2
0.10	120.8

The control policy has the ability to control the robot to move from randomly initialized points to the goal within a tolerance of $\pm 1m$. The sample time with the maximum value determined by V_T of $T = 0.07$ sec is in this simulation of the resulting Q function. Figures 1-6 demonstrate the ability to control the robot to the goal using this learned function for the determined sample time.

Figures 1-3 show that the robot is able to guide itself to the goal from an initial point in quadrant 1. The state time history shown in Figure 2 shows that the robot is able to guide itself there in under 8 seconds of simulated real-time. The commanded heading angle changes at each $T = 0.07$ sec are shown in Figure 3.

For further demonstration of the ability to control the robot, an initial condition beginning in quadrant 3 was tested. These results are shown in Figures 4-6. Figure 5 shows that the robot is able to reach the goal in this case in approximately 15 seconds, and the commanded heading angle history is shown in Figure 6.

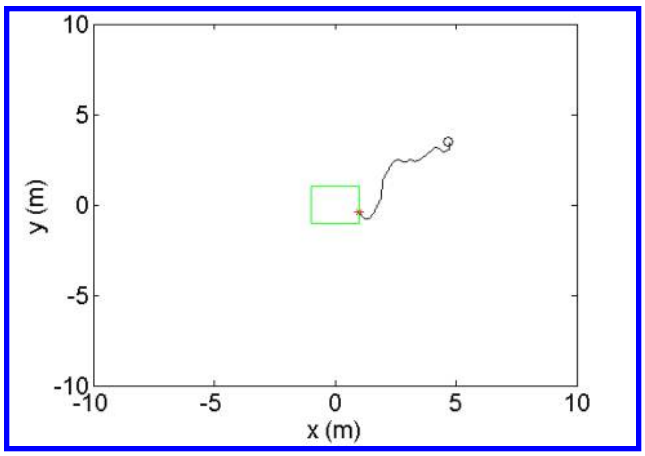

Figure 1. Simulation of Robot - Q1 Initial Condition

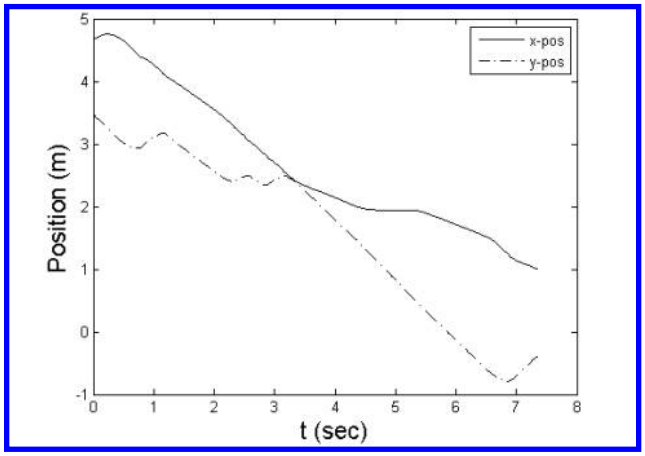

Figure 2. State History of Robot - Q1 Initial Condition

IV. System Dynamics Approximation

When using Q-learning to determine a control policy for dynamical systems, the benefit of not needing to have a model of the dynamics can lead to neglecting the dynamics completely. Learning a control policy for these systems is necessary, but often it is desired to also determine some approximation of the dynamics. This is especially important when dealing with the control of heterogeneous multi-agent systems since coordinating the agents requires knowing how the individual agents respond differently in time to similar action inputs.

A. First-Order Dynamics Learning

The simplest agent dynamics to represent are first-order dynamics. In a stable first-order system, the dynamics are described by the differential equation shown in Equation 7. Given the command value s_c, the time constant τ is needed to fully describe this differential equation.

$$\tau\dot{s} + s = s_c \tag{7}$$

The solution to this ODE is shown in Equation 8. This solution can be used to determine the next state, but τ is required to do so. This makes learning an approximate time constant all that is needed to determine the time behavior of the agent. Given the current state, s, the predicted next state, s^*, can be approximated using the current guess of the time constant, τ, and the sample time period, T, according to Equation 8.

$$s^* = se^{-T/\tau} + (1 - e^{-T/\tau})s_c \tag{8}$$

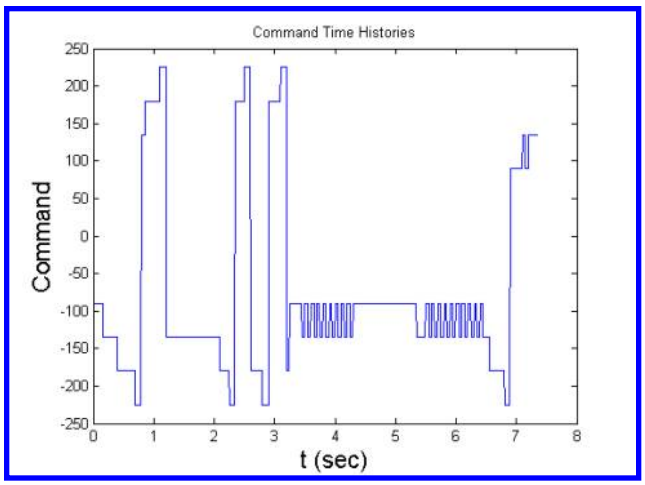

Figure 3. Command History of Robot - Q1 Initial Condition

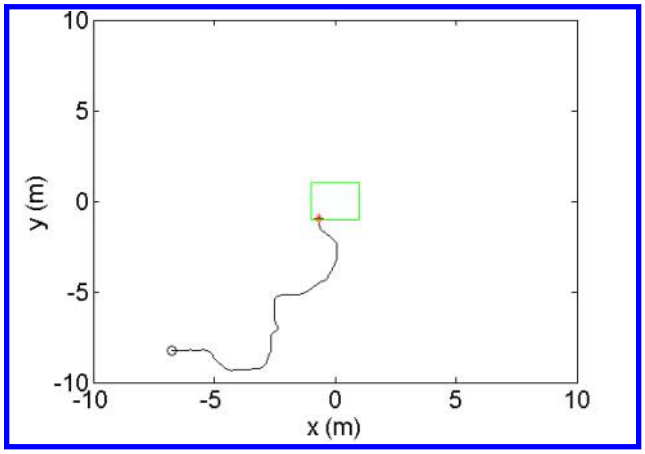

Figure 4. Simulation of Robot - Q3 Initial Condition

Equation 8 can be used to approximate a state transition for a single 1-dimensional state. Learning to approximate the time constant requires determining a reward that is a function of the current estimate of the sample time. By estimating the next state using Equation 8 and the current estimate of the time constant, the reward can be shaped by the error between the measured true next state and the estimated next state. The reward function used is shown in Equation 9.

$$r = ||s^* - s'||_2 \tag{9}$$

If more than one state dimension is to be approximated, multiple time constants would be needed. For instance, if this method were applied to a robot traversing a 2-D space with first-order dynamics in forward translation and rotation, one might want to know the state transition dynamics of both $\tau_{forward}$ and τ_{rotate} independently as they would most likely have different dynamics. Time constants for each state-action pair would be determined based on whether the robot were moving forward or rotating. To learn a time constant, a Monte Carlo learning formulation similar to the SDQL algorithm can be used. Algorithm 3 can shows how this can be accomplished.[14]

Algorithm 3 was used for the same example shown in Section III.B. After the 10,000 learning episodes completed, the FODL algorithm was able to successfully converge to the proper value of the time constant of $\tau = 0.2$ sec. Table 2 shows the V_1 values associated with each time constant, τ.

Over the course of the 10,000 learning episodes, the value associated with τ evolved as shown in Figure 7. As can be seen, the value associated with $\tau = 0.2$ became the maximum value early in the learning process. As learning episodes continue, the value function reinforces this as the best estimate of the time constant.

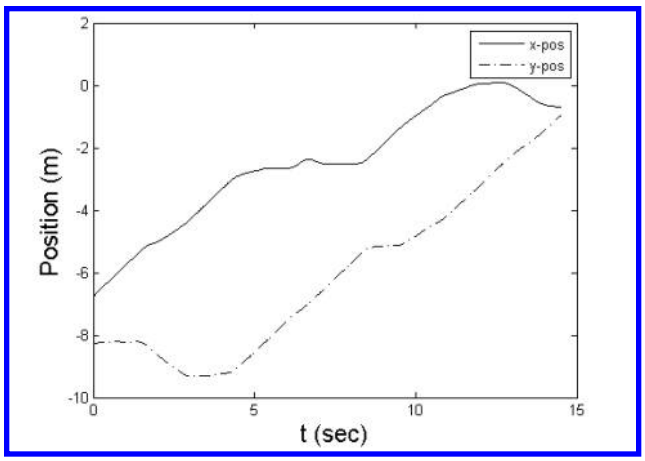

Figure 5. State History of Robot - Q3 Initial Condition

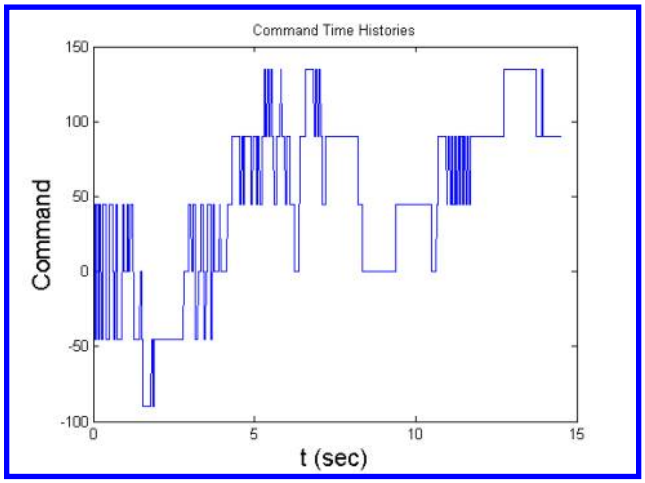

Figure 6. Command History of Robot - Q3 Initial Condition

B. Second-Order Dynamics Learning

In general, real world systems are more prone to exhibit second-order behavior rather than first-order. Second-order systems are the simplest systems that exhibit overshoot and oscillations, so higher-order systems can be approximated using second-order models.[15] It is therefore necessary to learn approximate models of second-order systems to approximate any systems that are higher than first-order. The second-order dynamics can be described by Equation 10.

$$\frac{d^2s}{dt^2} + 2\zeta\omega_n\frac{ds}{dt} + \omega_n^2 s = \omega_n^2 s_c \tag{10}$$

To approximate a second-order system requires determining 2 parameters: natural frequency and damping ratio. The learning framework used to determine these parameters is the same as for the first-order case, but the value function and the state approximation equations are different. The value function, here called V_2, is a function of the frequency and damping ratio. The state prediction equation is the solution to Equation 10 after a time period of T. This solution is dependent on the current approximation of damping ratio. For the case of $\zeta = 0$, the solution is as shown in Equation 11.

$$s^* = s_c + \frac{\dot{s}}{\omega_n}\sin(\omega_n T) + (s - s_c)\cos(\omega_n T) \tag{11}$$

For the case of $0 < \zeta < 1$, the solution is as shown in Equation 12.

Algorithm 3 First-Order Dynamics Learning (FODL)

- Determine T and $Q(s,a)$ for system (e.g., Sampled-Data Q-learning)
- Initialize $V_1(s,a,\tau)$ arbitrarily
- Repeat for each episode:
  - Initialize s, append T to s
  - Repeat for each timestep:
    - Choose a from s using greedy policy derived with $Q(s,a)$
    - Choose τ using policy derived from $V_1(s,a,\tau)$ (e.g., ε-Greedy)
    - Predict next state, s^*, with s,a, and τ using first-order approximations
    - Take action a, observe actual next state, s'
    - Observe r shaped from observed s' and predicted s^*
    - $V_1(s,a,\tau) \leftarrow V_1(s,a,\tau) + \alpha\ [\ r$ - $V_1(s,a,\tau)\ ]$
    - $s \leftarrow s'$
  - Until s is terminal

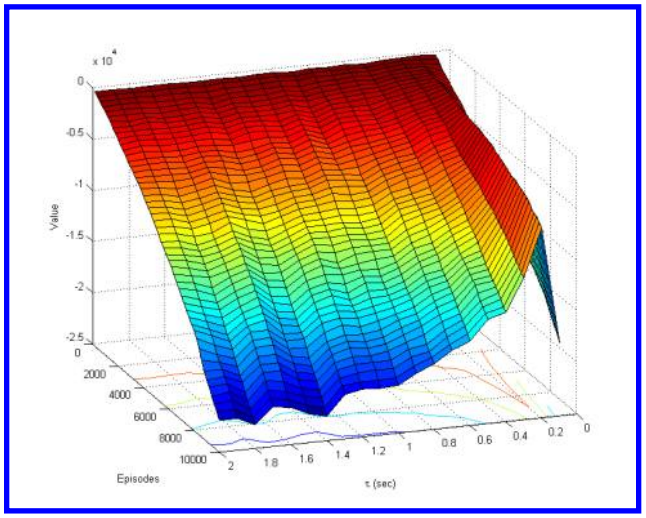

Figure 7. Time Constant Value History

$$s^* = s_c + \frac{\zeta}{\sqrt{1-\zeta^2}} e^{-\zeta\omega_n T}\left(s - s_c + \frac{\dot{s}}{\zeta\omega_n}\right)\sin(\omega_n T\sqrt{1-\zeta^2}) + e^{-\zeta\omega_n T}(s - s_c)\cos(\omega_n T\sqrt{1-\zeta^2}) \tag{12}$$

For the case of $\zeta = 1$, the solution is as shown in Equation 13.

$$s^* = s_c + (s - s_c)e^{-\omega_n T} + (\dot{s} + \omega_n s - \omega_n s_c)Te^{-\omega_n T} \tag{13}$$

And for the case of $\zeta > 1$, the solution is as shown in Equation 14.

Table 2. FODL Robot Value Function

$\tau(sec)$	V_1
0.1	-1.79×10^4
0.2	-0.65×10^4
0.3	-1.12×10^4
0.4	-1.44×10^4
0.5	-1.49×10^4
0.6	-1.68×10^4
0.7	-1.75×10^4
0.8	-1.86×10^4
0.9	-1.94×10^4
1.0	-2.03×10^4
1.1	-2.05×10^4
1.2	-2.03×10^4
1.3	-2.07×10^4
1.4	-2.26×10^4
1.5	-2.22×10^4
1.6	-2.14×10^4
1.7	-2.11×10^4
1.8	-2.27×10^4
1.9	-2.19×10^4
2.0	-2.19×10^4

$$s^* = s_c + \left(\frac{\dot{s} + (2\zeta\omega_n - z_1)s - s_c z_2}{z_2 - z_1} \right) e^{-z_1 T} - \left(\frac{\dot{s} + (2\zeta\omega_n + z_2)s - s_c z_1}{z_2 - z_1} \right) e^{-z_2 T} \tag{14}$$

where

$$z_1 = \omega_n(\zeta - \sqrt{\zeta^2 - 1})$$
$$z_2 = \omega_n(\zeta + \sqrt{\zeta^2 - 1})$$

To learn the second-order parameters, the FODL algorithm can be adjusted for learning the natural frequency and damping ratio rather than the time constant. This alternate version of dynamics learning, called Second-Order Dynamics Learning, is shown in Algorithm 4.

To demonstrate the SODL algorithm, the robot example from before was altered to have second-order dynamics in the heading angle equation of motion. For this example, the natural frequency of the robot heading angle was set to $\omega_n = 6$ rad/sec and the damping ratio is $\zeta = 0.8$. After 10,000 episodes of learning using Algorithm 4, the value of V_2 determined a maximum value associated with the correct frequency and damping ratio. Table 3 shows the final values, and they are plotted in Figure 8.

The way that the V_2 function is formulated implies that the value determined is for the *combination* of the variables ω and ζ rather than designating values for each separately. This is done because it is both of these variables together that determines the behavior of a system, and not either one separately. However, if one were to want to see how they are valued individually then the average values can be determined. If the value function entries for each instance of a particular ω are averaged together, an estimate of the value for that frequency is determined. Likewise, the same can be done for the damping ratio. Figures 9-10 show how the average values for the individual parameters evolve over time.

Algorithm 4 Second-Order Dynamics Learning (SODL)

- Determine T and $Q(s,a)$ for system (e.g., Sampled-Data Q-learning)
- Initialize $V_2(s,a,\omega_n,\zeta)$ arbitrarily
- Repeat for each episode:
  - Initialize s, append T to s
  - Repeat for each timestep:
    - Choose a from s using greedy policy derived with $Q(s,a)$
    - Choose ω_n and ζ using policy derived from $V_2(s,a,\omega_n,\zeta)$ (e.g., ε-Greedy)
    - Predict next state s^* with s, a, ω_n, and ζ using ζ-dependent solution
    - Take action a, observe actual next state s'
    - Observe r shaped from observed s' and predicted s^*
    - $V_2(s,a,\omega_n,\zeta) \leftarrow V_2(s,a,\omega_n,\zeta) + \alpha\ [\ r$ - $V_2(s,a,\omega_n,\zeta)\ ]$
    - $s \leftarrow s'$
  - Until s is terminal

Table 3. V_2 after 10,000 Episodes

	$\omega = 2$	$\omega = 4$	$\omega = 6$	$\omega = 8$	$\omega = 10$
$\zeta = 0.4$	−3220	−3107	−2412	−2671	−5130
$\zeta = 0.8$	−3450	−2610	−1466	−2606	−4110
$\zeta = 1.2$	−3283	−2883	−2393	−2323	−4092
$\zeta = 1.6$	−3008	−2855	−3180	−2940	−3155
$\zeta = 2.0$	−3240	−3602	−2795	−2576	−3484

V. Conclusions and Open Challenges

The work presented in this paper has shown that Reinforcement Learning-based techniques can be adapted to not only learn control policies for agents, but also learn an approximation of the agents' dynamics. Several conclusions can be drawn from these results. The Sampled-Data Q-learning algorithm is capable of determining longer sample times that still allow for successful control of the agent. The First-Order Dynamics Learning algorithm is capable of determining the time constants that best model the dynamics of the states for an individual agent with first-order dynamics. The Second-Order Dynamics Learning algorithm is capable of determining the best combination of natural frequency and damping ratio to model the second-order dynamics of the states for an agent.

There are a number of challenges for future research efforts. First, after learning the dynamics of agents it can be shown that in a hierarchical multiagent system the supervisory agents can use the dynamics information to determine commands to lower-level agents. Adaptation of these algorithms to stochastic systems is also possible. Another open problem is determining a means to either demonstrate the Second-Order Dynamics Learning algorithm's capability to approximate higher-order systems, or investigate alternate algorithms for approximating higher-order dynamics.

Acknowledgment

This work was sponsored (in part) by the Air Force Office of Scientific Research, USAF, under grant/contract number FA9550-08-1-0038. The technical monitor is Dr. Fariba Fahroo. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research or the U.S. Government.

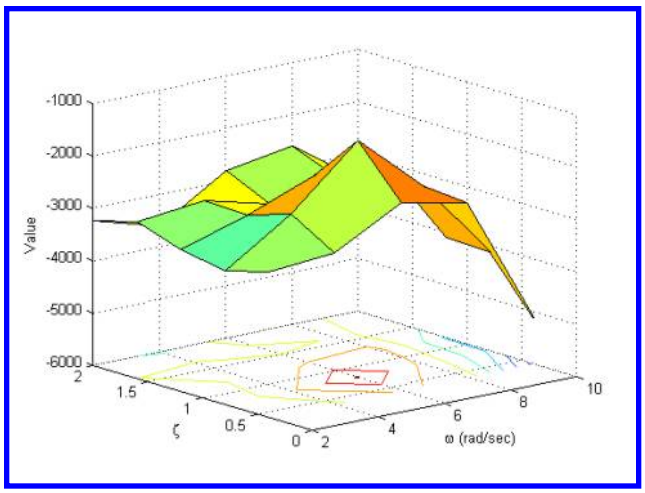

Figure 8. V_2 after 10,000 Episodes

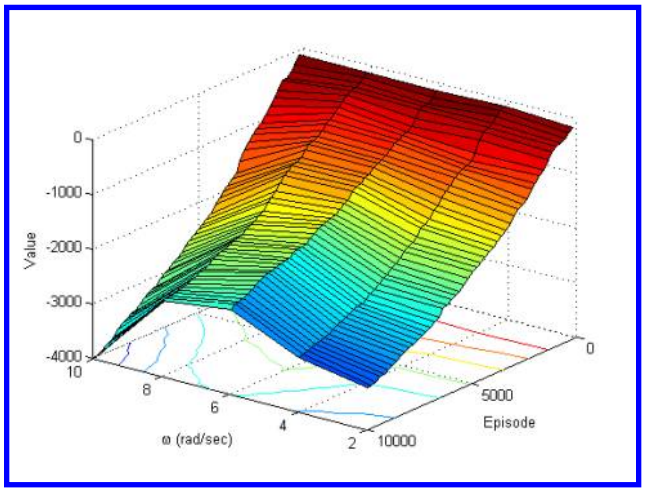

Figure 9. Natural Frequency Value History

References

[1]C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of National Conference on Artificial Intelligence*, AAAI-98, pages 746–752, 1998.

[2]J. Hu and M.P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the 15th International Conference on Machine Learning*, pages 242–250, Madison, WI, 1998.

[3]M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 535–542, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[4]S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, AAAI-02, pages 326–331, 2002.

[5]G. Tesauro. Extending q-learning to general adaptive multiagent systems. In *Advances in Neural Information Processing Systems*, volume 16, Vancouver and Whistler, Canada, 8-13 December 2003.

[6]R. Sutton and A. Barto. *Reinforcement Learning: An Introductions*. The MIT Press, Cambridge, MA, 1998.

[7]C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[8]K. Kirkpatrick and J. Valasek. Reinforcement learning for characterizing hysteresis behavior of shape memory alloys. *Journal of Aerospace Computing, Information, and Communication*, 6(3):227–238, March 2009.

[9]K. Kirkpatrick and J. Valasek. Active length control of shape memory alloy wires using reinforcement learning. *Journal of Intelligent Material Systems and Structures*, 22(14):1595–1604, September 2011.

[10]J. Valasek, M. Tandale, and J. Rong. A reinforcement learning - adaptive control architecture for morphing. *Journal of Aerospace Computing, Information, and Communication*, 2(5):174–195, April 2005.

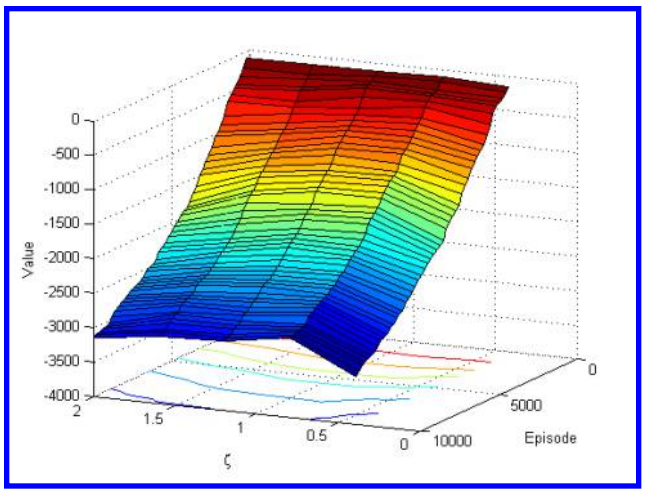

Figure 10. Damping Ratio Value History

[11] J. Valasek, J. Doebbler, M. D. Tandale, and A. J. Meade. Improved adaptive-reinforcement learning control for morphing unmanned air vehicles. *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, 38(4):1014–1020, August 2008.

[12] A. Lampton. *Discretization and Approximation Methods for Reinforcement Learning of Highly Reconfigurable Systems*. PhD thesis, Texas A&M University, October 2009.

[13] S. Whiteson, M. E. Taylor, and P. Stone. Empirical studies in action selection with reinforcement learning. *Adaptive Behavior*, 15:33–50, 2007.

[14] K. Kirkpatrick and J. Valasek. Reinforcement learning control with time dependent agent dynamics. In F. L. Lewis and D. Liu, editors, *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*. Wiley.

[15] Gene F. Franklin, Michael L. Workman, and Dave Powell. *Digital Control of Dynamic Systems*. 1997.